On the Performance of Evolutionary Algorithms with Unreliable Fitness Information[⋆]

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 consider the use of evolutionary algorithms (EAs) in byzantine environments in which fitness information can be computed by malicious agents. The performance of panmictic EAs is analyzed in this context, measuring the influence of the rate of unreliability of the environment. It is shown that even for simple problems there is noticeable performance degradation, highlighting the need for appropriate mechanisms to cope with this issue.

Keywords: Evolutionary algorithms · Byzantine faults · Panmixia · Resilience

1 Introduction

Bioinspired optimization is a computationally intensive activity that is very much in need of abundant computational resources. In this sense, unconventional computational environments such as cloud, volunteer-computing and P2P networks have received great interest in recent years as adequate platforms to successfully solve complex problems [3]. The irregular and dynamic computational landscape that these platforms provide can pose a challenge to bioinspired algorithms though, thus underlining the need for algorithmic resilience [2]. Fortunately, bioinspired techniques in general and evolutionary algorithms (EAs) in particular are inherently quite resilient and are also flexible enough to being adapted for working in environments plagued with volatility or heterogeneity [6].

In this work we turn our attention to disruptions caused by malicious activities [8]. This kind of phenomena fall under the umbrella of byzantine failures, and can be described as cheating faults, whereby a contributor of computational resources purposefully alters the outcome of the computation by submitting erroneous results (aimed to feigning an activity or even to willingly damage the computation). These faults can have an impact on the algorithm depending on the components of the algorithm that are affected [4]. Empirically, it has

⋆ 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

been shown that EAs (particularly cellular EAs) can withstand certain types of byzantine failures [5]. We are here interested in analyzing both qualitatively and quantitatively what the effect of other types of byzantine failures can be. To this end, we initially focus on panmictic EAs.

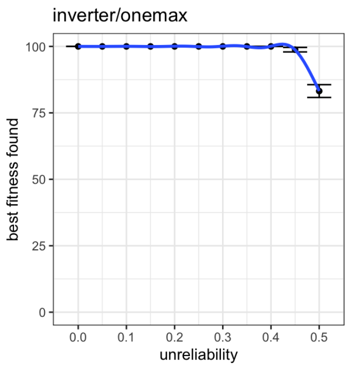
2 Algorithmic Setting

We consider an elitist generational EA with a panmictic population. This algorithm is used to optimize a certain objective function $f(\cdot)$. We will denote the values returned by this function as the *true fitness*. Let us further assume that a master-slave model [1] is used to compute fitness. Thus, we have a (possibly dynamic) network of computational nodes providing this fitness evaluation service. Some of these nodes are *cheaters* though. Similarly to [5], we consider a very simple model in which a fitness evaluation request returns an erroneous result with some probability p (which will be later a control parameter in the experimentation). We will denote this result $\hat{f}^t(\cdot)$ as the *unreliable fitness*, where the superscript t is used to indicate the time t at which this value is returned (unreliable nodes are not assumed to consistently return the same erroneous result should the same solution been submitted for evaluation multiple times). Of course, there is no way of knowing a priori whether the value obtained when evaluating an individual is its true fitness or is a misleading value. Now, we consider two simple models of malicious behavior with regard to unreliable fitness:

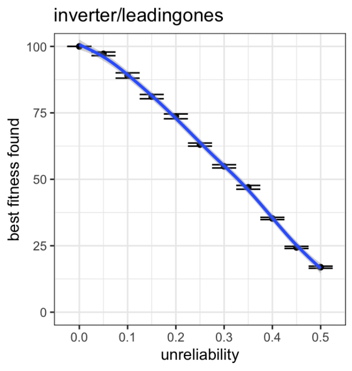
– randomizer: the malicious computational node will return a value which is uncorrelated with the true fitness. This can be a random value within the range of the function, or -in our implementation- the true fitness value of another solution evaluated previously. This behavior would be related to nodes which want to feign an activity, without doing the actual work.
– inverter: in this case, the value returned is negatively correlated with the true fitness. In our experiments, we keep track of the maximum fitness $f^t_{\max}$ and minimum fitness $f^t_{\min}$ computed so far, and return the reflection of the true fitness within this interval, i.e., $\hat{f}^t(x) = f^t_{\max} - (f(x) - f^t_{\min})$. This behavior would be related to nodes which want to actually inflict damage in the computation by leading the algorithm in the wrong direction.

3 Experimentation

We have conducted experiments with four objective functions, although due to space limitations we focus here on two of them, namely OneMax and Leading-Ones. In both cases, we consider 100-bit solutions. As to the EA, it has a population of $\mu = 100$ individuals, uses binary tournament selection, single-point crossover ($p_X = .9$), bit-flip mutation (p_M equivalent to a mutation rate $1/\ell$ per bit, where ℓ is the genome length), and elitist generational replacement. The total number of fitness evaluations is 10^6. As to the unreliability rate p, since values $p > .5$ arguably correspond to pathological situations, we focus on



(a)



(b)

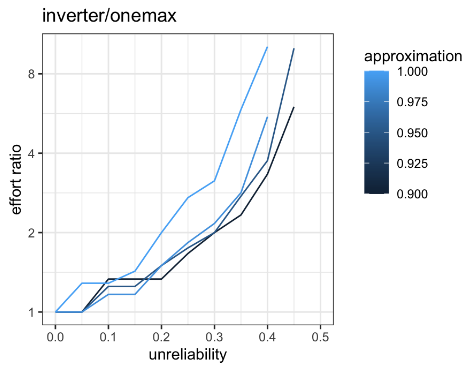
Fig. 1: Best true fitness of all solutions generated by the algorithm. (a) OneMax (b) Leading-Ones.

values between 0 and 0.5 in steps of 0.05, and use the results for $p = 0$ (reliable scenario) to define the base-performance of the algorithm. We will focus on the inverter model. We perform 50 runs for each parameter setting and problem.

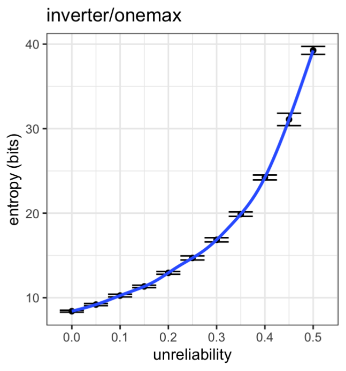
Figs. 1(a)–(b) show the best true fitness found as a function of the unreliability probability p. Similarly to what was found in [5] for a scenario analogous to randomizer, OneMax is barely affected by the unreliability factor p except in extreme cases. However, and quite distinctly, Leading-Ones is much more sensitive and not only fails to hit the optimum consistently for $p > 0$ but significantly degrades as p increases too. It must be noted that even though OneMax seems less affected (ultimately due to its simplicity), the search dynamics of the EA suffers greatly in the presence of unreliable fitness. If we define the relative effort of the algorithm for a certain unreliability rate p as the ratio between the mean number of evaluations required to find a solution whose true fitness is within some percentage of the optimum for that value of p and for $p = 0$, we can see in Fig. 2(a) how this effort noticeably grows up to nearly an order of magnitude for the larger values of p, and even precludes reaching fitness values close to the optimal value. The presence of unreliable fitness information makes the EA lose focus, as shown in Fig. 2(b) with a much larger population variance due to the presence of *impostors* who survive by virtue of malicious fitness assignation.

4 Conclusions

We have here studied the impact that the presence of unreliable (and even adversarial) fitness information has in the performance of panmictic EAs. It has been shown that even for simple objective functions unreliability can pose serious problems in terms of convergence and the effort required to attain good quality solutions. We are currently working on mechanisms to cope with this



(a)



(b)

Fig. 2: (a) Relative effort of the algorithm to find a certain approximation of the optimum (lines are discontinued if the approximation cannot be found for a certain unreliability rate). Note the logarithmic y-axis. (b) Minimum population entropy during the run. In both cases, the results are for OneMax.

issue. Ideas from noisy environments [7] can be serve as inspiration, although the nature of the phenomenon considered here is fundamentally different since uncertain fitness information does not gravitate around true fitness in this case.

References

1. Cantú-Paz, E.: Master-slave parallel genetic algorithms. In: Efficient and Accurate Parallel Genetic Algorithms, pp. 33–48. Springer US, Boston, MA (2001)
2. Cotta, C., Olague, G.: Resilient bioinspired algorithms: A computer system design perspective. In: Jiménez Laredo, J.L., et al. (eds.) Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 13224, pp. 619–631. Springer, Cham (2022)
3. Mengistu, T.M., Che, D.: Survey and taxonomy of volunteer computing. ACM Computing Surveys **52**(3) (2019)
4. 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)
5. 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 (2015)
6. 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)
7. Rakshit, P., Konar, A., Das, S.: Noisy evolutionary optimization algorithms – A comprehensive survey. Swarm and Evolutionary Computation **33**, 18–45 (2017)
8. Sarmenta, L.F.: Sabotage-tolerance mechanisms for volunteer computing systems. Future Generation Computer Systems **18**(4), 561–572 (2002)