Is there a (carbon-) free lunch? Energy/performance tradeoffs in population-based metaheuristics

JJ Merelo[1,2][0000−0002−1385−9741], Cecilia Merelo-Molina[3][0000−0002−5902−0159], Pablo García-Sánchez[1,2][0000−0003−4644−2894], Mario García-Valdez[4][0000−0002−2593−1114], and Juan Luis Jiménez-Laredo[1][0000−0002−9416−2005]

[1] Department of Computer Engineering, Automatics and Robotics, University of Granada jmerelo@ugr.es
[2] CITIC, UGR, Spain
[3] Zenzorrito, Granada, Spain
[4] Instituto Tecnológico Nacional de México, Tijuana, México

Abstract. There are many levels at which the implementation of an algorithm can be made *greener*, that is, more energy efficient; improving efficiency piecewise will be one of them, but at the highest level, energy efficiency will impact the algorithm results, with different choices having an independent impact on energy and algorithmic efficiency. In this paper, we will explore this trade-off in the Brave New Algorithm, a population-based metaheuristic implemented in the language Julia, which uses a JIT compiler whose overhead needs to be taken into account, analyzing how different operator optimizations impact the performance and energy consumption and finally, once a certain level of algorithmic performance is reached, how different parametrizations will impact both outcomes. We will especially look at the possibility of improving energy and algorithmic efficiency at the same time, if that is possible, and how this relates to the exploration/exploitation balance in this algorithm which is specifically designed for easy tuning of this balance.

Keywords: Green computing · Energy profiling · Metaheuristics

1 Introduction

Creating greener algorithms and implementations is the new frontier of software development: most performance gains usually carry heavier energy costs; since the end of Moore's Law [17], we must choose between performance and energy improvements, rarely being able to have both. In the case of optimization algorithms, there is another view of performance: how close you get to the (known) optimal solution of the problem we are seeking. Many low-level optimizations at the data structure or implementation level are, in general, performance-neutral from this point of view: the algorithm will behave the same way regardless of its implementation, so the only difference will be the energy or time spent running

it. However, when we carry those changes at the algorithm level, the scenario might change. Since these optimizations might have improved some operators' energy profiles more than others, applying them in different proportions to reduce the carbon footprint might result in different algorithmic performance. Furthermore, in general, some implementation choices might indeed have an impact on algorithmic performance: fixing a bug that improves performance, for instance, might result in better (or worse) energy costs, and using different data structures might alter the search space structure and thus yield gains or losses.

This is especially true for one of the most important factors in population-based optimization: the exploration/exploitation balance. Exploration and exploitation generate new individuals spending a different amount of energy; in practice, sampling the search space will spend energy depending on what you use, besides having a fixed cost due to the allocation and reallocation of memory; of course the number of individuals generated depending on the balance will also have an influence on the energy profile.

We will be working here with a version of an evolutionary algorithm called the Brave New Algorithm [9], which was designed with this exploitation/exploration balance in mind. The current version of the algorithm is implemented in the programming language Julia, opening different possibilities of optimization at multiple levels.

But for any kind of change, including the balance of operators used, it is extremely complicated or impossible to know in advance how the energy profile might change: following a sound methodology for measuring the impact of changes and measuring every single one for different scenarios is the only way to put a solid foundation to any kind of decision you might take on the implementation of the algorithm. This is what we will do in this paper: first, we will try to establish a methodology for measuring energy consumption for the algorithm runs themselves, discarding the overhead created by the system and the language runtime, which, in the case of JIT compiled languages like Julia, can be important; then we will try to perform optimizations of the implementation at different levels and check the impact they have on the energy consumption and fitness achieved. Once a certain level of algorithmic performance is reached, we will try to dive into this exploration/exploitation balance via different parameters and look for the way of achieving better energy efficiency without sacrificing algorithmic performance, if that is possible within the current setup.

Additionally, we are interested in exploring Julia as a language for EA implementation from different point of views, including performance and energy consumption; its JIT compiler generates high-performance code and it includes easy parallelization features, as well as native support for mathematical operations and data structures. Although not explicitly explored in this paper, we need to set the methodology as well as the baseline results for further advances in this area.

The rest of the paper is organized as follows: the next Section describes the state of the art in the exploration of energy consumption of evolutionary algorithms and other metaheuristics. Section 3 describes the algorithm and how

it has been parametrized for the purpose of this paper. Since there is no fixed methodology for measuring energy consumption, we will show how it was done for this experiment and the trade-offs it implies, together with the results, in Section 4; these results will be discussed in the last Section 5 along with our conclusions.

2 State of the art

System designers have widely employed evolutionary algorithms and other metaheuristics to optimize resource allocation and minimize the total energy consumption of cloud data centers [8]. In this work, however, we focus on a different yet complementary aspect of the problem: the algorithm design techniques that enable greener implementations of evolutionary algorithms themselves, thereby reducing their energy footprint. Previous studies have examined various facets of this topic, including trade-offs among the programming languages used to implement evolutionary algorithms, performance, and efficiency differences between high- and low-level languages, and the impact of specific operations and data structures on computational cost [11].

As a secondary objective in this paper, we have wanted to deal with methodological and implementation issues in a non-mainstream language; in fact, there are several implementations of evolutionary algorithms in Julia, such as [15] or Evolutionary.jl [18]. Before that, Julia had been used [12] as a baseline for comparing performance in basic evolutionary algorithms, with its performance being close to the median of all languages tested. Some other languages like Java or C++ might be several orders of magnitude faster. However, performance does not translate directly to energy consumption, and besides, back then Julia was still in its early versions vs. the more mature implementation that is now in production.

Moreover, our main interest is optimizing energy consumption. This can be done at many different levels, from the lowest, hardware level, to the implementation level, up to the algorithmic level. Clearly at this level the range of possible optimizations will be relatively small, but still it gives leverage to scientists that cannot change any other level to make their algorithm implementation *greener*. Some researchers have applied it to machine learning algorithms [4]; there are not so many cases where it has been applied to metaheuristics such as the one used in this paper.

3 A Brave New Algorithm

The Brave New Algorithm [9] is an evolutionary algorithm, using the usual mechanism of mutation, crossover, and selection to find the optimum of a given function, represented as a *chromosome*, in this case a vector of real numbers. However, how selection and reproduction take place is similar to how the novel Brave New World [6] organized society: in castes.

- The α caste generates new members by coupling the best individuals among them, to then undergo mutation and crossover.
- The β caste needs to reproduce a member of the α caste, and another member of its own caste, to then undergo mutation and crossover.
- The δ and ϵ castes generate new individuals by mutation. The only difference between them is that those who reproduce are chosen among the other members of the population, and that mutation rates can be set separately.
- The γ caste generates new members by mutation, but mutation might be applied several times while the new individual is better than the previous one.

The operators used are mutation, that will change 40% of the elements of the vector generating a new, random element, 2-point crossover, and selection via binary tournament [1]. After every generation, the population is re-ranked again, with the first $\%\alpha$ of the population assigned to that caste, and so on; through generations every caste will hold the same number of individuals. Since the two *upper* castes are devoted to exploitation and the rest to exploration, the proportion of individuals in them is the main parameter governing the balance. There are some restrictions in these percentages, due to the way new members are generated: the beta caste needs to have twice as many members as the alpha caste; the percentage of population in the α caste is, thus, the main parameter for this, since we can use it to generate the rest. In practice, what we have done is to vary the percentage of the γ caste, the "first" that performs local search, while keeping δ and ϵ fixed to a low value.

The implementation of the algorithm used is available under a free license at https://github.com/cecimerelo/BraveNewAlgorithm.jl. It has been chosen due to its simplicity and to explore the possibilities of Julia as a low-energy consumption language compared with Python or Javascript.

4 Experimental methodology and results

To carry out these experiments, we have used a methodology like the one used previously to compare different parametrizations, implementations and versions [10]: We employ `pinpoint`, [7], a command line tool that taps the RAPL interface [3] to measure energy consumption of a given process. The version we have used was compiled from commit `dfee658`. This tool was validated in [13] by comparing its output to other energy-measuring tools and finding that it was less error-prone than the rest and gave measures that were consistent with tools such as `perf` and `likwid` when they did not fail. The combination of the RAPL API, which places estimates in a register, and the overhead by the tool cannot be avoided, however, which is why we have to apply a careful experimental design to mitigate adverse effects.

We carry experiments in an AMD Ryzen 9 9950X 16-Core Processor[5], with Ubuntu Linux 25.04 and kernel version `6.14.0-29-generic`. Julia version has been 1.11.7 except when noted.[6]

We have used the Sphere function [5], defined as sum of the squares of the distance to the center minus a fixed value, which is part of the BBOB benchmarks; as a matter of fact, the implementation of this function is taken from the BlackBoxOptimizationBenchmarking.jl, also written in Julia. This function is also lightweight enough to allow us to compare the energy profiles of different combinations of genetic operators, which will then have a bigger impact on overall consumption.

Our initial experimental design includes two problem sizes: vectors with 3 or 5 dimensions (D), to check the impact of the parameters for different problem difficulties. This corresponds to the low end of the BBOB benchmark suites; remember that we are using this suit just for the purpose of profiling the energy for this language and algorithm, so for the time being we did not find it necessary to go into higher dimensions.

- Population size P: 200 or 400 individuals, which has an impact on the diversity of the population and thus the exploitation capabilities of the algorithm.
- Maximum number of generations without improvement: 10 or 50, which is the stopping criterion for the algorithm. In this paper we have made this specific choice for stopping criterion, because if the algorithm tends towards exploration, it will explore in fruitless directions generating (possibly) useless chromosomes that will not allow it to escape a local minimum; in particular, this will have the consequence that depending on the problem dimension the execution might fall well short of an optimum value. We have left it this way, however, due to its direct relationship with exploration.
- Percentage of population in the α caste: 10% or 25%, which is the main parameter that controls the exploration/exploitation balance, since only the α and β castes perform exploitation. The rest of the parameters will be set accordingly: β percentage will double that value. γ caste will hold a proportion equal to the α and β percentages subtracted from 90. δ and ϵ castes will always have the same proportion, each equal to 5%.

This means that the α percentage is the main one governing this exploration/exploitation balance. Every parametrization runs 30 times sequentially in a single core. Results written in standard output are processed and are available from this paper's repository at https://github.com/JJ/brave-new-green-algorithm.

Julia loads the script every time it is run, uses just-in-time compilation to generate low-level code, and this code is then run within its own virtual machine;

[5] Please note that AMD implements a version of the RAPL API which only allows the measurement of a single value for the processor, PKG or package, excluding memory and different components within the package.

[6] Julia has now changed the minor version to 1.12, with 1.12.4 being the latest released at the moment of writing this paper.

this takes a certain amount of time and consumes a certain amount of energy, adding to the system overhead. This is why in this case it is especially important to make baseline measurements to discount these two quantities when we make measurements on the actual workload. The main issue is *what* to consider baseline. One well-established principle is that you need to use the same code for it and the workload, so a common option is to use a command line flag to indicate, in runtime, what kind of workload is being run. The second issue is what to run as such a baseline workload. Since we are dealing with population-based algorithms, a straightforward option is to use the generation of the population as such. This is what we have done in this case: a flag indicates whether the program should stop after the initial generation or continue to perform the algorithm. This in turn means that there will be different baselines depending on D and P, four in total. For every parametrization, we will run 240 experiments; after an initial run with 120, we performed another one to increase the statistical accuracy, since the hardware conditions might be slightly different.

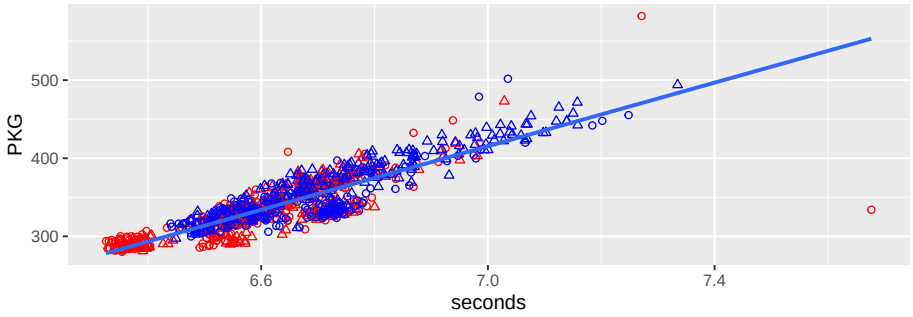

Fig. 1. PKG energy (in Joules) as a function of time for baseline experiments; shapes are related to P and color to D

Figure 1 plots the PKG energy (in Joules) vs. time taken for every experiment, with triangles representing a $P = 400$ and circles $P = 200$; blue is for $D = 5$ and red for $D = 3$. In general, energy consumption will depend linearly on time, but there are clusters of data points with the same shape and color that hint at these having an influence on energy consumption beyond simple time. We should also observe the dispersion in a range of several hundred milliseconds as well as several hundred Joules, although most results are below 350 Joules and 6.8 seconds. This dispersion, however, is a real issue. Besides hysteretic effects that have repeatedly been observed in this context [2], there is an additional issue that occurs when measurements take a relatively long time: some system-wide events might affect one of the parameter combinations more than others.

What we see in Figure 2 is a violin plot that shows a big dispersion, especially for the combination of $D = 3$ and $P = 200$; not only that, but there are three clusters along the violin. These three accumulations for certain energy values

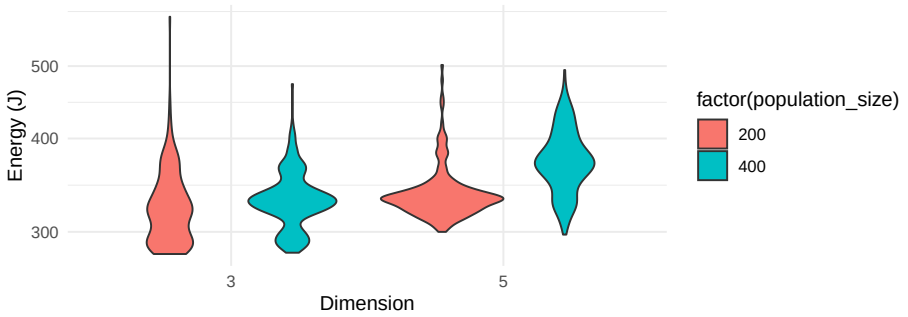

Fig. 2. Violin plot for the distribution of energy spent for every parameter value, initial configuration.

happen across the parameter values. Any of these variables have a high variance and many outliers. This is why we need to carefully consider what kind of statistical measure we should use for summarizing the results so that they can be subtracted from the measurements that carry the workload. Finally, we settled on the *trimmed mean*, eliminating the 20% values with the highest and lowest values. This gives a more robust central tendency measure, which is also closer to the median.

We should take into account that profiling for every kind of algorithm needs its own methodology that is adequate to the dispersion in the measurements taken; however, a high dispersion combined with time (or temperature) related effects will always lead to a certain amount of uncertainty in the conclusions. That will not invalidate these conclusions, but they will have to be explained and factored in when any kind of result is reached [14].

After performing experiments running just the baseline, the results obtained after performing measurements with the actual workload will be examined next. We will be running the full algorithm with the indicated termination condition for 30 runs. We will subtract the baseline trimmed average for PKG energy as well as time from every result. The PKG energy vs. time chart is shown in Figure 3 (left).

We see again that energy is related to the time taken, but we can also draw more conclusions. In general, "triangles" with $P = 400$ are *above* the line; that is, they consume less energy than would be usual for the time taken. If we take into account that the generation of the population is included in the baseline, this essentially means that most of the energy needed for bigger populations is consumed when that population is generated for the first time, not having a determinant influence in the energy consumption of the workload itself. Circles are below, so the opposite is true. What does seem to have a certain importance is the dimension of the problem: $D = 5$ (blue) are placed, in general, to the right-hand side indicating they need more time and more energy. Finally, the point size, which is related to the maximum number of generations with-

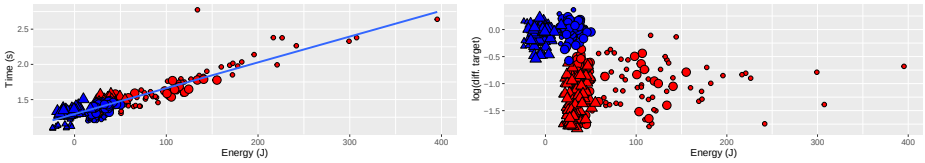

Fig. 3. Energy spent by the workload as a function of time (left) and difference to target fitness vs. energy (right); point size is related to the max number of generations without change; fill color is blue for dimension =5, red for dimension = 3; finally, shapes are related to population size.

out improvement, seems to be related to energy consumption: more generations without improvement means more consumption.

We would like to know if there is some correspondence between the amount of extra energy spent and the fitness achieved. We show this in Figure 3 (right). In this chart, that follows the usual convention, we see that in experiments where dimension is equal to 5 little energy is spent, but in fact fitness level is not very good. Triangles ($P = 200$) need less energy, or at least a more consistent amount of energy, to achieve a certain value. Finally, when too many generations without change are used, that will result in general in more energy needed for achieving a fitness level. What we want, however, for this chart is to move left (less energy) and down (better fitness), if possible, at the same time. However, that will not generally be possible, so we will have to settle with approaching every one of the problems in turn.

We will have to consider the characteristics of the language to do so. Julia is a garbage-collected language [16], implying that it will stop from time to time to collect unused pointers created by allocated memory that is no longer used. In most cases, optimizing code looks to reduce the number of allocations, so that the amount of time (and energy) devoted to it is reduced. This was achieved by copying one of the original vectors into a new one, instead of adding values dynamically to an empty vector, filling it with the result of the mutation. Additionally, the elements of the vector that were going to be mutated were generated in advance, guaranteeing that they were all different; in the version used so far, a specific element could be mutated twice, resulting in worse exploration of the space.

We need to find such a target for optimization, however, and we found it in the mutation operator. Mutation is applied every time a new individual (after the initial population) is generated; that is, several thousand times in this kind of experiment; reducing the number of allocations it uses should have an impact. This is what we will do next.

Figure 4 shows how the (theoretically) optimized mutation changes the energy profile of the implemented algorithm. Although it consumes less energy for the same time, it consumes more for the same number of evaluations, with black points representing it taking the upper half of the chart; that is, consuming more energy. This means that essentially the performance boost in time is being

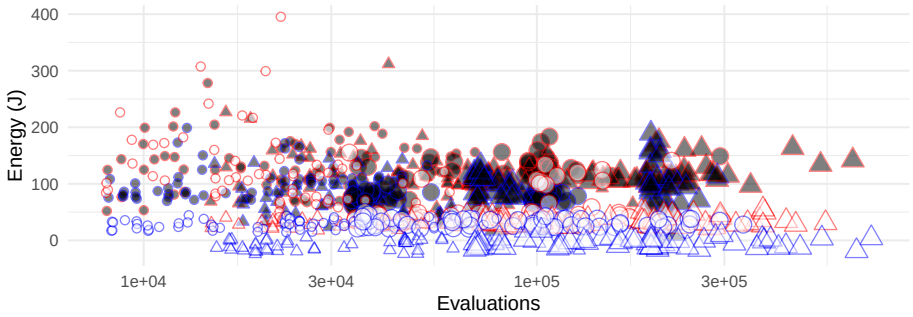

Fig. 4. Energy vs. evaluation for the initial (black-filled) and optimized configuration (white-filled). Rest of the legend as above.

done at the cost of more energy being spent doing the same amount of work, unfortunately. We will examine this in more detail next.

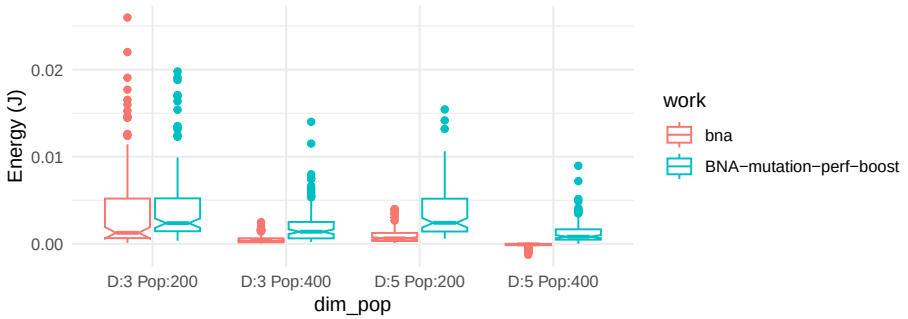

Fig. 5. Comparing energy consumption for initial configuration vs. optimized one.

If we compare the new results against the basic implementation, which is what we do in Figure 5, where we are doing a boxplot of Joules per evaluation for every combination of dimension and population. Even in this kind of measure, that does not consider the changes in the number of evaluations that might have taken place. What we see is an obvious superiority of the old vs. the new implementation. We can argue that the number of allocations is not so important for $D = 3, 5$; however, what we see is that while, as seen above in Figure 4, the new mutation implementation would consume less energy for the same amount of time, the original one is still faster, and that adds up to a lower overall energy consumption. As is usual in this line of research, even if intuitively we assume that some kind of programming improvement might lead to better energy consumption, we still need to make measurements to check what is the actual improvement observed. We do not even observe an improvement in the

dispersion of the results: although the distribution of the original implementation has several "ridges", the new one is longer, indicating a bigger dispersion. The difference seems to diminish with size, however, indicating that it is likely that for bigger sizes, which are also used in BBOB benchmarks, the difference will diminish or even revert. When we are evaluating, however, a whole algorithm, the interesting result is how much energy is spent for a certain fitness level. We show this in Figure 6.

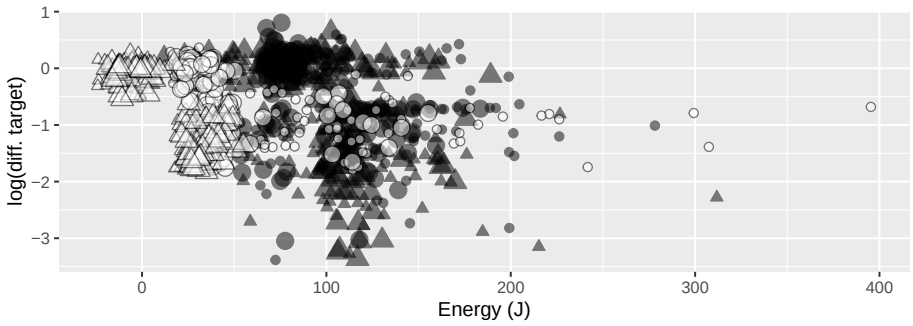

Fig. 6. Energy vs fitness difference to the optimum for initial and optimized algorithm. Lower is better for fitness values.

Looking at the charts from left to right, we see that, in general, and as seen in Figure 5, for a certain fitness level the original implementation spends less energy. However, we can also see that, since the exploration is improved, we can reach better (lower) fitness levels, below (logarithmic difference) -2, which was the lowest the original implementation was able to go. At the end of the day, in a *green* evolutionary algorithm we are trying to optimize two independent targets: energy consumption and fitness; we must find a trade-off between them, but clearly obtaining better fitness values is the highest priority. So, we will try to examine the current implementation from that point of view, and in fact, the mutation can be improved, since it is not guaranteed to generate individuals in the whole BBOB range, which is [-5,5]. We will make a new optimization by using a random number generator that is guaranteed to generate values with a uniform distribution in that range. The results are shown next.

As we can see in Figure 7, we can obtain much better fitness values, and even, in some cases, better energy expenses for the same fitness level; for $D = 3$, we see that in some cases the energy levels of the optimized mutation (`BNA-mutation-perf-boost`) are, in some cases, smaller than the new implementation for the same fitness reached; however, all panels show that for the same energy expenses, the fitness that is going to be obtained with the new implementation is much better. We need to look at the big picture, however. We represent in Figure 8 a violin plot for both implementations, showing that for $D = 3$ energy consumption for the implementation with the fixed random generator is slightly better, as well as

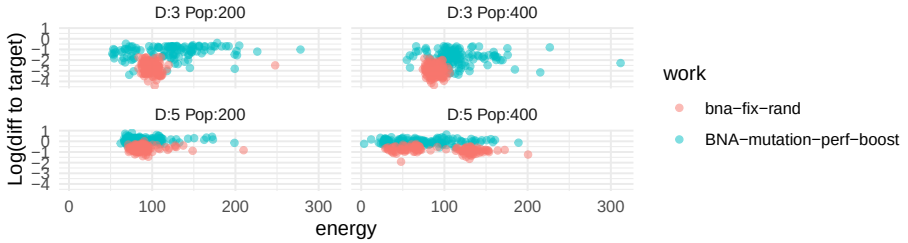

Fig. 7. Energy vs fitness difference to the optimum for all experiments for all parameter combinations; we compare here the optimized configuration vs. uniform random mutation.

less dispersed, while for $D = 5$ there is a minor difference. Again, we find that trade-off between energy consumption and fitness reached, although in this case, obtaining much better fitness for less or more or less the same energy is, given the circumstances, a good trade-off.

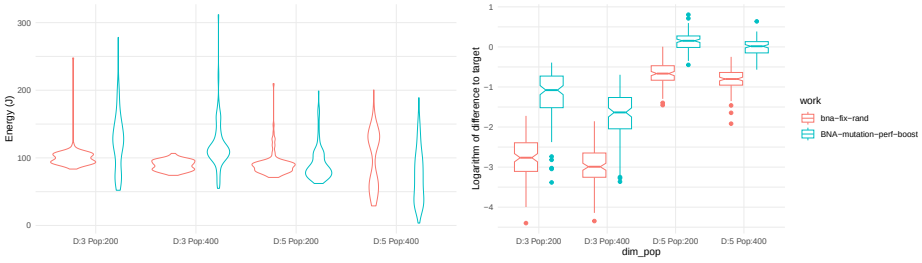

Fig. 8. Comparing energy consumption (left) and fitness levels (right) for previous mutation vs. uniform random one.

We can see in Figure 8 (right) how fitness levels are much better in this case, when a new random generator has been employed (`BNA-fix-rand`). Since we have reached a level that is good enough to be considered in production, we can now turn our sight to how the other parameters, the ones that govern the exploration/exploitation balance, affect the outcomes we are interested in.

What we can see in Figure 9 is how parametrizations affect energy consumption as well as fitness level reached. We can observe across all charts that population size does not have a substantial influence in energy spent, possibly because the biggest part of the energy is used to generate the initial population, and that has been factored out as part of the baseline. An ANOVA analysis, in fact, shows that the p-value for that parameter is slightly above 5%. It does, however, have an impact on fitness, so in general we will want bigger populations. Next, let us look at the α value that decides what percentage of the population

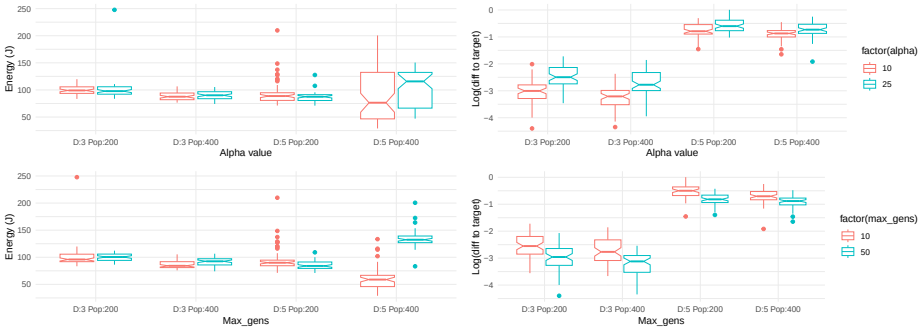

Fig. 9. Comparing energy consumption and fitness reached for the different values of α and the maximum number of generations without improving the value.

is going to be on the α, β and γ castes, being the main lever for the exploration/exploitation balance. In this case, ANOVA (and the charts, upper row) show no significant influence in energy spent, except for a significant interaction with the population; this what we see on the top row, rightmost chart: the lower the percentage, that is, the lower the amount of exploitation, the better in terms of energy spent. We can then go to the bottom row and observe than, in general, lower percentage will get the algorithm to reach better fitness levels; ANOVA analysis shows that it is a significant parameter in this case too, so we need to conclude that devoting a small part of the population to exploitation will be beneficial in both fronts, energy and fitness.

The maximum number of generations without improvement is significant at both levels; we need to look at the charts to reach a conclusion about which direction is best. The third row, which charts energy spent, shows that, again, lower values will get either the same energy or, in one case, less energy spent, significantly so. However, the bottom row charting fitness levels shows that higher values will reach lower fitness levels, again significantly. Since energy expenses and fitness levels go in different directions, this is a case where more fine tuning will be needed. The ANOVA analysis shows that the interaction between these two parameters is not significant in any case (energy or fitness), so we can treat them independently.

We can repeat these experiments again for a new version of Julia, 1.11.8, and the kernel, `6.14.0-37-generic`, to check if the algorithmic improvement is consistent across different hardware/software conditions.

In Figure 10 we have kept fixed the α percentage and varied the maximum number of generations between 10 and 25. As should be expected, the fitness results are very similar, showing a consistent behavior of the algorithm, although in this case it confirms that there is no need to wait for 50 generations to stop the algorithm; 25 are enough. And this saving in generations is rewarded with an improvement in the energy profile (shown at left). While in 9 PKG energy consumption for the higher value of `max_gens` was similar or higher, in this case

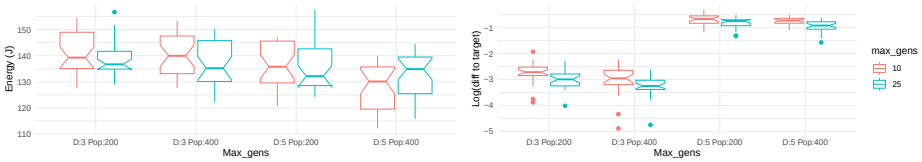

Fig. 10. Boxplot of energy consumption (left) and fitness (right) using Julia 1.11.8, varying maximum number of generations to stop.

we consistently find that, except for the dimension = 5 and population = 400, energy consumption is higher the lower are the iterations. However, what we see with this updated version of Julia is that our workload, once the baseline has been eliminated, consumes consistently more than the previous version. So, although these results are valid to confirm the differences in energy consumption for different parametrizations under a different platform, the platform itself will need to be rejected on the grounds of energy consumption, staying in the previous version of Julia without upgrading.

5 Conclusion and discussion

In this paper, we have profiled the energy spent by a population-based metaheuristic called Brave New Algorithm implemented in Julia. We have tried for the first time to carry out an integral assessment and improvement of energy consumption and performance in an evolutionary algorithm, including specific characteristics such as the exploration/exploitation balance.

We have first tried to establish a baseline for comparison that considers the overhead generated by the compilation of the language, as well as the variability that is inherent in measuring energy consumption for a specific workload. We needed to take repeated measurements of the baseline at two different points in time and then used the trimmed average. Even so, the fact that there are time-dependent effects on the measurements means that we will have to try, in the future, different strategies for measuring the baseline and the effect that mitigate them.

We have also performed energy-driven performance optimization testing three different configurations, a baseline that already included some suggested optimizations, followed by a change in the mutation operator that reduced the number of memory allocations and fixed an issue. For the sizes we tested, however, the change in energy consumption was either negative or nonexistent; however, this resulted in a better evolutionary algorithm performance, reaching better fitness levels. In the trade-off between energy consumption and fitness, we need to opt for the latter unless it implies a disproportionate increase in energy consumption. What we have to take into account is that the improvement in performance is due to the fact that it is exploiting more efficiently the space without getting stuck, so we do not have a stark energy/performance trade-off: we consume

slightly more energy even if every individual mutation is worse because we are using less generations to reach better fitness levels.

But still our focus is on making the algorithm greener, so finally we changed the random number generator to a uniform one. Theoretically, this should have increased energy consumption; however, we obtained some improvements or, depending on the dimension, slight increases in energy consumption; the performance, however, improved significantly. This gave us a platform on which to analyze the effect of the parameters that affected exploration and exploitation and how it affects energy balance. The first insight we obtained was the fact that stratification of the population and the maximum number of generations without change are independent. In general, a smaller α caste, a population with less exploitation, will result in better fitness and better energy profile, possibly due to the fact that it spends less generations looking for the solution. Staying for more generations exploring searching for a new solution, however, results also in better fitness, although never in a better energy profile. We have tested two values for this parameter; it remains thus a target for optimization.

One of the main issues is still the measuring methodology, which for the time being cannot distinguish per-process energy usage and, even if we take into account just the process, the additional overhead incurred by bookkeeping and program processing operations, including also the fact that measuring energy has time, and possibly temperature, dependent effects. These methodological issues will be tackled along with additional optimizations on different algorithm implementation or running levels.

Acknowledgements

This work is supported by the Ministerio español de Economía y Competitividad (Spanish Ministry of Competitivity and Economy) under project PID2023-147409NB-C21.

References

1. Blickle, T., Thiele, L.: A comparison of selection schemes used in evolutionary algorithms. Evolutionary Computation **4**(4), 361–394 (1996)
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, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3638530.3664093
3. Garcia, J.A.: Exploration of energy consumption using the Intel Running Average Power Limit interface. In: 2019 IEEE Space Computing Conference (SCC). pp. 1–10 (2019). https://doi.org/10.1109/SpaceComp.2019.00005
4. Gutiérrez, M., Moraga, M.Á., García, F.: Analysing the energy impact of different optimisations for machine learning models. In: 2022 international conference on ICT for sustainability (ICT4S). pp. 46–52. IEEE (2022)

5. Hansen, N., Auger, A., Ros, R., Finck, S., Pošík, P.: Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In: Proceedings of the 12th annual conference companion on Genetic and evolutionary computation. pp. 1689–1696 (2010)
6. Huxley, A.: Brave new world. DigiCat (2022)
7. Köhler, S., Herzog, B., Hönig, T., Wenzel, L., Plauth, M., Nolte, J., Polze, A., Schröder-Preikschat, W.: Pinpoint the Joules: Unifying runtime-support for energy measurements on heterogeneous systems. In: 2020 IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS). pp. 31–40 (2020). https://doi.org/10.1109/ROSS51935.2020.00009
8. Maryam, K., Sardaraz, M., Tahir, M.: Evolutionary algorithms in cloud computing from the perspective of energy consumption: A review. In: 2018 14th international conference on emerging technologies (ICET). pp. 1–6. IEEE (2018)
9. Merelo, C., Merelo, J.J., García-Valdez, M.: A brave new algorithm to maintain the exploration/exploitation balance. In: New Perspectives on Hybrid Intelligent System Design based on Fuzzy Logic, Neural Networks and Metaheuristics, pp. 305–316. Springer (2022)
10. Merelo, J.J., Garcia Valdez, M., Romero López, G.: Energy consumption of evolutionary algorithms implemented in low-level languages. In: Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing. p. 157–160. SAC '25, Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3672608.3707829, https://doi.org/10.1145/3672608.3707829
11. Merelo-Guervós, J.J., García-Valdez, M.: How evolutionary algorithms consume energy depending on the language and its level. In: Festa, P., Ferone, D., Pastore, T., Pisacane, O. (eds.) Learning and Intelligent Optimization. pp. 254–268. Springer Nature Switzerland, Cham (2025)
12. Merelo-Guervós, J.J., Blancas-Álvarez, I., Castillo, P.A., Romero, G., Rivas, V.M., García-Valdez, M., Hernández-Águila, A., Román, M.: 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)
13. Merelo-Guervós, J.J., García-Valdez, M., Castillo, P.A.: An analysis of energy consumption of JavaScript interpreters with evolutionary algorithm workloads. In: Fill, H., Mayo, F.J.D., van Sinderen, M., Maciaszek, L.A. (eds.) Proceedings of the 18th International Conference on Software Technologies, ICSOFT 2023, Rome, Italy, July 10-12, 2023. pp. 175–184. SCITEPRESS (2023). https://doi.org/10.5220/0012128100003538, https://doi.org/10.5220/0012128100003538
14. Merelo-Guervós, J.J., Romero-López, G., García-Valdez, M.: Time-related effects in the measurement of energy consumption in evolutionary algorithms. In: Europar 2025: Parallel Processing Workshops, Euro-Par 2025 (2025)
15. Sánchez-Díaz, X.F., Mengshoel, O.J.: EvoLP.jl: A playground for evolutionary computation in Julia. CEUR Workshop Proceedings (2023)
16. de Souza Amorim, L.E., Lin, Y., Blackburn, S.M., Netto, D., Baraldi, G., Daly, N., Hosking, A.L., Pamnany, K., Smith, O.: Reconsidering garbage collection in Julia: A practitioner report. In: Proceedings of the 2025 ACM SIGPLAN International Symposium on Memory Management. pp. 72–83 (2025)
17. Theis, T.N., Wong, H.S.P.: The end of Moore's Law: A new beginning for information technology. Computing in science & engineering **19**(2), 41–50 (2017)
18. wildart: Evolutionary.jl (2023), https://wildart.github.io/Evolutionary.jl/dev/