Evaluating the Impact of Hysteretic Phenomena and Implementation Choices on Energy Consumption in Evolutionary Algorithms*

Carlos Cotta[1,2] [0000−0001−8478−7549] and Jesús Martínez-Cruz[1] [0000−0002−8847−8900]

[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 jmcruz@uma.es

Abstract. As the demand for environmentally sustainable computing grows, understanding the energy consumption of AI systems has become increasingly important. This paper explores how hysteretic phenomena and implementation choices affect the energy consumption of evolutionary algorithms (EAs). Specifically, we consider the case of running EAs in batch and show how back-to-back executions can put a significant strain on the underlying processing device, resulting in increased energy consumption. An experimental analysis indicates that the introduction of short pauses can alleviate this problem and reduce consumption by up to 9% in the considered benchmark. We also conduct a comparative analysis between two twin implementations of the same EA library in Java and C++, revealing that the latter scales better in terms of energy efficiency and running time, thus underpinning the importance of implementation decisions and best practices when aiming to optimize an algorithm's energy consumption.

Keywords: Evolutionary Algorithms, Energy Consumption, Green AI, Sustainable Computing

1 Introduction

Sustainability denotes the capacity to maintain a trend or process over an extended period of time. It is a concept that is frequently brought up in connection with human activities, the impact they have on the environment, and the toll they exert on resource availability. These factors obviously jeopardize the regular operation of such activities and even introduce existential risks in our society as a whole. Optimizing processes with sustainability in mind is obviously desirable and is a goal to which artificial intelligence (AI) methods can and should contribute (e.g., [3,14,19,22]). However, it is also important to realize that applying

* 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.

these methods and even conducting research on them is in itself a human activity that can be monitored and critically analyzed following the very same sustainability principles as any other industrial, technological, or social process. In fact, AI methods have been identified to have a significant carbon footprint [4,12]. Most notably, the energy consumption of AI systems has skyrocketed in recent years (since 2021, the energy consumption of these techniques has experienced a 300,000-fold rise, so they may be responsible for up to 5-9% of the world's electricity demand and up to 2% of all CO_2 emissions [10]). Therefore, prioritizing energetically efficient computational platforms and algorithms is crucial.

The importance of energy efficiency is becoming an increasingly recognized factor by the community when developing AI techniques, although for the most part the focus remains on maximizing performance rather than energy efficiency. Although these are not completely opposed goals, the latter and most generally the different trade-offs attainable between both objectives are often overlooked. In particular, in connection with such trade-offs, the notions of *red AI* and *green AI* have been coined [18]. The former refers to AI research that seeks to push the state of the art through massive computational power, and particularly applies to scenarios in which the computational effort scales at a substantially faster pace than the gains obtained in the results [9]. Needless to say, this kind of research can still be valuable in the long run if the benefits of the results are socially desirable even if the short-term computational cost is disparate. Still, there is a trend to have it superseded by green AI research, that results in novel results without increasing computational cost or even reducing it. This involves a methodological shift in how research is conducted, or at the very least the adoption of a new set of best practices.

Although there have been a number of initiatives in this direction within the broad field of AI, in most cases such proposals come from the machine learning community (e.g., [9,20]), partially because of the specificities of algorithms in that area. The situation is different in the area of evolutionary computing, where, despite some foundational research [1,5,6] (mainly focused on algorithmic components and parameterization), there is still much unexplored territory. For example, factors involving specific implementation choices or experimental practices are largely unexplored and require deeper analysis due to the influence they can have on the final energy consumption of the algorithm. In this sense, following some preliminary investigation [2], we turn our attention here to practical issues arising when executing EAs in batches (a common practice in order to obtain large enough datasets of results to be deemed statistically significant) and to what impact some programming and test configuration choices can have on the energy consumption of these techniques.

2 Materials and Methods

As mentioned above, we will focus on energy issues when running batches of EA experiments. The general context of our experimentation is described in the following (Sect. 2.1). Subsequently, we will describe the particular EA library

considered in the experimental part and some implementation factors that may impact the energy profile of the algorithms (Sect. 2.2). Finally, we will detail the experimental setup considered (Sect.2.3).

2.1 Computational Context

Assume a given optimization problem —-defined by an objective function f– that we wish to tackle with evolutionary algorithms (EAs). To this end, and given the stochastic nature of EAs, standard practice dictates that this EA will have to be run multiple times, whether it is for scholarly reasons (i.e., to gather a large sample of results which can sustain statistical significance when assessing the performance of the algorithm or its parameterization) or for pure applied purposes (e.g., to obtain solutions of higher quality via multiple independent runs). Each of these collections of replicated runs is termed a *batch*. In this scenario, each run of the EA will not just have an associated energy cost, but it will also impact the system state in various ways, both logically (e.g., the cache memory, register values, or the state of memory paging among other factors will be different at the start of each run) and physically (e.g., thermal and electromagnetic changes in computer components due to their –possibly computationally intensive– operation). These changes can and are likely to influence system performance at later times. Consider, for instance, the CPU temperature: rising temperatures may trigger thermal throttling to prevent overheating [13,21], which could, in turn, slow down processing. In addition, slight changes in the electrical properties of the CPU components can take place [11,23] or increased energy consumption can occur due to factors such as fan operation [24].

Broadly speaking, we can refer to these effects as *hysteretic* phenomena [8], since the state and behavior of the system (the computational environment in this case) depend on its history, in the sense that the system does not immediately return to its initial state after completing an algorithm run. Subsequent runs of the EA may thus have a different energy cost and can contribute to further straining of the state of the system. To investigate this issue, we are going to analyze the behavior of the system when performing experiments in batches that incorporate rest periods of length ρ between runs, looking for any changes within the energy profile of the algorithms. The rationale for this strategy is to alleviate the pressure put on the computing components and allow them to return to a more energy-efficient state before performing the next EA experiment.

2.2 Implementation Factors

As anticipated in Sect. 1, implementation choices can have an impact on the behavior of the algorithm in any single run. Such choices can appear at many levels, ranging from the programming language used to the particular data structures or the low-level procedures used in different parts of the algorithm. Focusing on the choice of programming language, this is certainly an issue that has been recognized by the community; see, e.g., [16,17]. Most studies have focused on the

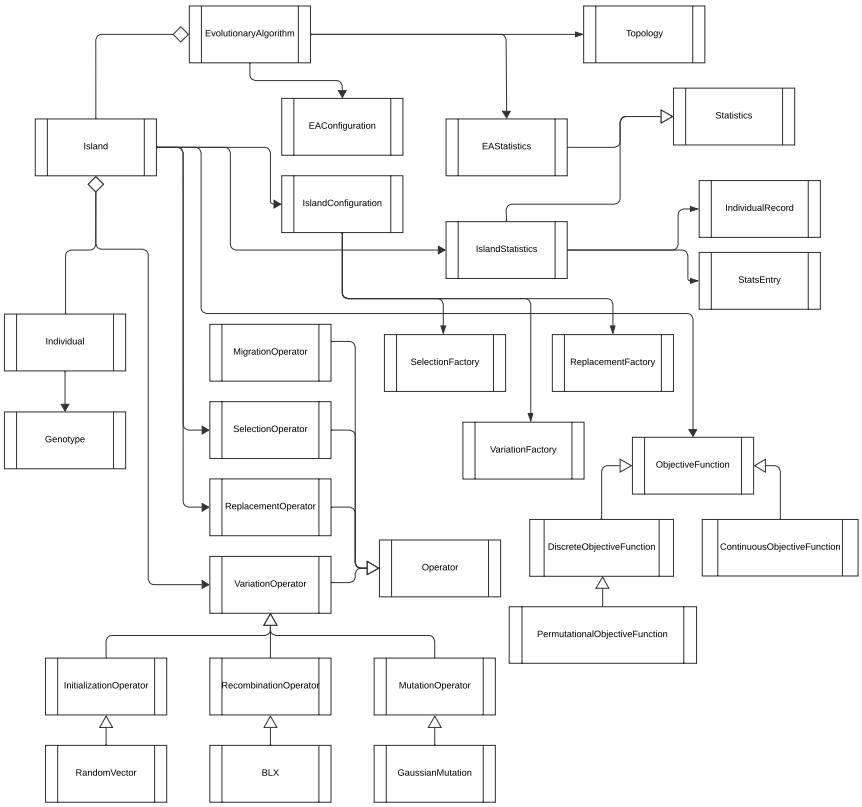


Fig. 1: UML diagram showing the relationship (usage, aggregation, inheritance) among the higher-level classes in the EA library considered. A myriad of other classes derive from these core classes in order to implement specific operators.

impact that this choice has on execution speed. However, we are more interested in the implications for energy consumption. To this end, a fair comparison requires that, at least in principle, the implementations considered are as similar as possible in logical terms and that any differences are due to issues deeply rooted in the underlying programming language (e.g., primitive data types, dynamic memory management, or concurrency model, just to cite some examples) and thus their influence cannot be dissociated from the latter, and are not due to higher-level implementation decisions.

In line with the principles mentioned, we have considered a flexible and expandable EA library, initially implemented in Java[3], and which we have inte-

[3] Available in our GitHub repository https://github.com/Bio4Res/ea.

grally ported to C++[4], adhering to the very same design and structure as the original library. Fig. 1 provides a depiction of this structure and the relationship among classes in the library. These classes implement useful design patterns (such as Strategy, Factory, and Prototype [7]), which enable the creation at runtime of the EA algorithm's specific operations and parameters from a JSON configuration file.

For its translation to C++ (using a modern version based on the C++20 standard) and to ensure a fair comparison, special attention has been paid to replicate exactly the original Java library design, including class hierarchy and design patterns. Only two mandatory changes were implemented during the code translation:

- Given C++'s lack of a garbage collector, smart pointers (`std::unique_ptr` and `std::shared_ptr`) have been employed for automatic heap memory management, deliberately avoiding any use of memory allocators.
- In the Java library, each gene in the genotype is modeled as a generic reference to an `Object` which, in practice and for a specific EA, allows for completely flexible genotype content. In the C++ version, we have opted (temporarily) to model a gene as a *safe* union type (`std::variant`) between `int` or `double` values, which is sufficient to model a large number of cases. This decision takes into account that accessing values stored within a `std::variant` incurs runtime overhead, which is also present in its Java counterpart when recasting the specific gene value from `Object`.

Regarding external dependencies, both Java and C++ implementations make use of a library for JSON management[5], which has no impact during the problem execution. Similarly, for factory methods in C++, another library is used for converting strings to enumerations, a task that is done at compile time, thus not affecting performance.

2.3 Experimental Setup

To perform the experiments, we have considered three multidimensional numerical optimization functions, namely ackley, rastrigin and rosenbrock; see Table 1. Each of these functions has been tested with different dimensions $d \in \{10, 20, 50, 100\}$. In all cases, we have considered a standard generational elitist EA (popsize = 100, binary tournament selection, BLX-α recombination, Gaussian mutation) for $maxevals = 10^7$ function evaluations per run[6]. We have performed batches of $n = 30$ runs of this algorithm on each combination of objective function, dimensionality, and programming language, leaving some rest ρ between runs. More precisely, we have considered $\rho \in \{0, 10, 100\}$ (in seconds).

[4] Available in https://github.com/Bio4Res/ea-cpp

[5] json-simple (https://cliftonlabs.github.io/json-simple/) in the case of Java, and nlohmann.json (https://github.com/nlohmann/json) in the case of C++.

[6] The implementation of the objective functions and the configuration files used in the experimentation are available in https://github.com/Bio4Res/ea-test-energy

Table 1: Functions considered in the experimentation.

function	mathematical formulation
ackley	$f(\mathbf{x}) = -a \cdot \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$ $a = 20, b = 0.2, c = 2\pi$
rastrigin	$f(\mathbf{x}) = a \cdot d + \sum_{i=1}^{d}\left(x_i^2 - 10\cos(2\pi x_i)\right)$ $a = 10$
rosenbrock	$f(\mathbf{x}) = \sum_{i=1}^{d-1}\left[b(x_{i+1} - x_i^2)^2 + (a - x_i)^2\right]$ $a = 100$

There is sufficient resting time between batches to allow each of them to start from *fresh* CPU state.

The experiments are carried out on a desktop computer endowed with an Intel Core i7-9700F processor with 16 GB RAM, NDIVIA GeForce GTX 1050 Ti, and Seagate ST1000DM010 2EP102 (SATA III, 6 Gb/s) hard disk, running Windows 10 Pro (22H2), and using Java 20.0.1, while EA's C++ version was compiled using Microsoft Visual C++ 2022, and optimized for performance. Energy measurements are made using the Intel® Power Gadget tool. In order to identify the actual contribution of running each algorithm, we determine the basal consumption of the system (measured when the computer is run with no EA or any other user application running) and subtract it from the actual measurements when running the EA, so as to obtain the excess consumption due to the algorithm run.

3 Results

Tables 2-3 show an overview of the results obtained for each implementation and for each parameter in terms of the energy consumed per run[7]. It is clear from the inspection of these results that there is a trend of reduced energy consumption per run as the resting time is increased. This reduction ranges from 1.2% to 4.9% for $\rho = 10s$ and from 4.7% to 8.9% for $\rho = 100s$. A more precise analysis indicates that all pairwise differences (that is, between the results for a given implementation, objective function, and dimensionality, for different resting times) are always significant at $\alpha = 0.01$ according to a Wilcoxon test. The introduction of these short resting times seems to be effective in bringing the system back to a more energy-efficient state. This is well illustrated in Fig. 2, where a comparison of the CPU temperature measured online during the first and last run of each batch is shown for the 100-dimensional ackley function (the results are analogous for the remaining functions) for different rest times between runs. The CPU clearly overheats when runs are executed consecutively, with no pauses in between. This effect decreases as the value of ρ increases, eventually

[7] All experimental data is available in our OSF repository https://osf.io/xd54u/

Table 2: Processor energy consumption (J) per run (rests excluded) as a function of the rest time between runs for different functions and dimensionality in the Java implementation. The mean and standard error or the mean are shown in each case.

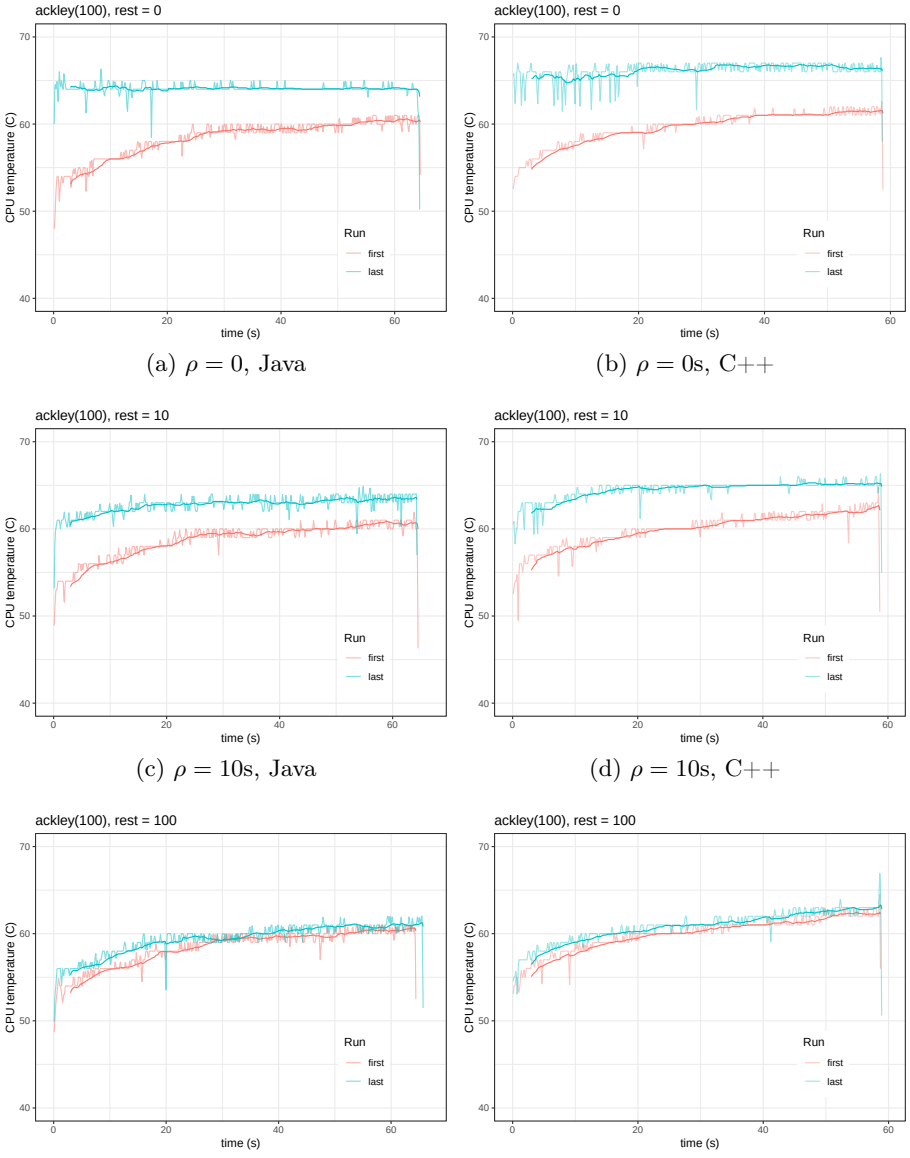
	$d = 10$			$d = 20$		
rest (s)	**rastrigin**	**ackley**	**rosenbrock**	**rastrigin**	**ackley**	**rosenbrock**
0	224.2 ± 1.1	224.5 ± 1.0	223.6 ± 1.2	377.9 ± 2.1	384.9 ± 13.3	341.4 ± 1.6
10	212.5 ± 0.6	215.1 ± 0.6	213.8 ± 0.6	367.1 ± 1.5	352.2 ± 0.8	329.7 ± 0.9
100	205.1 ± 0.5	206.6 ± 0.5	204.9 ± 0.4	349.8 ± 1.1	337.6 ± 0.5	317.3 ± 1.1
	$d = 50$			$d = 100$		
rest (s)	**rastrigin**	**ackley**	**rosenbrock**	**rastrigin**	**ackley**	**rosenbrock**
0	788.0 ± 2.6	768.3 ± 2.9	697.6 ± 3.0	1481.7 ± 4.6	1457.8 ± 4.4	1295.9 ± 4.3
10	774.5 ± 1.8	759.3 ± 2.1	681.1 ± 1.9	1463.0 ± 3.6	1439.6 ± 3.0	1280.6 ± 4.4
100	740.4 ± 1.3	729.2 ± 1.7	656.3 ± 1.8	1418.0 ± 3.6	1397.8 ± 3.2	1223.6 ± 2.1

Table 3: Processor energy consumption (J) per run (rests excluded) as a function of the rest time between runs for different functions and dimensionality in the C++ implementation. The mean and standard error or the mean are shown in each case.

	$d = 10$			$d = 20$		
rest (s)	**rastrigin**	**ackley**	**rosenbrock**	**rastrigin**	**ackley**	**rosenbrock**
0	303.3 ± 1.7	309.7 ± 1.6	350.6 ± 1.9	470.4 ± 3.6	438.4 ± 1.9	464.6 ± 2.0
10	293.2 ± 1.3	295.2 ± 0.9	334.4 ± 0.9	453.2 ± 1.2	422.8 ± 0.9	448.1 ± 1.2
100	281.2 ± 1.1	282.8 ± 0.8	318.9 ± 0.4	430.1 ± 0.5	402.3 ± 0.6	425.1 ± 0.4
	$d = 50$			$d = 100$		
rest (s)	**rastrigin**	**ackley**	**rosenbrock**	**rastrigin**	**ackley**	**rosenbrock**
0	781.0 ± 2.4	767.7 ± 2.5	759.8 ± 2.2	1379.9 ± 3.5	1363.0 ± 3.8	1337.3 ± 4.2
10	765.8 ± 1.6	749.4 ± 1.7	742.8 ± 1.4	1364.4 ± 2.7	1344.1 ± 2.8	1318.2 ± 2.5
100	743.7 ± 9.6	719.4 ± 0.7	712.8 ± 0.8	1312.6 ± 0.7	1293.9 ± 1.2	1272.0 ± 2.0

becoming negligible for longer rest periods. This has a direct impact on the algorithm's energy consumption, as demonstrated in Tables 2-3. In fact, notice that, in general, the standard error of the mean tends to decrease for larger values of ρ, signaling the greater homogeneity of energy consumption during runs. This can also be observed more clearly in Fig. 3, where the average power per run is depicted for each run in a batch. Despite some individual variability, there is a noticeable trend of increasing power for smaller values of ρ, and a fairly constant behavior of the largest value of ρ.

Turning now our attention to the influence of the programming language used in the implementation, Fig. 5 provides a direct comparison between the



(a) $\rho = 0$, Java

(b) $\rho = 0$s, C++

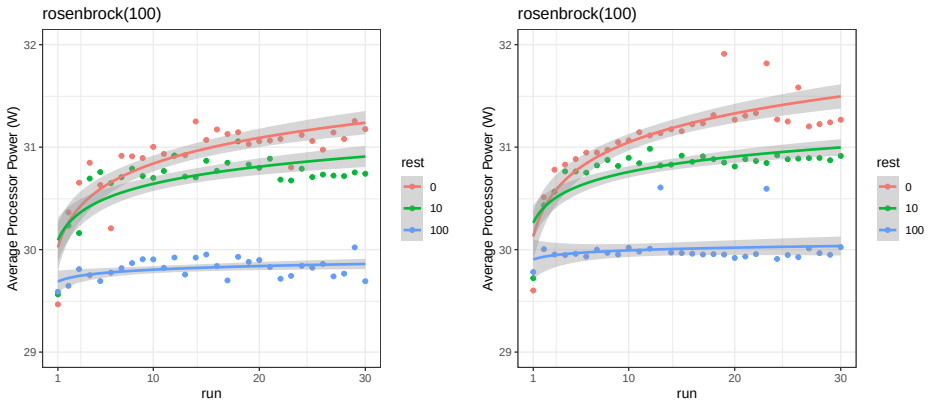
(c) $\rho = 10$s, Java

(d) $\rho = 10$s, C++

(e) $\rho = 100$s, Java

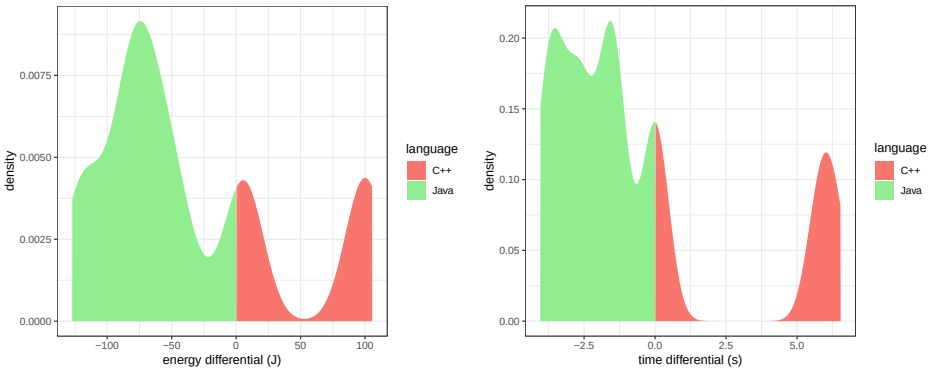
(f) $\rho = 100$s, C++

Fig. 2: Comparison of the CPU temperature during the first and last run of each batch on the ackley function ($d = 100$) for different values of the rest ρ. The light line shows the online measurements and the dark line is a moving average (w=3s).



(a) Java (b) C++

Fig. 3: Evolution of average power per each run of a batch on the rosenbrock function for $d = 100$, broken down for rest time. The continuous line shows a power-law fit to the data points. (a) Java implementation (b) C++ implementation. The results are analogous for the remaining functions



(a) energy (b) time

Fig. 4: (a) Density plot of the mean energy differential (namely the difference between the mean energy consumptions) between implementations in Java and C++ for any given function, dimensionality and rest time. Negative values correspond to lower energy consumption by the Java implementation, and conversely for positive values. (b) Idem for the time differential.

results attained in each of the aforementioned cases. It is interesting to note that the C++ implementation has a comparatively worse energy behavior for lower dimensionality d, but the gap closes as d increases, to the point where C++ becomes superior to Java in some cases. This is a relevant phenomenon

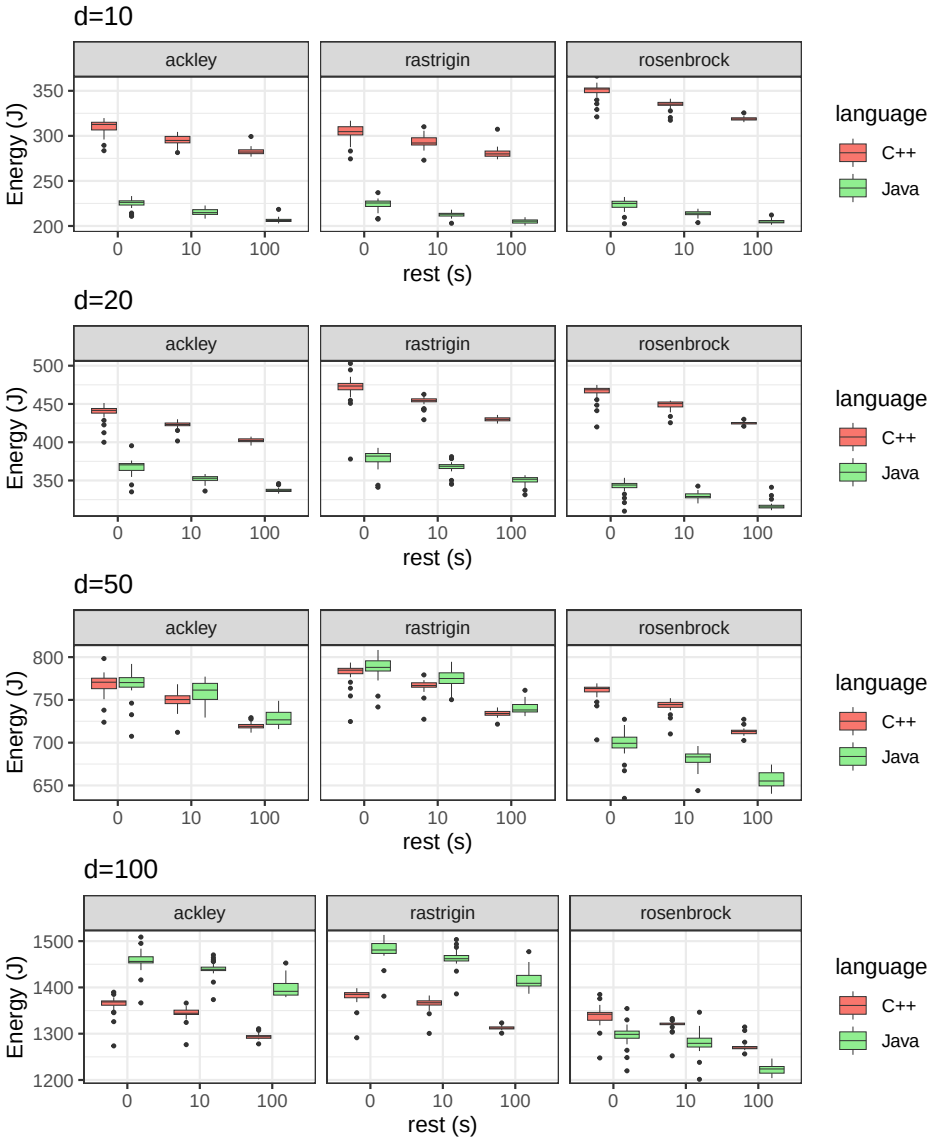


Fig. 5: Distribution (factored per function, dimensionality and programming language) of energy consumption in each run of the EA as a function of the rest time. All differences between Java and C++ are significant at $\alpha = 0.01$ except for the ackley function ($d = 50, \rho = 0$).

which we attribute to the internal functioning of the Java virtual machine and the increasing strain to which larger genotypes subject memory, resulting in a more intensive use of the garbage collector. This is also consistent with the

data shown in Fig. 4b, where it can be seen that there is a modal advantage of Java (resulting from its faster performance on lower dimensionality), but C++ can reach a difference twice as long due to its better speed for $d = 100$. This also correlates with the energy differential shown in Fig. 4a, showing that while on most occasions (for the benchmark considered) Java has a smaller energy footprint, C++ surpasses Java precisely in the largest dimension.

4 Conclusions

The use of AI methods poses challenges from an environmental perspective. For this reason, any AI method, including evolutionary computing, should be utilized following criteria of sustainability and accountability. This work has focused precisely on some practical issues about the implementation and utilization of energy assessments and how they can impact energy consumption. We have shown that hysteresis can be relevant when executing EAs in low-end computing devices such as portable gadgets or desktop computers. We have demonstrated that simple strategies, such as introducing small pauses between runs (a strategy that hardly causes any practical inconvenience to practitioners), can produce meaningful energy savings, which accumulate over time and should not be overlooked. Further savings can also be achieved through careful algorithmic choices and optimized programming practices. Our experiments with two twin implementations of the same EA library in both Java and C++ have shown that the latter seems to scale better, both in terms of running time and energy requirements. Needless to say, confirming these results on a larger benchmark and in even higher-dimensional functions is of the foremost interest in the future. It is also particularly interesting to study the influence that the build setup has. Specifically, for the C++ version, we aim to observe to which extent the use of different compilers (e.g., g++ and clang++) and compilation flags may significantly affect energy consumption, much in line with related research in the literature on the impact of JavaScript runtime environments [15]. Obviously, the compiler availability and the hardware specifications of different computational platforms (laptop, desktop computer, low-end gadgets, etc.) are factors that may have a relevant influence. Studying these factors and establishing some solid methodological guidelines is one of our ultimate goals. Other lines of work include dynamic adjustment for rest times between runs by monitoring the system state.

Acknowledgements

The authors thank the reviewers for their valuable feedback.

References

1. Abdelhafez, A., Alba, E., Luque, G.: A component-based study of energy consumption for sequential and parallel genetic algorithms. The Journal of Supercomputing **75**(10), 6194–6219 (Oct 2019). https://doi.org/10.1007/s11227-019-02843-4

2. Cotta, C., Martínez-Cruz, J.: Energy consumption analysis of batch runs of evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. p. 87–88. GECCO '24 Companion, ACM, New York, NY, USA (2024). https://doi.org/10.1145/3638530.3664093
3. Cowls, J., Tsamados, A., Taddeo, M., Floridi, L.: The AI gambit: leveraging artificial intelligence to combat climate change—opportunities, challenges, and recommendations. AI & Society **38**(1), 283–307 (Feb 2023). https://doi.org/10.1007/s00146-021-01294-x
4. Dhar, P.: The carbon impact of artificial intelligence. Nature Machine Intelligence **2**, 423–425 (2020). https://doi.org/10.1038/s42256-020-0219-9
5. Díaz-Álvarez, J., Castillo, P.A., Fernández De Vega, F., Chávez, F., Alvarado, J.: Population size influence on the energy consumption of genetic programming. Measurement and Control **55**(1-2), 102–115 (Jan 2022). https://doi.org/10.1177/00202940211064471
6. Fernández de Vega, F., Chávez de la O, F., Díaz, J., García, J.A., Castillo, P.A., Merelo Guervós, J.J., Cotta, C.: A Cross-Platform Assessment of Energy Consumption in Evolutionary Algorithms - Towards Energy-Aware Bioinspired Algorithms. In: Handl, J., et al. (eds.) Parallel Problem Solving from Nature - PPSN XIV. Lecture Notes in Computer Science, vol. 9921, pp. 548–557. Springer, Berlin Heidelberg (2016). https://doi.org/10.1007/978-3-319-45823-6_51
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1 edn. (1994)
8. Hassani, V., Tjahjowidodo, T., Do, T.N.: A survey on hysteresis modeling, identification and control. Mechanical Systems and Signal Processing **49**(1), 209–233 (2014). https://doi.org/10.1016/j.ymssp.2014.04.012
9. Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., Pineau, J.: Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. Journal of Machine Learning Research **21**(1), 1–43 (2020)
10. Hidalgo, I., et al.: Sustainable Artificial Intelligence Systems: An Energy Efficiency Approach. preprint, TechRxiv (Dec 2023). https://doi.org/10.36227/techrxiv.24610899.v1
11. Lakshminarayanan, V., Sriraam, N.: The effect of temperature on the reliability of electronic components. In: 2014 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT). pp. 1–6. IEEE (2014). https://doi.org/10.1109/CONECCT.2014.6740182
12. Lannelongue, L., Grealey, J., Inouye, M.: Green algorithms: Quantifying the carbon footprint of computation. Advanced Science **8**(12), 2100707 (2021). https://doi.org/10.1002/advs.202100707
13. Lee, Y., Shin, K.G., Chwa, H.S.: Thermal-aware scheduling for integrated cpus–gpu platforms. ACM Trans. Embed. Comput. Syst. **18**(5s) (Oct 2019). https://doi.org/10.1145/3358235
14. Li, X., Wang, Q., Tang, Y.: The Impact of Artificial Intelligence Development on Urban Energy Efficiency—Based on the Perspective of Smart City Policy. Sustainability **16**(8), 3200 (Apr 2024). https://doi.org/10.3390/su16083200
15. Merelo-Guervós, J.J., García-Valdez, M., Castillo, P.A.: Energy consumption of evolutionary algorithms in JavaScript. In: Villani, M., Cagnoni, S., Serra, R. (eds.) Artificial Life and Evolutionary Computation. pp. 3–15. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-57430-6_1
16. Merelo-Guervós, J.J., et al.: Ranking programming languages for evolutionary algorithm operations. In: Squillero, G., Sim, K. (eds.) Applications of Evolution-

ary Computation. pp. 689–704. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-55849-3_44
17. Merelo-Guervós, J.J., et al.: A comparison of implementations of basic evolutionary algorithm operations in different languages. In: 2016 IEEE Congress on Evolutionary Computation (CEC). pp. 1602–1609. IEEE (2016). https://doi.org/10.1109/CEC.2016.7743980
18. Schwartz, R., Dodge, J., Smith, N.A., Etzioni, O.: Green AI. Communications of the ACM **63**(12), 54–63 (2020). https://doi.org/10.1145/3381831
19. Song, M., Pan, H., Shen, Z., Tamayo-Verleene, K.: Assessing the influence of artificial intelligence on the energy efficiency for sustainable ecological products value. Energy Economics **131**, 107392 (Mar 2024). https://doi.org/10.1016/j.eneco.2024.107392
20. Strubell, E., Ganesh, A., McCallum, A.: Energy and Policy Considerations for Modern Deep Learning Research. Proceedings of the AAAI Conference on Artificial Intelligence **34**(09), 13693–13696 (2020). https://doi.org/10.1609/aaai.v34i09.7123
21. Sulaiman, D.R.: Microprocessors thermal challenges for portable and embedded systems using thermal throttling technique. Procedia Computer Science **3**, 1023 – 1032 (2011). https://doi.org/10.1016/j.procs.2010.12.168
22. Vinuesa, R., Azizpour, H., Leite, I., Balaam, M., Dignum, V., Domisch, S., Felländer, A., Langhans, S.D., Tegmark, M., Fuso Nerini, F.: The role of artificial intelligence in achieving the Sustainable Development Goals. Nature Communications **11**(1), 233 (Jan 2020). https://doi.org/10.1038/s41467-019-14108-y
23. Walczyk, C., et al.: Impact of temperature on the resistive switching behavior of embedded HfO2-based RRAM devices. IEEE Transactions on Electron Devices **58**(9), 3124–3131 (2011). https://doi.org/10.1109/TED.2011.2160265
24. Zhang, Q., et al.: A survey on data center cooling systems: Technology, power consumption modeling and control strategy optimization. Journal of Systems Architecture **119**, 102253 (2021). https://doi.org/10.1016/j.sysarc.2021.102253