Analyzing Resilience to Computational Glitches in Island-Based Evolutionary Algorithms

Rafael Nogueras and Carlos Cotta(✉)

Dept. Lenguajes y Ciencias de la Computación,
Universidad de Málaga, ETSI Informática, Campus de Teatinos,
29071 Málaga, Spain
ccottap@lcc.uma.es

Abstract. We consider the deployment of island-based evolutionary algorithms (EAs) on irregular computational environments plagued with different kind of glitches. In particular we consider the effect that factors such as network latency and transient process suspensions have on the performance of the algorithm. To this end, we have conducted an extensive experimental study on a simulated environment in which the performance of the island-based EA can be analyzed and studied under controlled conditions for a wide range of scenarios in terms of both the intensity of glitches and the topology of the island-based model (scale-free networks and von Neumann grids are considered). It is shown that the EA is resilient enough to withstand moderately high latency rates and is not significantly affected by temporary island deactivations unless a fixed time-frame is considered. Combining both kind of glitches has a higher toll on performance, but the EA still shows resilience over a broad range of scenarios.

1 Introduction

The great success of metaheuristics in the last decades comes partly from the fact that their underlying algorithmic models are very much amenable to deployment in parallel and distributed environments [1]. Indeed, nowadays the use of parallel environments is a key factor to approach the resolution of complex computational problems, and population-based metaheuristics are ideal tools in this context. Specifically, evolutionary algorithms (EAs) have been used on this kind of setting since the 1980s with excellent results and can greatly benefit from parallelism [2,16]. Following this line, during the last years there has been a growing interest in the use of EAs in distributed computing environments that move away from classical dedicated networks so common in the past. Among such environments we can cite cloud computing [19], P2P networks [14,28], or volunteer computing (VC) [7], just to name a few.

Some of these emerging computational scenarios –in particular P2P and VC systems– are characterized by several distinctive features which can be summarized under the umbrella term of *irregularity*. Such irregularity is the result of

A. Auger et al. (Eds.): PPSN 2018, LNCS 11101, pp. 411–423, 2018.
https://doi.org/10.1007/978-3-319-99253-2_33

their being part of interconnected techno-social systems [26] composed of heterogeneous layers of resources with a complex dynamics. Thus, the computational substrate may be composed of a collection of computing nodes with heterogeneous capabilities [4,17], a feature that has to be accounted for, typically by finding an appropriate balancing of the computational load [6] or by distributing data appropriately [23]. The dynamism of the computational environment is another outstanding feature of these systems: computational nodes can have an uncontrollable dynamics caused by user interventions, interruptions of the network, eventual blockages, delays in communications, etc. The term *churn* is used to denote this phenomenon [24].

While sometimes it may be possible to try to hide these computational irregularities by adding intermediate layers, this can be a formidable challenge in multiple situations [8], and therefore algorithms may need being adapted to run natively (that is, irregularity-aware) on these computational systems. Focusing on EAs, these are fortunately very resilient, at least at a fine-grained scale – see [15,18]. Furthermore, in cases in which they can be more sensitive to environmental disruptions (e.g., in coarse-grain settings such as the island model [12]), they can be augmented with the necessary functionality to endure some of the difficulties caused by the irregularity of the computational substrate. In line with this, previous work has studied the resilience of EAs in scenarios plagued with instability and heterogeneity [21,22]. This does not exhaust the sources of irregularity though. Computational glitches can take place in additional different forms, such as traffic overloads or transient computational limitations, which to the best of our knowledge have not been analyzed in this context. Studying the performance of EAs in the presence of these is precisely the focus of this work. To this end we deploy an island-based EA on a simulated computational environment that allows experimenting in a controlled way with different intensities of such computational glitches, namely communication latency and temporary process deactivations. This will described in more detail in Sect. 2.

2 Methodology

As anticipated in previous section, we consider an island-based EA working on a simulated environment, in order to have control on the different issues under study. Each island of the EA runs on a computational node of this environment. In the following we will describe the basic algorithmic details of the EA, as well as how the network and the computational glitches are modeled.

Algorithmic Model. The algorithm considered is a steady-state EA with one-point crossover, bit-flip mutation, binary tournament selection and replacement of the worst parent. This algorithm is deployed on a computational environment in which each node hosts an island running an instance of the previously mentioned EA. After each iteration of the basic EA, these islands perform migration (stochastically with probability p_{mig}) of single individuals to neighboring islands.

Algorithm 1. Overview of the island-based evolutionary algorithm.

```
for i ∈ [1 ··· n_ι] do in parallel
    Initialize(pop_i) ;                          // initialize i-th island population
    buffer_i ← ∅ ;                               // initialize i-th migration buffer
end
while ¬ BudgetExhausted() do
    for i ∈ [1 ··· n_ι] do in parallel           // basic evolutionary cycle
        CheckMigrants (pop_i, buffer_i) ;        // accept migrants (if any)
        DoIteration(pop_i) ;                     // selection, reproduction, replacement
        if rand() < p_mig then
            for j ∈ N_i do
                SendMigrants(pop_i, buffer_j) ;  // send migrants
            end
        end
    end
end
```

In each migration event the migrant is randomly selected from the current population and the receiving island inserts it in its population by replacing the worst individual [20]. The whole process is illustrated in Algorithm 1.

Network Model. We assume a network composed by n_ι nodes interconnected following a certain topology. More precisely, we consider two possibilities for this purpose: a von Neumann (VN) grid and a scale-free (SF) network. The first one is a classical structure often used in spatially-structured EAs [10,25] and can be described as a regular toroidal mesh in which each node is connected to four neighbors (those located at unit Manhattan distance), see Fig. 1a. As to the second one, it is a complex network structure commonly found in many natural and artificial systems (e.g., P2P networks) as a consequence of their growth dynamics, i.e., their continuous expansion by the addition of new nodes that attach preferentially to sites that are already well connected [5]. The result is a network topology in which node degrees are distributed following a power-law (i.e., the fraction $p(d)$ of nodes with d neighbors goes as $p(d) \sim d^{-\gamma}$ for some constant parameter γ). To generate a network of this kind we use the Barabási-Albert model [3], whereby the network is grown from a clique of $m + 1$ nodes by adding a node at a time, connecting it to m of the nodes previously added (selected with probability proportional to their degree) where m is a parameter of the model. Figure 1b shows an example of a SF network. As anticipated by the power-law distribution of node degrees, a few nodes will have a large connectivity and increasingly more nodes will have a smaller number of neighbors.

Modeling Glitches. The functioning of the island-based EA described before is disturbed by the presence of perturbations of two types: (i) communication delays and (ii) temporary process deactivations. Both of them can have a diverse

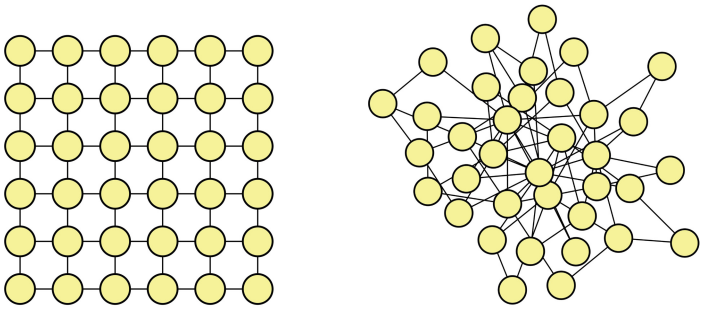

Fig. 1. (a) Example of a grid with von Neumann topology (toroidal links not shown for simplicity) (b) Example of a scale-free network with $m = 2$.

set of causes in real networks but in the specific context considered in this work, they can –from a very broad and abstract perspective – be considered to stem from the intrinsic properties of the underlying computational substrate, namely the fact it may be often composed of non-dedicated, low-end computational devices. Focusing firstly on network latency, it is a major issue on P2P systems: their decentralized nature makes them inherently more scalable than client-server architectures but also hampers effective communications due to bandwidth limitations and routing information maintenance overhead [13]. This can exert a strong influence on the performance of applications running on this kind of environments, e.g., [29]. To test the extent to which this factor also affects the performance of our island-based EA, we introduce a tunable delay in the communication between islands: whenever individuals are sent for migration purposes, they will only arrive to the destination island after some time λ, measured in a machine-independent way as a number of iterations of the basic evolutionary cycle in Algorithm 1.

The second factor considered is the temporary deactivation of a process. This can be due to a number of factors related to the way the operating system of a certain computational node schedules processes (e.g., the node can engage in swapping, or another high-priority process may kick in –recall we could be considering a VC scheme whereby our algorithm would be using just the spare CPU and bandwidth of a certain device– and the EA can be put to sleep). In such a case, we assume the computation process is still active but its execution is temporarily frozen. This means that it will not execute any evolutionary cycle nor it will send any individuals to neighboring islands (but it cannot prevent other islands from sending migrants to it; these migrants will be simply kept in the input buffer and processed later when the node wakes up). This is related to the issue of instability mentioned in Sect. 1 and can be considered as a slightly more benign form of churn, that is, the island is not completely lost as it would happen when a node goes out of the system and the process is terminated. In order to model this factor we need two parameters p_s and t_s: the first one indicates

the probability that each island is put to sleep in each iteration (assumed for simplicity to be constant and fixed for all islands), and the second one denotes the number of cycles it will remain in this dormant state.

3 Experimentation

We consider $n_\iota = 64$ islands of $\mu = 32$ individuals each, and a total number of evaluations $maxevals = 250,000$. We use crossover probability $p_X = 1.0$, mutation probability $p_M = 1/\ell$, where ℓ is the genotype length, and migration probability $p_{mig} = 1/(5\mu) = 1/160$. Regarding the network parameters, we use $m = 2$ in the Barabási-Albert model in order to define the topology of the SF network; in the case of the VN topology, we consider a 8×8 toroidal grid. As for the computational glitches, we consider the following settings:

- Latency values $\lambda = k\mu$ for $k \in \{0, 1, 2, 4, 8\}$. Intuitively, these values indicate a communication delay analogous to k full generations elapsed on an island.
- Node deactivations are done with values $p_s = k/(\mu n_\iota)$ and $t_s = k\mu$, with $k \in \{0, 1, 2, 4, 8, 16, 32\}$. Intuitively, a certain value of k would indicate both the average number of islands being deactivated per generation and the number of generations they would remain in that state.

The experimental benchmark comprises Deb's trap function [9] (TRAP, concatenating 32 four-bit traps), Watson et al.'s Hierarchical-if-and-only-if function [27] (HIFF, using 128 bits) and Goldberg et al.'s Massively Multimodal Deceptive Problem [11] (MMDP, using 24 six-bit blocks). These functions provide a scalable benchmark exhibitting properties of interest such as multimodality and deception. We perform 20 simulations for each configuration and measure performance as the percentage deviation from the optimal solution in each case.

Firstly, let us analyze how the latency of communications affects the performance of the algorithm. Figure 2 and Table 1 show the results. As expected, the performance of the algorithm degrades as the latency of communications increases (that is, as we move to the right along the X axis). This can be interpreted in terms of the role of migration: when individuals are migrated the receiving island can benefit both from increased diversity and from quality genetic material. In fact these two factors are intertwined since good (in terms of fitness) fresh information is more likely to proliferate in the target population, and can hence re-focus the search conducted in the island or contribute to drive it out of stagnating states. To the extent that this information starts to constitute a glimpse from the past (as it happens when the latency is in the upper range of values considered), the migrants tend to be less significant in terms of fitness (since the receiving island has more time to evolve and advance in the mean time). They will still carry diversity (and actually in some cases this diversity might be probably higher in comparative terms, since the emitting island was in a less-converged state), but the impact on the receiving island will be less marked, at least in the cases in which latency is high (cf. Table 1), without excluding that for the lower range of latency values considered this diversity

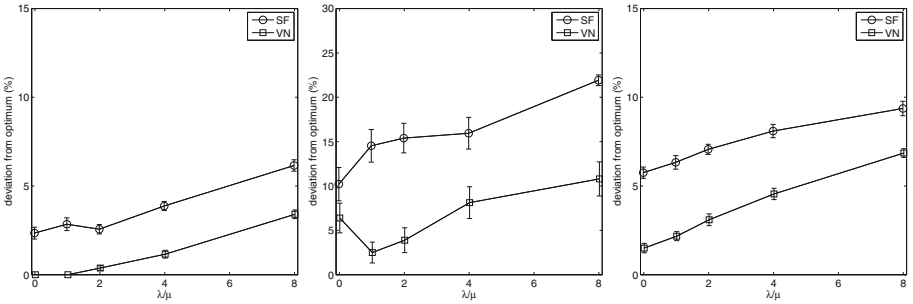

Fig. 2. Average deviation from the optimal solution as a function of the latency parameter for SF and VN topologies. (a) TRAP (b) HIFF (c) MMDP.

Table 1. Results (20 runs) of the different EAs on SF (upper portion of the table) and VN (lower portion of the table) networks for different latency values. In this table and subsequent ones, the median ($\tilde{x}$), mean ($\bar{x}$) and standard error of the mean ($\sigma_{\bar{x}}$) are indicated. A symbol $\star|\bullet|\circ$ is used to indicate statistically significant differences at $\alpha = 0.01|0.05|0.10$ with respect to the case $\lambda = 0$ according to a Wilcoxon ranksum test.

SF	TRAP			H-IFF			MMDP		
Latency (λ)	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
0	2.50	2.34 ± 0.33		11.11	10.21 ± 1.87		5.99	5.75 ± 0.32	
μ	2.50	2.84 ± 0.36		16.67	14.53 ± 1.83	$\circ$	5.99	6.33 ± 0.38	
2μ	2.50	2.56 ± 0.26		16.67	15.40 ± 1.67	$\circ$	7.49	7.06 ± 0.29	$\bullet$
4μ	3.75	3.88 ± 0.25	$\star$	19.44	15.95 ± 1.79	$\bullet$	7.49	8.10 ± 0.38	$\star$
8μ	6.25	6.16 ± 0.32	$\star$	21.88	21.92 ± 0.60	$\star$	8.99	9.37 ± 0.41	$\star$
VN	TRAP			H-IFF			MMDP		
Latency (λ)	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
0	0.00	0.00 ± 0.00		0.00	6.39 ± 1.68		1.50	1.50 ± 0.27	
μ	0.00	0.00 ± 0.00		0.00	2.50 ± 1.17	$\circ$	1.50	2.17 ± 0.25	
2μ	0.00	0.37 ± 0.13	$\star$	0.00	3.89 ± 1.40		3.00	3.10 ± 0.33	$\star$
4μ	1.25	1.16 ± 0.22	$\star$	11.11	8.13 ± 1.79		4.49	4.55 ± 0.32	$\star$
8μ	3.75	3.41 ± 0.24	$\star$	13.89	10.80 ± 1.92	$\circ$	7.32	6.86 ± 0.24	$\star$

boost can sometimes provide a minor improvement. The results are also qualitatively similar for both network topologies in which the degradation trend is analogous.

Let us now turn our attention to the effect of temporary island deactivations. The results for different intensities of this factor are shown in Table 2. As it can be seen, there is hardly a degradation of results even for large glitch rates. To interpret this, notice that the presence of dormant islands resembles

Table 2. Results (20 runs) of the different EAs on SF (upper portion of the table) and VN (lower portion of the table) networks for different deactivation parameters and a constant number of evaluations. The statistical comparison is done with respect to the case $k = 0$.

SF	TRAP			H-IFF			MMDP		
k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
0	2.50	2.34 ± 0.33		11.11	10.21 ± 1.87		5.99	5.73 ± 0.32	
1	1.25	1.94 ± 0.27		16.67	13.11 ± 1.98		4.49	5.00 ± 0.36	
2	2.50	2.37 ± 0.28		13.89	11.08 ± 1.78		5.99	5.82 ± 0.37	
4	2.50	2.34 ± 0.29		16.67	14.57 ± 1.35		5.99	5.90 ± 0.39	
8	2.50	2.88 ± 0.41		16.67	15.07 ± 1.54	•	7.32	6.86 ± 0.34	•
16	2.50	2.66 ± 0.36		19.44	18.44 ± 1.25	⋆	5.99	6.11 ± 0.36	
32	2.50	2.22 ± 0.28		16.67	16.05 ± 1.71	•	5.99	6.03 ± 0.35	
VN	TRAP			H-IFF			MMDP		
k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
0	0.00	0.00 ± 0.00		0.00	6.39 ± 1.68		1.50	1.50 ± 0.27	
1	0.00	0.00 ± 0.00		0.00	3.06 ± 1.24		1.50	1.26 ± 0.29	
2	0.00	0.06 ± 0.06		0.00	6.10 ± 1.79		1.50	1.56 ± 0.22	
4	0.00	0.12 ± 0.09		0.00	5.42 ± 1.81		1.50	1.48 ± 0.28	
8	0.00	0.06 ± 0.06		11.11	8.06 ± 1.59		1.50	1.95 ± 0.31	
16	0.00	0.25 ± 0.11	•	11.11	7.15 ± 1.58		1.50	1.91 ± 0.30	
32	0.00	0.25 ± 0.11	•	11.11	8.89 ± 1.76		2.83	2.36 ± 0.22	•

transient heterogeneous computational capabilities: within a small time window, each island will have conducted a different number of cycles, which is to some extent analogous to assume they are running on nodes with different computational power; however, on the larger scale, these dormant periods distribute rather homogeneously over all islands, and thus they advance on average at the same rate. Of course, these perturbations become better smoothed out in the longer term the finer they are (hence some effects can be observed in the upper range of values of k, where glitches are more coarse-grained), but EAs are in any case resilient enough to withstand heterogeneous advance rates without dramatic performance losses [22].

A different perspective can be obtained if we approach these results from the point of view of a fixed time frame, as opposed to a fixed computational effort distributed over a variable time frame. Obviously, the presence of dormant islands contributes to dilute the computational effort exerted over a certain time frame, so studying the resilience of the EA to this dilution is in order. To do so, we consider a time frame dictated by the number of cycles performed by the EA in the base ($k = 0$) case. The results under these conditions are shown in Table 3 and Fig. 3. As expected there is a clear trend of degradation in this case.

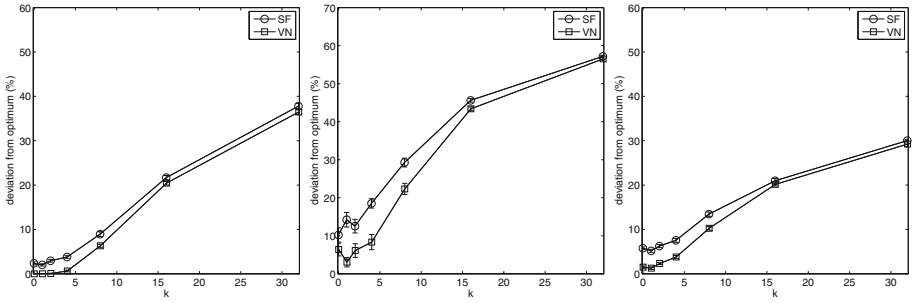

Fig. 3. Average deviation from the optimal solution as a function of the deactivation parameters for SF and VN topologies and a constant number of cycles. (a) TRAP (b) HIFF (c) MMDP.

Table 3. Results (20 runs) of the different EAs on SF (upper portion of the table) and VN (lower portion of the table) networks for different deactivation parameters and a constant number of cycles. The statistical comparison is done with respect to the case $k = 0$.

SF	TRAP			H-IFF			MMDP		
k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
0	2.50	2.34 ± 0.33		11.11	10.21 ± 1.87		5.99	5.75 ± 0.32	
1	1.25	2.00 ± 0.25		16.67	14.21 ± 1.91		5.24	5.17 ± 0.35	
2	2.81	2.94 ± 0.30		16.67	12.53 ± 1.80		5.99	6.24 ± 0.34	
4	3.75	3.78 ± 0.30	⋆	19.44	18.51 ± 1.23	⋆	7.49	7.54 ± 0.43	⋆
8	9.69	8.94 ± 0.53	⋆	30.38	29.26 ± 1.15	⋆	13.48	13.45 ± 0.50	⋆
16	21.56	21.69 ± 0.47	⋆	45.31	45.62 ± 0.46	⋆	21.39	20.98 ± 0.49	⋆
32	38.44	37.81 ± 0.60	⋆	57.47	57.20 ± 0.29	⋆	30.29	30.03 ± 0.27	⋆
VN	TRAP			H-IFF			MMDP		
k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
0	0.00	0.00 ± 0.00		0.00	6.39 ± 1.68		1.50	1.50 ± 0.27	
1	0.00	0.00 ± 0.00		0.00	3.06 ± 1.24		1.50	1.27 ± 0.29	
2	0.00	0.06 ± 0.06		0.00	6.11 ± 1.80		3.00	2.32 ± 0.23	•
4	0.00	0.62 ± 0.19	⋆	5.56	8.33 ± 2.01		3.00	3.79 ± 0.33	⋆
8	6.25	6.31 ± 0.33	⋆	22.05	22.32 ± 1.48	⋆	10.15	10.26 ± 0.41	⋆
16	20.63	20.44 ± 0.41	⋆	43.40	43.40 ± 0.49	⋆	20.31	20.15 ± 0.32	⋆
32	37.19	36.47 ± 0.52	⋆	56.60	56.53 ± 0.32	⋆	29.29	29.25 ± 0.53	⋆

Still, the EA can withstand scenarios in which islands remain deactivated for a number of cycles equivalent to twice the population size, although performance significantly degrades for larger deactivation rates/periods in which a much more significant part of the computational effort is lost (up from about 25% for $k = 4$).

Table 4. Results (20 runs) of the different EAs on SF networks for different latency and deactivation parameters and a constant number of cycles. Three symbols are shown next to each entry indicating from left to right statistical comparisons with respect to (i) $\lambda = 0$, $k = 0$, (ii) same λ and $k = 0$, and (iii) same k and $\lambda = 0$ using a Wilcoxon ranksum test. Blanks indicate no statistically significant difference for the corresponding comparison, and $\star|\bullet|\circ$ have the same meaning as in Tables 1, 2 and 3.

SF		TRAP			H-IFF			MMDP		
λ	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
0	0	2.50	2.34 ± 0.33		11.11	10.21 ± 1.87		5.99	5.75 ± 0.32	
	1	1.25	2.00 ± 0.25		16.67	14.21 ± 1.91		5.24	5.17 ± 0.35	
	2	2.50	2.41 ± 0.28		19.44	14.68 ± 1.99	●●	5.99	5.23 ± 0.36	
	4	3.13	3.00 ± 0.33		19.44	15.97 ± 1.81	●●	5.99	6.33 ± 0.31	
	8	2.50	2.81 ± 0.35		19.44	17.91 ± 1.34	⋆⋆	7.49	7.44 ± 0.46	⋆⋆
μ	0	2.50	2.84 ± 0.36		16.67	14.53 ± 1.83	○ ○	5.99	6.33 ± 0.38	
	1	3.75	3.22 ± 0.34	●	16.67	16.58 ± 1.62	●	6.57	6.47 ± 0.40	●
	2	2.50	2.25 ± 0.32		16.67	14.86 ± 1.34	○	6.57	6.39 ± 0.43	○
	4	2.50	2.66 ± 0.22		18.06	15.00 ± 1.89	●	5.99	6.33 ± 0.35	
	8	3.44	2.97 ± 0.30		19.44	17.58 ± 1.62	⋆	8.07	7.56 ± 0.50	⋆○
2μ	0	2.50	2.56 ± 0.26		16.67	15.40 ± 1.67	○ ○	7.49	7.06 ± 0.29	● ●
	1	3.44	3.09 ± 0.30	●	16.67	13.69 ± 1.94		7.49	7.16 ± 0.34	⋆ ⋆
	2	2.81	3.19 ± 0.23	○ ●	18.06	17.14 ± 1.58	⋆	5.99	6.41 ± 0.28	●
	4	4.37	3.97 ± 0.30	⋆⋆●	18.06	15.00 ± 2.14	●	7.49	7.27 ± 0.35	⋆ ○
	8	4.06	3.69 ± 0.41	●●	20.14	18.81 ± 1.35	⋆○	7.49	7.45 ± 0.41	⋆
4μ	0	3.75	3.88 ± 0.25	⋆ ⋆	19.44	15.95 ± 1.79	● ●	7.49	8.10 ± 0.38	⋆ ⋆
	1	4.37	4.16 ± 0.25	⋆ ⋆	19.44	17.66 ± 1.16	⋆	7.49	7.59 ± 0.32	⋆ ⋆
	2	4.06	4.31 ± 0.32	⋆ ⋆	20.14	18.23 ± 1.16	⋆	8.99	8.26 ± 0.30	⋆ ⋆
	4	4.69	4.53 ± 0.39	⋆ ⋆	20.49	17.41 ± 1.61	⋆	8.66	8.30 ± 0.42	⋆ ⋆
	8	5.00	4.81 ± 0.24	⋆●⋆	20.83	18.17 ± 1.64	⋆	8.82	8.25 ± 0.27	⋆
8μ	0	6.25	6.16 ± 0.32	⋆ ⋆	21.88	21.92 ± 0.60	⋆ ⋆	8.99	9.37 ± 0.41	⋆ ⋆
	1	5.94	5.75 ± 0.33	⋆ ⋆	21.70	22.35 ± 0.81	⋆ ⋆	10.15	9.79 ± 0.30	⋆ ⋆
	2	5.94	6.09 ± 0.26	⋆ ⋆	22.14	22.47 ± 0.87	⋆ ⋆	9.57	9.59 ± 0.33	⋆ ⋆
	4	6.25	6.25 ± 0.38	⋆ ⋆	21.88	22.76 ± 0.78	⋆ ⋆	10.48	10.51 ± 0.36	⋆●⋆
	8	5.00	5.47 ± 0.35	⋆○⋆	25.00	23.53 ± 1.52	⋆●⋆	10.48	10.58 ± 0.48	⋆○⋆

Finally, let us consider the cross-effect of having both types of computational glitches. To this end, we have focused on the lower range of deactivation rates ($0 \leqslant k \leqslant 8$) in which degradation is moderate at most, leaving aside parameter settings for which extreme degradation already takes place on its own. Also, we have fixed $t_s = \mu$ in order to have more fine-grained island deactivations and isolate the analysis on the interplay between p_s and λ. The results are shown

Table 5. Results (20 runs) of the different EAs on VN grids for different latency and deactivation parameters and a constant number of cycles. Statistical comparisons follow the same notation as in Table 4.

VN		TRAP			H-IFF			MMDP		
λ	k	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
0	0	0.00	0.00 ± 0.00		0.00	6.39 ± 1.68		1.50	1.50 ± 0.27	
	1	0.00	0.00 ± 0.00		0.00	3.06 ± 1.24		1.50	1.27 ± 0.29	
	2	0.00	0.00 ± 0.00		0.00	3.33 ± 1.36		1.50	1.76 ± 0.25	
	4	0.00	0.12 ± 0.09		0.00	4.72 ± 1.35		1.50	1.78 ± 0.23	
	8	0.00	0.19 ± 0.10	○ ○	11.11	8.06 ± 1.42		3.00	3.13 ± 0.25	⋆⋆
μ	0	0.00	0.00 ± 0.00		0.00	2.50 ± 1.17	○○	1.50	2.17 ± 0.25	
	1	0.00	0.25 ± 0.11	●●●	0.00	5.28 ± 1.37		3.00	2.77 ± 0.27	⋆○⋆
	2	0.00	0.19 ± 0.10	○○○	5.56	5.56 ± 1.27	○	2.83	2.42 ± 0.22	● ○
	4	0.00	0.28 ± 0.13	●●	0.00	5.56 ± 1.61		3.00	2.50 ± 0.24	● ●
	8	0.00	0.37 ± 0.13	⋆⋆	0.00	4.03 ± 1.63	○	3.00	3.56 ± 0.31	⋆⋆
2μ	0	0.00	0.37 ± 0.13	⋆ ⋆	0.00	3.89 ± 1.40		3.00	3.10 ± 0.33	⋆ ⋆
	1	0.00	0.44 ± 0.14	⋆ ⋆	0.00	4.44 ± 1.43		3.00	3.05 ± 0.25	⋆ ⋆
	2	0.00	0.50 ± 0.14	⋆ ⋆	0.00	3.33 ± 1.36		3.00	3.52 ± 0.29	⋆ ⋆
	4	0.00	0.56 ± 0.14	⋆ ●	5.56	6.39 ± 1.52		3.00	3.34 ± 0.33	⋆ ⋆
	8	1.25	0.69 ± 0.14	⋆ ⋆	0.00	5.56 ± 1.45		4.49	4.68 ± 0.31	⋆⋆⋆
4μ	0	1.25	1.16 ± 0.22	⋆ ⋆	11.11	8.13 ± 1.79		4.49	4.55 ± 0.32	⋆ ⋆
	1	1.25	1.34 ± 0.21	⋆ ⋆	0.00	2.22 ± 1.24	○●	4.49	4.89 ± 0.26	⋆ ⋆
	2	1.25	0.97 ± 0.16	⋆ ⋆	0.00	3.33 ± 1.36	●	5.66	5.35 ± 0.30	⋆ ⋆
	4	1.25	1.50 ± 0.19	⋆ ⋆	0.00	1.88 ± 1.30	●⋆○	4.49	4.64 ± 0.26	⋆ ⋆
	8	1.56	1.81 ± 0.23	⋆●⋆	5.56	7.57 ± 1.83		5.99	6.11 ± 0.32	⋆⋆⋆
8μ	0	3.75	3.41 ± 0.24	⋆ ⋆	13.89	10.80 ± 1.92	○ ○	7.32	6.86 ± 0.24	⋆ ⋆
	1	3.75	3.66 ± 0.28	⋆ ⋆	13.89	12.26 ± 1.80	● ⋆	7.49	7.51 ± 0.27	⋆ ⋆
	2	3.75	3.56 ± 0.29	⋆ ⋆	16.67	13.56 ± 1.91	⋆ ⋆	7.49	7.70 ± 0.24	⋆⋆⋆
	4	3.75	3.66 ± 0.27	⋆ ⋆	19.44	15.78 ± 1.69	⋆●⋆	7.49	7.41 ± 0.29	⋆ ⋆
	8	4.37	4.47 ± 0.26	⋆⋆⋆	16.67	14.51 ± 1.79	⋆ ⋆	8.66	8.44 ± 0.43	⋆⋆⋆

in Tables 4 and 5. As it can be seen, both factors strongly interact in degrading performance: if we inspect the first block in either table (corresponding to having no latency) we observe that performance differences only start to become significant for larger values of p_s; however, remaining blocks are plagued with significant performance differences, even for small values of p_s. Furthermore, we can see that having $p_s > 0$ can provoke significant differences in scenarios in which the mere presence of latency would not suffice, cf. Table 1. It is nevertheless interesting to observe that this latter factor, namely latency, seems to have a stronger influence in the performance of the algorithm in this scenario,

as indicated by the fact that turning off deactivations for a given latency value does not usually provide a significant difference (that is, unless the former is typically in the upper end of its range) whereas the converse is often the case.

4 Conclusions

Resilience is a property that any algorithm running on an irregular computational environment should feature. Evolutionary algorithms are in this sense well-prepared thanks to the intrinsic resilience provided by their population-based nature. In particular, we have shown in this work that an island-based EA can withstand significant computational glitches without major performance losses. Indeed, the range of latency values and deactivation rates for which noticeable degradation takes place can be considered at the very least moderately high (e.g., latency values larger than a couple of generations of the EA). This complements previous findings that showed both the sensitivity of these techniques to more serious disruptions (such as node failures) and their amenability for being endowed with mechanisms to endure such severe glitches. In this sense, it would be of the foremost interest to study harder scenarios integrating node failures with the computational perturbations considered in this work, analyzing how the EA can react to the corresponding variety of fluctuations in the computational landscape. Such a study could certainly encompass other algorithmic variants of EAs, as well as additional network topologies. The study could be also conducted along other dimensions such as the effect that the migration probability can have in order to counteract glitches.

Acknowledgements. This work is supported by the Spanish Ministerio de Economía and European FEDER under Projects EphemeCH (TIN2014-56494-C4-1-P) and DeepBIO (TIN2017-85727-C4-1-P) and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

References

1. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley, Hoboken (2005)
2. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Trans. Evol. Comput. **6**(5), 443–462 (2002)
3. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Rev. Mod. Phys. **74**(1), 47–97 (2002)
4. Anderson, D.P., Reed, K.: Celebrating diversity in volunteer computing. In: Proceedings of the 42nd Hawaii International Conference on System Sciences, HICSS 2009, pp. 1–8. IEEE Computer Society, Washington (2009)
5. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)
6. Beltrán, M., Guzmán, A.: How to balance the load on heterogeneous clusters. Int. J. High Perform. Comput. Appl. **23**, 99–118 (2009)

7. Cole, N.: Evolutionary algorithms on volunteer computing platforms: the Milky-Way@Home project. In: de Vega, F.F., Cantú-Paz, E. (eds.) Parallel and Distributed Computational Intelligence. Studies in Computational Intelligence, vol. 269, pp. 63–90. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-10675-0_4
8. Cotta, C., et al.: Ephemeral computing and bioinspired optimization - challenges and opportunities. In: 7th International Joint Conference on Evolutionary Computation Theory and Applications, pp. 319–324. SCITEPRESS, Lisboa, Portugal (2015)
9. Deb, K., Goldberg, D.: Analyzing deception in trap functions. In: Whitley, L. (ed.) Second Workshop on Foundations of Genetic Algorithms, pp. 93–108. Morgan Kaufmann Publishers, Vail (1993)
10. Dorronsoro, B., Alba, E.: Cellular Genetic Algorithms Operations Research/ Computer Science Interfaces, vol. 42. Springer, Heidelberg (2008). https://doi.org/10.1007/978-0-387-77610-1
11. Goldberg, D., Deb, K., Horn, J.: Massive multimodality, deception and genetic algorithms. In: Männer, R., Manderick, B. (eds.) Parallel Problem Solving from Nature - PPSN II, pp. 37–48. Elsevier Science Inc., New York (1992)
12. Hidalgo, J., Lanchares, J., Fernández de Vega, F., Lombraña, D.: Is the island model fault tolerant? In: Thierens, D., et al. (eds.) Genetic and Evolutionary Computation - GECCO 2007, pp. 2737–2744. ACM Press, New York (2007)
13. Kumar, P., Sridhar, G., Sridhar, V.: Bandwidth and latency model for DHT based peer-to-peer networks under variable churn. In: 2005 Systems Communications (ICW 2005, ICHSN 2005, ICMCS 2005, SENET 2005), pp. 320–325. IEEE August 2005
14. Laredo, J., Castillo, P., Mora, A., Merelo, J.J.: Evolvable agents, a fine grained approach for distributed evolutionary computing: walking towards the peer-to-peer computing frontiers. Soft Comput. **12**(12), 1145–1156 (2008)
15. Laredo, J., Castillo, P., Mora, A., Merelo, J.J., Fernandes, C.: Resilience to churn of a peer-to-peer evolutionary algorithm. Int. J. High Perform. Syst. Archit. **1**(4), 260–268 (2008)
16. Lässig, J., Sudholt, D.: General scheme for analyzing running times of parallel evolutionary algorithms. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) Parallel Problem Solving from Nature - PPSN XI, pp. 234–243. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15844-5_24
17. Lastovetsky, A.: Heterogeneous parallel computing: from clusters of workstations to hierarchical hybrid platforms. Supercomput. Front. Innovations **1**(3), 70–87 (2014)
18. Lombraña González, D., Fernández de Vega, F., Casanova, H.: Characterizing fault tolerance in genetic programming. Future Generation Computer Systems **26**(6), 847–856 (2010)
19. Meri, K., Arenas, M., Mora, A., Merelo, J.J., Castillo, P., García-Sánchez, P., Laredo, J.: Cloud-based evolutionary algorithms: an algorithmic study. Nat. Comput. **12**(2), 135–147 (2013)
20. Nogueras, R., Cotta, C.: An analysis of migration strategies in island-based multimemetic algorithms. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 731–740. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_72
21. Nogueras, R., Cotta, C.: Self-healing strategies for memetic algorithms in unstable and ephemeral computational environments. Nat. Comput. **16**(2), 189–200 (2017)

22. Nogueras, R., Cotta, C.: Analyzing self-⋆ island-based memetic algorithms in heterogeneous unstable environments. Int. J. High Perform. Comput., Appl (2016). https://doi.org/10.1177/1094342016678665
23. Renard, H., Robert, Y., Vivien, F.: Data redistribution algorithms for heterogeneous processor rings. Int. J. High Perform. Comput. Appl. **20**, 31–43 (2006)
24. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: 6th ACM SIGCOMM Conference on Internet Measurement - IMC 2006, pp. 189–202. ACM Press, New York (2006)
25. Tomassini, M.: Spatially Structured Evolutionary Algorithms Natural Computing Series. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-29938-6
26. Vespignani, A.: Predicting the behavior of techno-social systems. Science **325**(5939), 425–428 (2009)
27. Watson, R.A., Hornby, G.S., Pollack, J.B.: Modeling building-block interdependency. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 97–106. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0056853
28. Wickramasinghe, W., Steen, M.V., Eiben, A.E.: Peer-to-peer evolutionary algorithms with adaptive autonomous selection. In: Thierens, D. (ed.) Genetic and Evolutionary Computation - GECCO 2007, pp. 1460–1467. ACM Press, New York (2007)
29. Zhou, J., Tang, L., Li, K., Wang, H., Zhou, Z.: A low-latency peer-to-peer approach for massively multiplayer games. In: Despotovic, Z., Joseph, S., Sartori, C. (eds.) AP2PC 2005. LNCS (LNAI), vol. 4118, pp. 120–131. Springer, Heidelberg (2006). https://doi.org/10.1007/11925941_10