# Enhancing Evolutionary Optimization Performance under Byzantine Fault Conditions[*]

Carlos Cotta[1,2][0000−0001−8478−7549]

[1] Dept. Lenguajes y Ciencias de la Computación, ETSI Informática, Campus de Teatinos, Universidad de Málaga, 29071 Málaga, Spain
[2] ITIS Software, Universidad de Málaga, Spain
ccottap@lcc.uma.es

**Abstract.** We evaluate the performance of panmictic evolutionary algorithms (EAs) in Byzantine environments, where fitness values are unreliable due to the potential presence of malicious agents. We investigate the impact of this phenomenon on the performance of the algorithm considering two different models of malicious behavior of different severity, taking the unreliability rate of the environment as a control parameter. We observe how there can be a significant toll in the quality of the results as the prevalence of cheating behavior increases, even for simple functions. Subsequently, we endow the EA with mechanisms based on redundant computation to cope with this issue, and examine their effectiveness. Our findings indicate that while a mechanism based on statistical averaging can be an effective approach under a relatively benign fault model, more hostile environments are better tackled via an approach based on majority voting.

**Keywords:** Evolutionary algorithms · Byzantine faults · Panmixia · Resilience

## 1 Introduction

Emerging computational environments such as peer-to-peer networks and volunteer computing (VC) platforms [10,11] have become promising ecosystems for running computationally intensive tasks. This is for example the case of evolutionary algorithms (EAs) [7,21]. Needless to say, when deploying any complex computing task on this kind of environments numerous challenges arise due to the dynamic nature and irregularity of the resulting computational landscape [1], thus highlighting the need for algorithmic resilience. Fortunately, EAs are not just inherently resilient [5] but also flexible enough to incorporate mechanisms to cope with perturbations caused by the volatility or heterogeneity of the environment [15,16].

[*] This work is supported by Spanish Ministry of Science and Innovation under project Bio4Res (PID2021-125184NB-I00 – http://bio4res.lcc.uma.es) and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

This work is primarily concerned with another source of disruption not related to the irregularities of the computational substrate but to logical failures of malicious nature [19]. Specifically, we consider *cheating faults*, a kind of Byzantine failure whereby one or more contributors of computational resources do not provide trustworthy results but do however purposefully alter the computation by submitting wrong results. This can be done with the mere purpose of feigning an activity (for instance, in order to obtain any rewards associated with the participation in the VC platform) or even with the malicious aim of damaging the computation itself. Previous research has shown that these faults can have an impact on EAs which will depend on the precise components of the algorithm targeted or affected by such faults [13]. Distributed EAs can retain global asymptotic convergence under some conditions [20] (see also [12]), and EAs with fine-grained spatial structure can withstand certain mild types of Byzantine faults [14]. More hostile faults seem to quickly degrade the performance of EAs on some problems though [3], when the evaluation of fitness is targeted by malicious agents.

Our analysis in this work aims to examine this performance drop more qualitatively, and to study some countermeasures to cope with these faults. To this end, we will start by providing a description of the algorithmic setting in which Byzantine faults can take place (Sect. 2). Then, two types of Byzantine faults of different severity are considered, as described in Sect. 2.1, and two strategies for coping with these are provided in Sect. 2.2, making use of redundant computation in order to handle the uncertainty in fitness values associated to such faults. Subsequently, we report the experimental results attained when using either of these strategies in Sect. 3. We close the paper with an outlook of these results and an outline of future work in Sect. 4.

## 2 Algorithmic Setting

Let us consider an EA with a panmictic population aiming to optimize a certain objective function $f(\cdot)$, which we shall term the *true fitness* function. In order to evaluate this function for the individuals in the population, the EA relies on a number of *helpers*. For instance, this is consistent with the use of master-slave models [2], whereby individuals are distributed among a collection of computational nodes that provide this fitness evaluation service. These helpers may vary dynamically and are not directly traceable by the EA, e.g., imagine they are behind some cloud or VC service layer, and hence the EA as a client has no control –nor even knowledge– on where the computation is done (i.e., it cannot pinpoint the particular source of each fitness value computed either). Now, the issue under study here arises when some of these helpers are *cheaters*, which provide wrong results. As a crucial consideration, notice that in the scenario considered here fitness evaluation is never uncheatable, that is, there is no trusted helper which could be eventually used to check whether a particular fitness result is correct or not. Therefore, any coping mechanism has to work under the assumption that invocations to the objective function might be always subject to failure.

## 2.1 Modelling Byzantine Faults

As a starting point, we focus a very simple model whereby fitness evaluation requests get an erroneous result with some probability $\rho$ (c.f. [19,20]). We shall denote this wrong result $\hat{f}^t(\cdot)$ as the *unreliable fitness*, where the superscript $t$ represents the current time and is used to denote the fact that cheaters do not necessarily return the same wrong result if the very same solution is submitted for evaluation at different times. Obviously, it is impossible to know beforehand whether the value obtained after an evaluation request is its actual true fitness or an incorrect value, as mentioned before.

In order to quantitatively represent the cheating behavior we are going to consider two simple models of malicious computation:

(i) randomizer, whereby cheaters return a value which is uncorrelated with the true fitness, e.g., a random value within the range of the function (this behavior would correspond to nodes which want to merely feign an activity); in this work, we have considered cheaters that return a previously observed fitness value (randomly selected, and thus lacking any logical relation with the solution submitted for evaluation).

(ii) inverter, whereby cheaters return a value which is inversely correlated with the true fitness (as it would happen if there were computational actors which wanted to inflict damage on the optimization process). In this work, we have considered the following inverter function:

$$\hat{f}^t(x) = f^t_{\max} - (f(x) - f^t_{\min}), \tag{1}$$

where $f^t_{\max}$ and $f^t_{\min}$ are respectively the maximum and minimum fitness observed so far.

## 2.2 Strategies for Handling Byzantine Faults

In order to tackle this kind of faults in an optimization setting, some inspiration can be drawn from noisy environments [9,17]. However, it must be noted that the unreliable fitness does not gravitate in this case around some underlying true fitness (as it is commonly assumed in many scenarios with uncertainty or noisy fitness functions). For this reason, the phenomenon tackled here has a fundamentally different nature. To scrutinize this issue, we are going to consider mechanisms to handle unreliable fitness based on redundant computation: firstly, each new solution will be re-evaluated $k$ times for some $k > 1$; Let $\mathcal{F}^t(x) = [\hat{f}^t_1(x), \ldots, \hat{f}^t_k(x)]$ be the sequence of so obtained unreliable fitness values. Subsequently, we will try to extract an approximation $\tilde{f}^t(x)$ of the true fitness (ideally, an perfect estimation thereof) from these $k$ fitness values by using some appropriate function. To this end, we have considered two possibilities here:

(i) average$k$, namely the average of the $k$ fitness values, i.e.,

$$\tilde{f}^t(x) = \frac{1}{k}\sum_{i=1}^{k} \hat{f}^t_i(x). \tag{2}$$

**Table 1.** Results (mean deviation from the optimum) of the EA with no unreliability handler. In all tables, symbols next to numerical values indicate statistical significance – check the main text of Sect. 3 (p. 5) for details.

| | inverter | | | |
|---|---|---|---|---|
| $\rho$ | OneMax | Trap | MMDP | Leading-Ones |
| 0.00 | $0.00 \pm 0.00\ \star$ | $5.02 \pm 0.52\ \star$ | $0.00 \pm 0.00\ \star$ | $0.00 \pm 0.00\ \star$ |
| 0.05 | $0.00 \pm 0.00\ \star$ | $5.24 \pm 0.45\ \star$ | $0.00 \pm 0.00\ \star$ | $2.80 \pm 0.69\bullet\bullet$ |
| 0.10 | $0.00 \pm 0.00\ \star$ | $5.04 \pm 0.41\ \star$ | $0.00 \pm 0.00\ \star$ | $10.94 \pm 1.00\bullet\bullet$ |
| 0.15 | $0.00 \pm 0.00\ \star$ | $3.76 \pm 0.36\circ\star$ | $0.00 \pm 0.00\ \star$ | $18.88 \pm 0.80\bullet\bullet$ |
| 0.20 | $0.00 \pm 0.00\ \star$ | $4.64 \pm 0.39\ \star$ | $0.00 \pm 0.00\ \star$ | $26.32 \pm 0.90\bullet\bullet$ |
| 0.25 | $0.00 \pm 0.00\ \star$ | $6.02 \pm 0.35\bullet\star$ | $0.00 \pm 0.00\ \star$ | $36.94 \pm 0.58\bullet\bullet$ |
| 0.30 | $0.00 \pm 0.00\ \star$ | $6.98 \pm 0.33\bullet\star$ | $0.71 \pm 0.21\bullet\bullet$ | $45.12 \pm 0.61\bullet\bullet$ |
| 0.35 | $0.00 \pm 0.00\ \star$ | $8.48 \pm 0.28\bullet\star$ | $6.40 \pm 0.73\bullet\bullet$ | $53.02 \pm 0.71\bullet\bullet$ |
| 0.40 | $0.00 \pm 0.00\ \star$ | $10.46 \pm 0.27\bullet\star$ | $13.86 \pm 1.01\bullet\bullet$ | $64.74 \pm 0.42\bullet\bullet$ |
| 0.45 | $1.22 \pm 0.85$ | $10.88 \pm 0.26\bullet\star$ | $17.01 \pm 1.08\bullet$ | $75.64 \pm 0.40\bullet\bullet$ |
| 0.50 | $16.80 \pm 2.41\bullet$ | $19.60 \pm 1.94\bullet\star$ | $20.64 \pm 1.27\bullet\star$ | $83.08 \pm 0.39\bullet\star$ |
| | randomizer | | | |
| $\rho$ | OneMax | Trap | MMDP | Leading-Ones |
| 0.00 | $0.00 \pm 0.00\ \star$ | $5.02 \pm 0.52\ \star$ | $0.00 \pm 0.00\ \star$ | $0.00 \pm 0.00\ \star$ |
| 0.05 | $0.00 \pm 0.00\ \star$ | $5.36 \pm 0.60\ \star$ | $0.00 \pm 0.00\ \star$ | $0.60 \pm 0.34\bullet\bullet$ |
| 0.10 | $0.00 \pm 0.00\ \star$ | $4.78 \pm 0.53\ \star$ | $0.00 \pm 0.00\ \star$ | $1.42 \pm 0.48\bullet\bullet$ |
| 0.15 | $0.00 \pm 0.00\ \star$ | $4.98 \pm 0.51\ \star$ | $0.00 \pm 0.00\ \star$ | $1.70 \pm 0.57\bullet\bullet$ |
| 0.20 | $0.00 \pm 0.00\ \star$ | $3.58 \pm 0.34\bullet\star$ | $0.00 \pm 0.00\ \star$ | $2.60 \pm 0.64\bullet\bullet$ |
| 0.25 | $0.00 \pm 0.00\ \star$ | $2.88 \pm 0.32\bullet\star$ | $0.00 \pm 0.00\ \star$ | $8.18 \pm 1.19\bullet\bullet$ |
| 0.30 | $0.00 \pm 0.00\ \star$ | $1.54 \pm 0.18\bullet\star$ | $0.00 \pm 0.00\ \star$ | $13.18 \pm 1.34\bullet\bullet$ |
| 0.35 | $0.00 \pm 0.00\ \star$ | $1.22 \pm 0.19\bullet\star$ | $0.13 \pm 0.07\circ\circ$ | $15.88 \pm 1.16\bullet\bullet$ |
| 0.40 | $0.00 \pm 0.00\ \star$ | $1.38 \pm 0.12\bullet\star$ | $1.30 \pm 0.17\bullet\bullet$ | $19.64 \pm 1.06\bullet\bullet$ |
| 0.45 | $0.00 \pm 0.00\ \star$ | $2.14 \pm 0.18\bullet\star$ | $2.62 \pm 0.28\bullet\bullet$ | $22.76 \pm 1.52\bullet\bullet$ |
| 0.50 | $0.00 \pm 0.00\ \star$ | $3.26 \pm 0.17\bullet\star$ | $4.31 \pm 0.30\bullet\bullet$ | $28.94 \pm 1.33\bullet\bullet$ |

(ii) majority$k$ (cf. [4]), namely keeping the most repeated value (or an average of the most repeated values if there was a tie). More precisely, let $\sigma : \mathbb{R} \times \mathbb{R}^k \to \{0, \dots k\}$ be defined such that $\sigma(f, \mathcal{F})$ is the number of occurrences of value $f$ in vector $\mathcal{F}$, and let $\sigma^*(x,t) = \max\{\sigma(f, \mathcal{F}^t(x)) \mid f \in \mathcal{F}^t(x)\}$. Now, we consider

$$\tilde{f}^t(x) = \frac{1}{|\mathcal{F}'|} \sum_{f \in F'} f \qquad (3)$$

where $\mathcal{F}' = \{f \in \mathcal{F}^t(x) \mid \sigma(f, \mathcal{F}^t(x)) = \sigma^*(x,t)\}$.

## 3 Experimental Results

The experiments have been conducted with an EA that uses binary tournament selection, single-point crossover ($p_X = .9$), bit-flip mutation ($p_M$ equivalent to a

**Table 2.** Results (mean deviation from the optimum) of the EA with majority3.

| | inverter | | | |
|---|---|---|---|---|
| $\rho$ | OneMax | Trap | MMDP | Leading-Ones |
| 0.00 | $0.00 \pm 0.00\ \star$ | $9.46 \pm 0.34\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $0.00 \pm 0.00\ \star$ |
| 0.05 | $0.00 \pm 0.00\ \star$ | $9.62 \pm 0.47\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $0.32 \pm 0.21\circ\circ$ |
| 0.10 | $0.00 \pm 0.00\ \star$ | $9.66 \pm 0.50\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $2.26 \pm 0.62\bullet\bullet$ |
| 0.15 | $0.00 \pm 0.00\ \star$ | $9.48 \pm 0.39\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $6.22 \pm 0.91\bullet\bullet$ |
| 0.20 | $0.00 \pm 0.00\ \star$ | $10.08 \pm 0.46\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $12.98 \pm 1.04\bullet\star$ |
| 0.25 | $0.00 \pm 0.00\ \star$ | $9.90 \pm 0.41\bullet\bullet$ | $0.08 \pm 0.06$ | $23.12 \pm 0.92\bullet$ |
| 0.30 | $0.00 \pm 0.00\ \star$ | $9.96 \pm 0.42\bullet\bullet$ | $0.17 \pm 0.08\bullet\star$ | $33.28 \pm 0.86\bullet\star$ |
| 0.35 | $0.00 \pm 0.00\ \star$ | $10.40 \pm 0.35\bullet\bullet$ | $0.89 \pm 0.17\bullet\star$ | $44.20 \pm 0.61\bullet\star$ |
| 0.40 | $0.00 \pm 0.00\ \star$ | $12.44 \pm 0.33\bullet\bullet$ | $5.34 \pm 0.58\bullet\star$ | $57.44 \pm 0.63\bullet$ |
| 0.45 | $0.00 \pm 0.00\ \star$ | $12.96 \pm 0.35\bullet\bullet$ | $14.77 \pm 0.70\bullet\star$ | $72.78 \pm 0.48\bullet\star$ |
| 0.50 | $16.52 \pm 2.37\bullet\star$ | $22.30 \pm 1.81\bullet\bullet$ | $22.67 \pm 1.05\bullet\circ$ | $84.76 \pm 0.35\bullet\bullet$ |

| | randomizer | | | |
|---|---|---|---|---|
| $\rho$ | OneMax | Trap | MMDP | Leading-Ones |
| 0.00 | $0.00 \pm 0.00\ \star$ | $9.46 \pm 0.34\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $0.00 \pm 0.00\ \star$ |
| 0.05 | $0.00 \pm 0.00\ \star$ | $10.16 \pm 0.45\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $0.00 \pm 0.00\ \star$ |
| 0.10 | $0.00 \pm 0.00\ \star$ | $8.82 \pm 0.41\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $0.20 \pm 0.20$ |
| 0.15 | $0.00 \pm 0.00\ \star$ | $9.60 \pm 0.49\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $0.12 \pm 0.12$ |
| 0.20 | $0.00 \pm 0.00\ \star$ | $9.34 \pm 0.48\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $0.60 \pm 0.44$ |
| 0.25 | $0.00 \pm 0.00\ \star$ | $9.04 \pm 0.39\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $0.50 \pm 0.25\bullet\bullet$ |
| 0.30 | $0.00 \pm 0.00\ \star$ | $9.82 \pm 0.39\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $1.52 \pm 0.56\bullet\bullet$ |
| 0.35 | $0.00 \pm 0.00\ \star$ | $9.46 \pm 0.52\bullet\bullet$ | $0.00 \pm 0.00\ \star$ | $2.12 \pm 0.70\bullet\bullet$ |
| 0.40 | $0.00 \pm 0.00\ \star$ | $8.86 \pm 0.42\bullet\bullet$ | $0.04 \pm 0.04$ | $6.20 \pm 1.25\bullet\bullet$ |
| 0.45 | $0.00 \pm 0.00\ \star$ | $8.84 \pm 0.39\bullet\bullet$ | $0.13 \pm 0.07\circ\star$ | $9.22 \pm 1.22\bullet\bullet$ |
| 0.50 | $0.00 \pm 0.00\ \star$ | $9.92 \pm 0.34\bullet\bullet$ | $1.61 \pm 0.22\bullet\bullet$ | $17.20 \pm 1.53\bullet\bullet$ |

mutation rate $1/\ell$ per bit, where $\ell$ is the number of bits), and elitist generational replacement. The population size is $\mu = 100$ individuals, and the total number of fitness evaluations is $10^6$ (including redundant computations). The unreliability rate $\rho$ ranges from 0 to 0.5 in steps of 0.05 (the results for $\rho = 0$ can be used for gauging the basal performance of the EA). We consider a *raw* EA that uses no unreliability handling mechanism in addition to the majority$k$ and average$k$ handlers. For the latter two, the value $k = 3$ is considered. Four objective functions are used in the experiments, namely OneMax (using $\ell = 100$ bits), Deb's 4-bit fully deceptive function [6] (Trap, using 25 blocks of 4 bits), Goldberg et al.'s Massively Multimodal Deceptive Problem [8] (MMDP, using 17 blocks of 6 bits), and Rudolph's Leading-Ones [18] (using $\ell = 100$ bits). We perform 50 runs for each handler and problem.

Table 1–3 show the numerical results attained. Each entry in these tables indicates the mean relative error (percentage distance from the optimum) and the standard error of the mean, measured from the best true fitness of any solution generated during each run. In addition, two symbols indicate the statistical significance (according to a Wilcoxon test) of the difference with respect to the

**Table 3.** Results (mean deviation from the optimum) of the EA with average3.

| | inverter | | | |
|---|---|---|---|---|
| $\rho$ | OneMax | Trap | MMDP | Leading-Ones |
| 0.00 | 0.00 ± 0.00 ⋆ | 9.46 ± 0.34●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.05 | 0.00 ± 0.00 ⋆ | 9.30 ± 0.37●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.10 | 0.00 ± 0.00 ⋆ | 9.02 ± 0.41●● | 0.04 ± 0.04 | 0.14 ± 0.11 ⋆ |
| 0.15 | 0.00 ± 0.00 ⋆ | 9.04 ± 0.33●● | 0.13 ± 0.07○○ | 3.00 ± 0.77●⋆ |
| 0.20 | 0.00 ± 0.00 ⋆ | 10.50 ± 0.32●● | 0.41 ± 0.14●● | 15.16 ± 1.50● |
| 0.25 | 0.00 ± 0.00 ⋆ | 10.78 ± 0.35●● | 1.23 ± 0.18●● | 22.82 ± 1.08●⋆ |
| 0.30 | 0.00 ± 0.00 ⋆ | 11.56 ± 0.35●● | 2.70 ± 0.25●● | 36.00 ± 0.94●● |
| 0.35 | 0.00 ± 0.00 ⋆ | 12.24 ± 0.33●● | 4.37 ± 0.31●● | 47.62 ± 0.68●● |
| 0.40 | 0.00 ± 0.00 ⋆ | 12.92 ± 0.34●● | 9.56 ± 0.37●● | 56.88 ± 0.57●⋆ |
| 0.45 | 0.00 ± 0.00 ⋆ | 13.62 ± 0.32●● | 17.17 ± 0.69●● | 73.30 ± 0.40● |
| 0.50 | 19.10 ± 2.33●● | 28.26 ± 1.95●● | 26.17 ± 0.80●● | 85.26 ± 0.30●● |
| | randomizer | | | |
| $\rho$ | OneMax | Trap | MMDP | Leading-Ones |
| 0.00 | 0.00 ± 0.00 ⋆ | 9.46 ± 0.34●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.05 | 0.00 ± 0.00 ⋆ | 9.52 ± 0.51●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.10 | 0.00 ± 0.00 ⋆ | 9.50 ± 0.47●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.15 | 0.00 ± 0.00 ⋆ | 9.46 ± 0.51●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.20 | 0.00 ± 0.00 ⋆ | 8.74 ± 0.42●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.25 | 0.00 ± 0.00 ⋆ | 9.54 ± 0.42●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.30 | 0.00 ± 0.00 ⋆ | 10.02 ± 0.37●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.35 | 0.00 ± 0.00 ⋆ | 10.06 ± 0.36●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.40 | 0.00 ± 0.00 ⋆ | 10.48 ± 0.41●● | 0.00 ± 0.00 ⋆ | 0.00 ± 0.00 ⋆ |
| 0.45 | 0.00 ± 0.00 ⋆ | 10.62 ± 0.38●● | 0.17 ± 0.08● | 0.00 ± 0.00 ⋆ |
| 0.50 | 0.00 ± 0.00 ⋆ | 11.12 ± 0.35●● | 0.21 ± 0.09●⋆ | 0.02 ± 0.02 ⋆ |

base case (no handler, $\rho = 0$) and to the best result for all three scenarios (using average$k$, majority$k$, or no handler) and the same $\rho$ and cheating model (entry marked with $\star$) respectively. In either case, $\bullet$ and $\circ$ indicate significance at $\alpha = .05$ and $\alpha = 0.1$ respectively, and absence of a symbol indicates no statistical significance of the corresponding difference. Analyzing firstly the results corresponding to the raw EA (Table 1) it is evident that in general the unreliability of the environment has a clear toll on the performance of the EA, whose results markedly degrade for increasing values of $\rho$. There is an interesting anomaly for the Trap function, in which moderate values of $\rho$ provide a subtle improvement in the results. Clearly, the slightly misleading fitness information seems to be carrying the EA out of some of the deceptive local optima. On a more general note, it is clear that the randomizer model provides a milder disturbance than the inverter model across all problems. This is consistent with the lower population diversity attained during the run (compare the curves labelled as none in Fig. 1 and Fig. 2 for each problem, which indicate a more more focused search for the latter model).

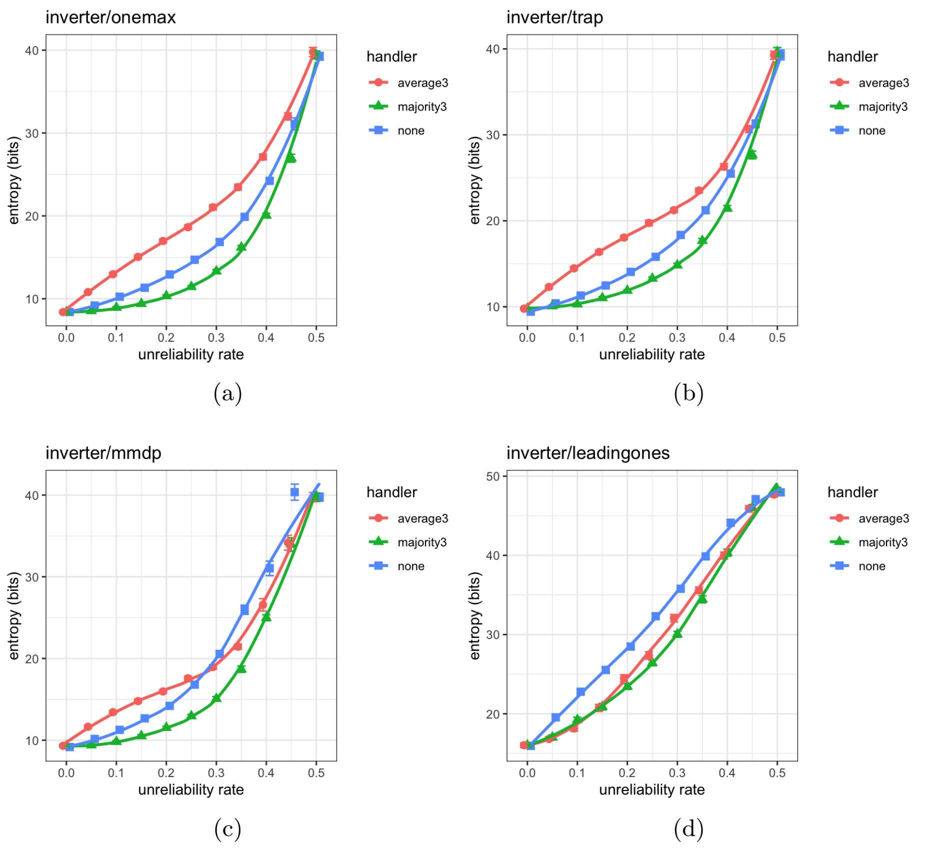
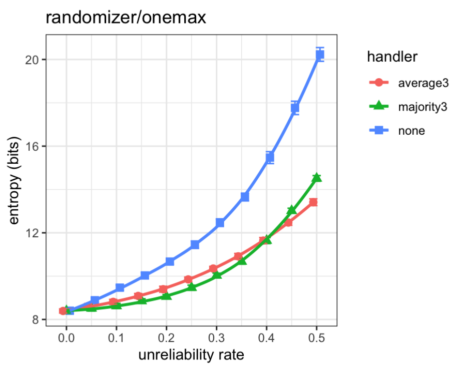
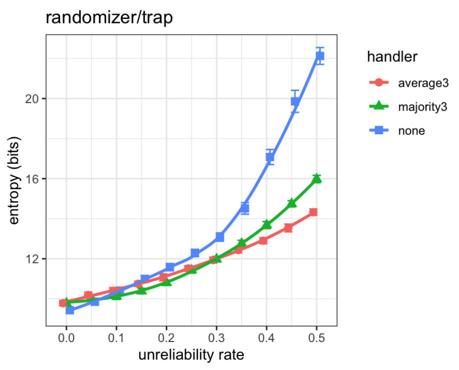


**Fig. 1.** Minimum population entropy as a function of the unreliability rate for the inverter model. The data points are the average of 50 runs, the error bars span the standard error of the mean, and the solid lines are visual guides. (a) OneMax (b) Trap (c) MMDP (d) Leading-Ones.

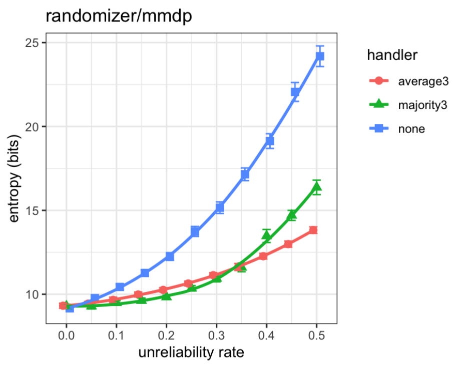
Focusing on this inverter model, observe now the results of majority3 and average3 (Tables 2 and 3 respectively). The majority3 handler provides a clear improvement for this harder unreliability model. The OneMax problem is quite simple and therefore the EA seems to be less sensitive to $\rho$, but still this handler matches the best results and improves these for the largest value of $\rho$. The result for the Trap function are slightly worse due to the previously mentioned anomaly, which the majority3 actually ameliorates. This can be also seen in Fig. 1, in which the lowest diversity of the population is depicted for the inverter model. Note how the majority3 handler manages to reach a lower entropy, which indicates greater convergence of the population. Unfortunately, for the Trap function this implies falling into suboptimal regions. Note also in Table 3 that for low values of $\rho$ the average3 handler is actually more competitive in the Leading-Ones



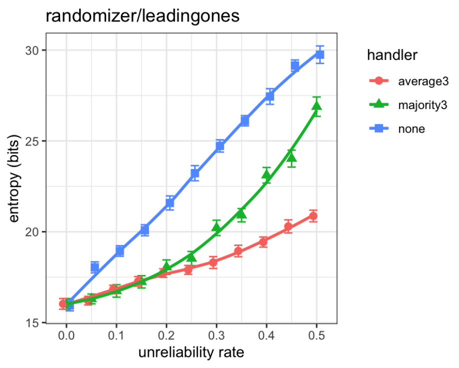
(a)



(b)



(c)



(d)

**Fig. 2.** Minimum population entropy as a function of the unreliability rate for the randomizer model. The data points are the average of 50 runs, the error bars span the standard error of the mean, and the solid lines are visual guides. (a) OneMax (b) Trap (c) MMDP (d) Leading-Ones.

function. The reason for this may be found in the nature of this particular objective function: the worst values (typically those of random solutions in the initial population) are close to 0, and therefore tend to be out-weighted when averaged with high true fitness values, therefore providing valuable information to be exploited by the EA.

Finally, the randomizer model seems to provide a less challenging scenario, and this is where the average3 handler seems to excel. The results for the OneMax and the Leading-Ones functions are markedly superior, and this can be basically attributed to the same reason mentioned before with regard to the structure of fitness values and the possibility for high true fitness values to drag the average upwards, supplying guidance to the EA. This is also true for the MMDP function in which the signal-to-noise ratio is lower but the diversity balance seems to fit well to the EA. Indeed, it can be observed in Fig. 2 that the search is much more focused under this failure model, as reflected by the smaller population entropy attained by the EA.

## 4 Conclusions

The presence of cheaters can have a noticeable impact in the performance of panmictic EAs, even for simple objective functions. As the unreliability of the environment increases, the performance of the EA (as measured by the average fitness attained after a fixed number of invocations to the fitness function) drops in general. We have considered two mechanisms to cope with this issue on the basis of performing redundant computations. A handler based on averaging (reminiscent of the approaches commonly used in noisy environments) can work well in a relatively benign scenario in which cheaters provide random yet unbiased values. We hypothesize the reason lies in the possibility of marginally exploiting some fitness gradient information via the aggregate average. However, dealing with malevolent agents that actively try to provide fitness information manipulated to direct the algorithm in the opposite direction is better dealt with using a handler based on majority voting (defaulting to the average if no majority is obtained), which imposes a more strict criterion for assigning fitness, and relies of the observation that for unreliability rates less than 0.5 (and therefore non-pathological) the chances of obtaining a majority vote for the real fitness is higher than the opposite. Further research is required to optimize the computational tradeoffs involved in these mechanisms. We are currently working along this line, as well as towards the development of more sophisticated handling mechanisms and their deployment in more complex scenarios.

## Acknowledgments

The author thanks Daan van den Berg (VU Amsterdam) for interesting discussions arising from a previous work [3].

## References

1. Camacho, D., et al.: From ephemeral computing to deep bioinspired algorithms: New trends and applications. Future Generation Computer Systems **88**, 735–746 (2018)
2. Cantú-Paz, E.: Master-slave parallel genetic algorithms. In: Efficient and Accurate Parallel Genetic Algorithms, pp. 33–48. Springer US, Boston, MA (2001)
3. Cotta, C.: On the performance of evolutionary algorithms with unreliable fitness information. In: Mora, A.M. (ed.) EvoStar 2023 Late Breaking Abstracts. Brno, Czech Republic (2023)
4. Cotta, C.: Tackling adversarial faults in panmictic evolutionary algorithms. In: Genetic and Evolutionary Computation Conference Companion (GECCO'23 Companion). ACM Press, New York, NY (2023), 2 pages, In press

5. Cotta, C., Olague, G.: Resilient bioinspired algorithms: A computer system design perspective. In: Jiménez Laredo, J.L., Hidalgo, J.I., Babaagba, K.O. (eds.) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 13224, pp. 619–631. Springer, Cham (2022)
6. 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, Colorado, USA (1993)
7. Fernández de Vega, F.: Evolutionary algorithms: Perspectives on the evolution of parallel models. In: Novais, P., Camacho, D., Analide, C., El Fallah Seghrouchni, A., Badica, C. (eds.) Intelligent Distributed Computing IX. pp. 13–22. Springer International Publishing, Cham (2016)
8. 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, NY, USA (1992)
9. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments – A survey. IEEE Transactions on Evolutionary Computation **9**(3), 303–317 (2005)
10. Lavoie, E., Hendren, L.: Personal volunteer computing. In: Proceedings of the 16th ACM International Conference on Computing Frontiers. pp. 240–246. ACM, New York NY (2019)
11. Mengistu, T.M., Che, D.: Survey and taxonomy of volunteer computing. ACM Computing Surveys **52**(3) (2019)
12. Muszynski, J.: Cheating-Tolerance of Parallel and Distributed Evolutionary Algorithms in Desktop Grids and Volunteer Computing Systems. Ph.D. thesis, University of Luxembourg (2015)
13. Muszyński, J., Varrette, S., Bouvry, P., Seredyński, F., Khan, S.U.: Convergence analysis of evolutionary algorithms in the presence of crash-faults and cheaters. Computers & Mathematics with Applications **64**(12), 3805–3819 (2012)
14. Muszyński, J., Varrette, S., Dorronsoro, B., Bouvry, P.: Distributed cellular evolutionary algorithms in a byzantine environment. In: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop. pp. 307–313. IEEE Press, Hyderabad, India (2015)
15. Nogueras, R., Cotta, C.: Self-healing strategies for memetic algorithms in unstable and ephemeral computational environments. Natural Computing **16**(2), 189–200 (2017)
16. Nogueras, R., Cotta, C.: Analyzing self-$\star$ island-based memetic algorithms in heterogeneous unstable environments. The International Journal of High Performance Computing Applications **32**(5), 676–692 (2018)
17. Rakshit, P., Konar, A., Das, S.: Noisy evolutionary optimization algorithms – A comprehensive survey. Swarm and Evolutionary Computation **33**, 18–45 (2017)
18. Rudolph, G.: Convergence properties of evolutionary algorithms. Verlag Dr. Kovač (1997)
19. Sarmenta, L.F.: Sabotage-tolerance mechanisms for volunteer computing systems. Future Generation Computer Systems **18**(4), 561–572 (2002)
20. Varrette, S., Tantar, E., Bouvry, P.: On the resilience of [distributed] EAs against cheaters in global computing platforms. In: 25th IEEE International Symposium on Parallel and Distributed Processing Workshop Proceedings. pp. 409–417. IEEE, Anchorage AK (2011)
21. Xiong, N., Molina, D., Ortiz, M.L., Herrera, F.: A walk into metaheuristics for engineering optimization: Principles, methods and recent trends. International Journal of Computational Intelligence Systems **8**, 606–636 (2015)