Delays in computing with parallel metaheuristics on HPC infrastructure

Sylwia Biełaszek[1][0000-0001-8947-6272], Adam Nowak[1][0009-0001-7842-5522], Krzysztof Gądek[1][0009-0001-9504-5484], and Aleksander Byrski[1][0000-0001-6317-7012]

AGH University of Science and Technology, Faculty of Computer Science, Institute of Computer Science, Kraków, Poland, bielsyl@agh.edu.pl, olekb@agh.edu.pl
https://www.agh.edu.pl/

Abstract. Due to their structure, metaheuristics such as parallel evolutionary algorithms (PEA) are well suited to be run on parallel and distributed infrastructure, e.g. supercomputers. However, there are still many issues that are not well researched in this context, e.g. existence of delays in HPC-grade implementations of metaheuristics and how they affect the computation itself. The lack of this knowledge may expose the fact, that the power of supercomputers in this context may be not properly used. We want to focus our research on examining such white spots. In the paper we focus on giving the evidence for the existence of delays, showing the differences among them in different island topologies, try to explain their nature and prepare to propose dedicated migration operators considering these observations.

Keywords: high performance computing, delays, PEA, island model, genetic algorithm, migrations

1 Introduction

Due to their nature (the need to process a significant number of individuals, in particular computing their fitness on and on), metaheuristics have a very high time complexity [9]. Therefore, the need to utilize parallel and distributed HPC infrastructure when the computing results are in quick demand is imminent. Many different computing platforms were constructed, e.g. DEAP [1], jMetal [2], EASEA [3], PlatEMO [4], AGH-AgE platform [5] to name a few. For the last two decades we have been doing metaheuristic-focused research using different platforms, some of the mentioned and some of our own (e.g. [18]).

[1] https://github.com/deap
[2] https://jmetal.sourceforge.net/
[3] https://easea.unistra.fr/index.php/EASEA_platform
[4] PlatEMO source code http://bimk.ahu.edu.cn/index.php?s=/Index/Software/index.html
[5] https://gitlab.com/age-agh/age3

When thinking about proper utilizing of HPC infrastructure, one has not only to "divide and conquer" the algorithm, assigning processes (e.g. islands in the parallel evolutionary algorithm) to nodes and managing their communication (e.g., by some central node) as this will soon hamper the overall efficiency. The selection of the programming libraries and technologies used should suit the needs of reaching the scalability and reliability of the whole computing system. Therefore in this paper, we wanted to focus on intricacies of HPC that may affect the computing with metaheuristics, and pave the way for further adaptation of the algorithms towards proper use of HPC-grade infrastructure.

We were inspired by the work of predecessors examining the island model of genetic algorithms [1], [7], [6], in terms of taxonomy [12], topology [13], theoretical analyses [14] and mainly the migration mechanism [5], [16], and interested in the best possible setting of this mechanism, having examined various parameters and migration strategies in previous work, we decided to investigate the behavior of delays occurring in PEA in HPC.

We have used a dedicated lightweight computing system based on Python Ray [6], in order to use actor-based concurrency to achieve high scalability of the system (and our computations). When running the experiments, we have observed significant delays in migration. One has to remember that if our aim would be the simulation, controlling the delays would be crucial. Metaheuristics however can work on a little bit "outdated" data (e.g. delayed migrants in parallel evolutionary algorithm). Anyway, we are convinced that proper handling of such phenomena might at least increase the efficiency of our algorithms (if not the efficacy).

In this paper, we are trying to answer the question of how "outdated" migration data come to the islands in the island evolution algorithm, how these phenomena are related to the used island topologies, and what we can do with that in the future.

The paper is organized as follows. In the next section we present the state-of-the-art of the parallel island model and some metaheuristics platforms. Next, our research methodology is described, namely the computing system used and the plan for experiments. In the following section, the experimental results are discussed. Finally, the paper is summarized and conclusions are presented in the last section, where future work is also indicated.

2 Population decomposition in algorithms and implementations of the metaheuristics

Metaheuristics such as evolutionary algorithms (EA) are, as researchers confirms [19] [7], universal optimization methods. Such methods often suffer from problems with maintaining the diversity of the population [15], therefore one of common ideas to deal with this problem is decomposition of population and introduction of migration mechanism [9]. Such approach is believed to increase

[6] http://ray.io

the diversity and thus efficacy of the entire algorithm, as noticed in works [17] [7] on a parallel model of the evolutionary algorithm.

In many reports Island model of evolutionary algorithm performs excellently in comparisons with sequential development and usually ranks among the best when compared to many other solutions. This happens both for standard benchmark functions and for more complex problems [8], such as graph partitioning, set coverage, labor planning problem [2] and TSP [10].

In detailed analyses such as the paper by Skolicki and DeJong [15], the dynamics of the island model is described as a two-level structure. Two evolutionary processes interact, one at the local level and the other at a higher level between islands. These processes are complementary and may contribute to the overall result to varying degrees. It is believed that in the process of evolution in the island model, we first accumulate diversity between islands and then slowly release it by triggering migrations. It is important, of course, how we set its parameters, such as the migration interval [16] and the size of the migrant group, as well as the topology.

For research and application purposes, many platforms were constructed to support parallel, distributed, and HPC infrastructure in the context of running metaheuristics. For example, EASEA [7] is an Artificial Evolution platform (Free Open Source Software) that allows easy to implement evolutionary algorithms and to exploit the massive parallelism of many core architectures in order to optimize virtually any real-world problems, typically allowing for speedups up to x500 on 3,000 machines, depending on the complexity of the evaluation function.

PlatEMO [8] - open source MATLAB Platform for Evolutionary Multi-Objective Optimization, which includes numerous multi-objective evolutionary algorithms, test problems, and performance indicators. Enables one to properly benchmark existing algorithms and to apply selected algorithms to solve real-world problems and easily compare several evolutionary algorithms and collect statistical results.

AgE platform [9] is a Java-based solution developed as an open source project by the AGH-UST Intelligent Information Systems Group for the development and execution of agent-based applications in simulation and computational tasks. Its modular architecture allows one to use components to assembly and run agent-based computations for various problems such as black-box complex discrete problems (e.g. LABS, OGR, Job-shop) or continuous optimization problems [4].

jMetal stands for Metaheuristic Algorithms in Java. It is an object-oriented Java-based framework for multiobjective optimization with metaheuristics.

[7] `https://easea.unistra.fr/index.php/EASEA_platform`

[8] Source code of PlatEMO `http://bimk.ahu.edu.cn/index.php?s=/Index/Software/index.html`

[9] Project homepage: `https://gitlab.com/age-agh/age3`

DEAP[10] is an evolutionary algorithm framework with wide possibilities of creating user own types, customizing initializers, choosing operators, and implementing user algorithms that fit all he needs.

All these platforms are popular and proven in research; however, as we want to delve into the intricacies of HPC infrastructure, we utilize an implemented platform based on the actor model of concurrency, trying to reach efficient scalability of the whole system. In particular, our article examines the impact of delays occurring in various PEA topologies and is another attempt to answer the age-old question: how to obtain the appropriate PEA parameterization, topologies and migration strategies, keeping in mind the no-free-lunch theorem [20].

3 The algorithm and the computing framework used

The evolutionary algorithm processes a population of individuals (initially randomly selected). Each individual represents a potential solution to the problem, which is encoded in its genotype contained in its chromosome. The genotype allows to calculate the phenotype, i.e. the argument of the objective function. Individuals are compared using this function known as the fitness function [9]. The typical sequence of the algorithm is as follows: random initialization of the individuals, evaluation by applying the fitness function, selection of the parents, crossover and mutation, return to evaluation (while the stopping condition is false).

In a parallel version of EA the population is divided into several islands. These populations work simultaneously on separate islands, replacing a certain number of individuals at certain times. The result of PEA's work is the result of the best island.

The framework used is planned to support the parallel metaheuristics in as simple way as possible, i.e. introducing the software agents for realizing of the computing island, and introduce queues in order to communicate them in efficient way, avoiding any possible synchronization.

As we can see in Fig. 1, the core abstractions of the model are:

- Island - main actor that holds the Computation, neighbouring islands and new immigrants,
- Computation - the actor that creates and runs the algorithm,
- Migration - the actor that receives new immigrants,
- Emigration - the actor that selects the destination and sends the emigrants.

The created abstractions needed a proper programming framework to create an efficient implementation. We considered the most popular solutions, such as Akka (Scala) and Ray (Python). Since the evolutionary algorithms have already been developed using jMetalPy, we decided to use Ray. It allowed us to easily integrate the framework with the evolutionary algorithm. Ray provides distributed

[10] https://deap.readthedocs.io/en/master/

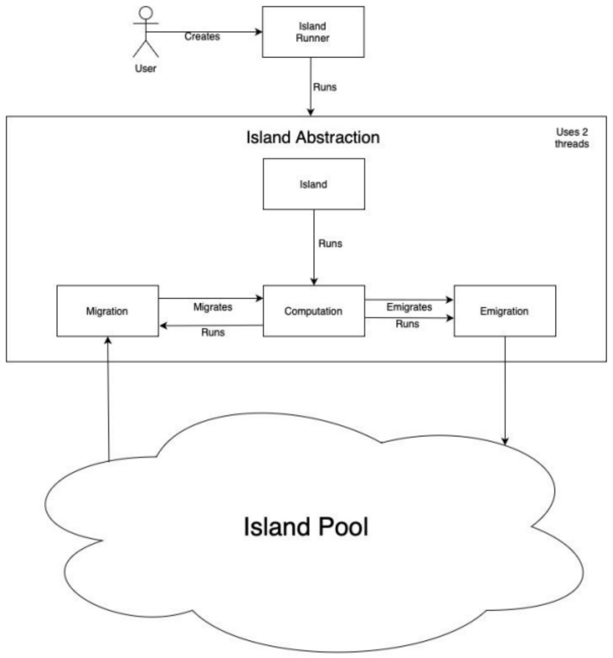


Fig. 1: Actor system architecture

abstractions (such as actors) through the Ray Core library. Asynchronous communication was achieved thanks to the pipelining pattern [11].

4 Planning the experiments

The main purpose of the experiments is to check whether the introduction of asynchronous communication affects the age difference (delay) between immigrants and the destination island. The parameters of the experiment are the following: number of islands = 50, 100, 150, 200, number of emigrants = 5, migration interval = 5, topology = torus, ring, complete. We used Rastrigin and Sphere problems (200 dimensions), 16 individuals in population and offspring population equal to 4.

The framework was deployed to the HPC cluster Ares located in the ACC Cyfronet in Krakow. The machine is equipped with 532 nodes of 48 core 2.90GHz Intel cpus. The number of cpu cores required for the efficient running of the framework doubled the number of islands.

The topology determines which islands migration takes place between. In Figure 2 three topologies used in the research were shown. We can describe them as follows:

[11] Pipelining pattern. https://docs.ray.io/en/latest/ray-core/patterns/pipelining.html

- *complete* - each island is connected to each other and can send/receive migrants to/from them,
- *torus* - islands are arranged in a matrix, islands at the right edge of the matrix connect to those on its left edge to form a cylinder, and then the islands at the bottom of the cylinder connect to those at the top - smaller number of connections between the islands is put together. Every island is connected with four other islands,
- *ring* - connections (ring-shaped) - each island sends migrants to only one neighbor and receives migrants only from one other neighbour - the smallest number of connections.

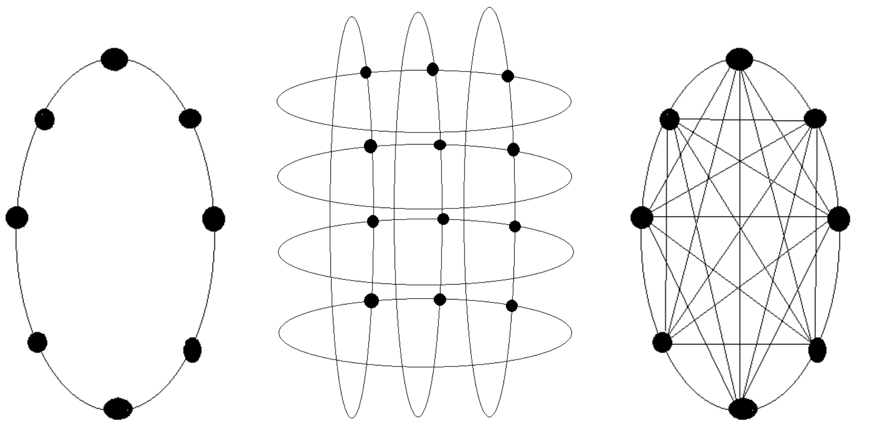

(a) Ring topology (b) Torus topology (c) Complete topology

Fig. 2: Topologies

To investigate how migration strategies affect delays, three migration strategies without repetition performed in PEA created in our previous work [3] were used. When we selected n migrants, in:

- *"best strategy"* - n individuals were selected in order of fitness values, starting from the best one,
- *"random strategy"* - n individuals were randomly selected,
- *"max distanse strategy (mDist)"* - Euclidean distances of all individuals were tested in pairs and then the farthest ones were selected in pairs (if possible) until the number of migrants set in the parameters was exhausted.

We have realized our research using two benchmark functions: Rastrigin and Sphere [11] using three topologies (complete, torus and ring) and three migrant selecting strategies ("best", "max distance" and "random").

5 Experimental study

During the experiments, we observed the occurring delays by comparing epoch on the source islands (islands from which migrants started) with the epoch on the destination islands (islands where migrants arrived). We were also interested in how the working time of one island behaves relative to the working time of each other island and how long is the common working time of all islands in the model. We also compared the final and average results obtained with specific topologies and migration strategies for both problems. Each algorithm setting was tested 10 times and the results were averaged at appropriate places and compared at other places to check the certainty and quality of the trends studied.

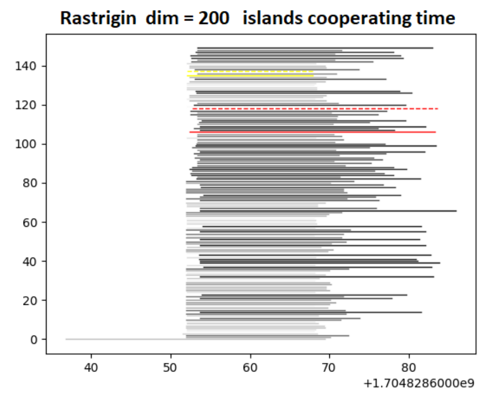


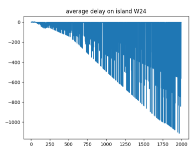
Fig. 3: Islands cooperation. Axis: Y - number = name of island, X - timestamp.

In Fig. 3 shows how all islands work in the PEA algorithm running on HPC. Each line shows the working time of some island. Darker color means islands with a better score. The yellow lines indicate the weakest work and the red lines indicate the best working islands. A dashed line (red or yellow) indicates "second in line" (in meaning: best or worst). The section between 53 and 68 is the time when all the islands are working. It is a time of cooperation when the exchange of migrants between all islands is possible, how long they cooperate with each other and what is the percentage of their solitary work (for example, in a ring topology - without a neighbor next door).

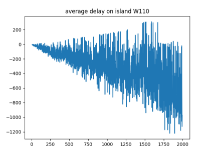
We have observed differences in how the yellow and red lines are arranged in relation to each other in the charts for individual topologies. For complete and torus topologies, yellow lines are short and red lines are long, like on Fig. 3. In the ring topology, when using the "mdist" and "random" strategies, the yellow line is shorter than or equal to the red one, while when using the ring topology with the "best" strategy, sometimes the red line is not the longest.

5.1 Delays - types

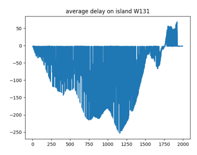
When running the genetic island algorithm with migrations in HPC, there are delays in the arrival of migrants. We define it as the difference between the generation on the starting island when they left and the generation on the destination island when they arrived. We observed them when we ran our algorithm, observing the operation of the islands and the process of arrival of migrants. They vary, as we can see in picture 4, depending on the topology and migration strategy, but they are also different on different islands in a single test (this is influenced, among other things, by how good the solution is generated by a given island).



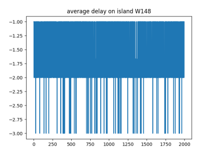
(a)



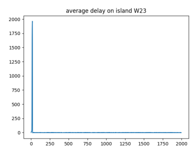
(b)



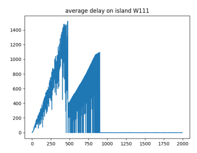
(c)



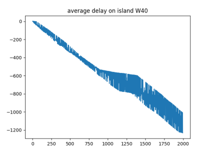
(d)



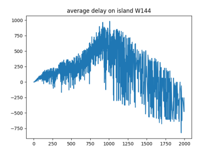
(e)



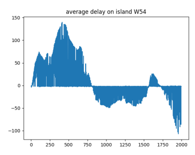
(f)



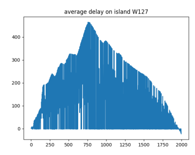
(g)



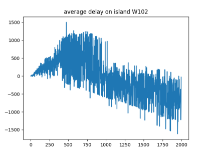
(h)



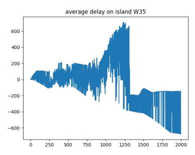
(i)



(j)



(k)



(l)

Fig. 4: Delay types

The delays observed in the tests took many characteristic shapes.
Let's define two concepts:

- *acceleration* - means that migrants arrive on the island from islands that have been evolving longer at the time of the migrants' departure than the destination island at the time of the migrants' arrival,
- *delay* - the opposite situation.

Let us now describe some of the most common patterns. Other shapes often are modifications or combinations of the following.

- *A* - triangles - the delay grows continuously, it is getting bigger,

- *B* - triangles - rounded at the top (positive values occur) - look like triangles A, only sometimes there are migrants arriving too early - but only up to a certain step,
- *C* - increasing delays most of the time, after a certain point improvement. Finally, the delays disappear and positive values appear,
- *D* - we have here constant small delays of two values,
- *E* - first, very time-delayed migrants arrived on the island. Then, migrants arrive without delay,
- *F* - very large, sudden acceleration for about half of the island's operation time, then there was no delays,
- *G* - slightly bold back-slash shape - all the time increasing delays,
- *H* - first, more and more modern individuals arrive, then less and less modern ones arrive, until individuals arrive that are more and more delayed,
- *I* - irregular sine wave. Alternately, delayed and accelerated migrants arrive on the island with varying intensity and period and amplitude,
- *J* - triangle lying on its side - this island was delayed at the very beginning and then there was a time point from which the delay started to decrease,
- *K, L* - H shape variant, slightly different shapes, initially accelerations, then increasing delays.

In different topologies, some of the latency patterns described above are more common and others are less frequent:

- Complete topology: This topology has by far the most delays. There is a very large communication crowd here, which is visible in the delay patterns characteristic for this topology.
  - Best strategy: The best performing islands are characterized by delays with H-type graphs (for the best ones - a narrower shape). The worst is definitely type A. Intermediate shape B shape and thick H shape.
  - MDist strategy: Here, the best islands are those that stop accepting migrants about halfway through their operation - for them, the delay pattern resembles Type F. The H-type appears in the middle of the ranking. Then, more worse, it is type B. The worst islands also have pattern A.
  - Random strategy: In complete topology with random strategy, islands working with delays from the thin H pattern win. The thicker H-shaped ones are slightly worse (1/3 of the rate). Then (3/4 of the rate) there are type B delays. And the worst have shape A.
- Torus topology: In this topology, each island has exactly four neighbors. Delays are more regular. Migrant crowds occur less frequently and are relatively small compared to the complete topology. However, sometimes there is an accumulation of arriving migrants and a delay occurs. Here is where the greatest variety of delay patterns occur. Therefore, this topology is quite a good compromise between a full graph and a ring topology, where there are very few neighbors.
  - Best strategy: The latencies for the top islands resemble Pattern J. The intermediate ones have different shapes - the better, the slimmer. And the worst islands have delay shape types A and G.

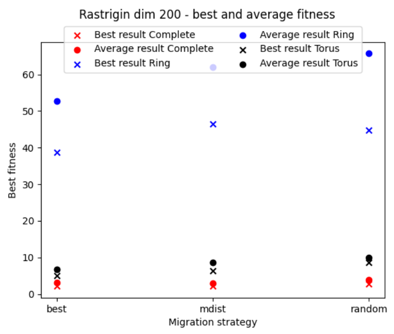
  - mDist strategy: The latencies for the top islands resemble Pattern J, but additionally there are negative values of a small quantity. Not many of the most worst islands have shapes of delays of type A, in the middle of ranking, various shapes begin quite quickly.
  - Random strategy: The winning islands have lag shapes of type F, a small percentage of the worst ones are of type A and G, and the intermediate shapes are very varied.
- Ring topology: In the ring topology, communication is the most orderly. The traffic congestion is the smallest, but if the neighbor of the island finishes work earlier, no migrant with fresh genetic material reaches the island from that moment on.
  - Best strategy: The latencies of the worst islands are represented by shapes A and D (with a constant amplitude from -3 to -1). Various shapes appear in the middle of ranking, most often I and F. The best is shape I or for half the time of the island's; operation shape I and J, and then a straight line like in shape F.
  - mDist strategy: About half of the islands have delays that end abruptly in the middle of the work section. The best islands have a sinusoidal shape, a triangle lying on its side or standing on top.
  - Random strategy: There is a great variety of shapes here. Many sinusoidal shapes. Characteristically, the patterns end abruptly when the neighboring island stops working and the flow of arriving migrants stops. The lags of the best islands have an upward convex parabola pattern, like the upper part of a sine wave ending halfway down the line.

Table 1: minimal and maximal values in delays averaged by topologies and migrant strategies

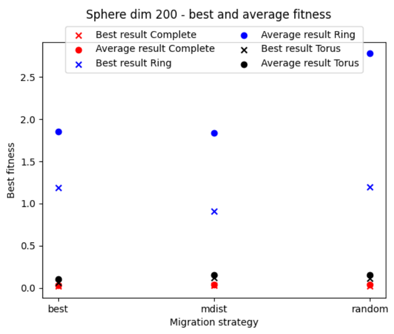
	average Min Delays	average Max Delays
complete	-1077,107481	672,6903784
ring	-316,9725226	319,8765899
torus	-607,6686456	565,9689137
complete best	-965,4746667	711,1034778
complete mdist	-1234,076	614,8229631
complete random	-1031,771778	692,1446942
torus best	-621,4404059	578,341527
torus mdist	-547,5165006	523,7945917
torus random	-654,0490302	595,7706223
ring best	-324,4927714	321,2856487
ring mdist	-315,4192413	308,517391
ring random	-311,0055551	329,82673

In the tab. 1 we show minimal and maximal values in delays averaged by topologies and migrant strategies.

5.2 Results achieved while using three topologies



(a) Rastrigin problem



(b) Sphere problem

Fig. 5: Different topologies results of Rastrigin and Sphere problem

As we can see in the Fig. 5 with the results for both problems:

The complete topology that wins generates large congestion and delays, but diverse migrants from multiple islands with different evolving populations clearly improve the results. The torus topology generating average results in this ranking has fewer connections between islands than in the complete topology and more than in the ring topology. Each island connects to four of its neighbors. Ring topology is the worst in the ranking. This is probably due to the large number of undelivered migrants when the destination island is no longer working, or a long time without arriving migrants when the source island is no longer working. When comparing the results, we also found that longer-acting islands have a better prognosis for a better result. And since in the maxdistance strategy the islands cooperate the longest (this common section is the longest in seconds), maybe it is a matter of the time needed to calculate this maxdistance?

5.3 Summary of the experimental results

As we show above, we can observe different types of latency when running PEA on HPC, depending on the topology.

There is a relationship between the topology of the islands' connections (in the sense of migration) and traffic congestion and the increased delays that occur when individuals migrate from island to island. Furthermore, when different migration strategies were used, a difference was observed in the types of delays that occurred or in the frequency of their different types. The topology of connections and the migration strategy are also related to the speed at which PEA achieves a good result.

The most dense traffic occurs in the complete topology, and yet its results compares better than the results of torus topology and much better than the results of ring topology. The ring topology, where number of connections between islands is the smallest (each island has only two neighbors, one of them receives migrants from and the other sends migrants to), while using asynchronous work of islands, is exposed to the fact that many islands work alone most of the time, while their neighbors finish their work before them. The torus topology is in some sense intermediate between complete and ring. Each island, having more connections with other islands, has a greater chance of long-term cooperation than in the ring topology and a smaller chance compared to the complete topology.

6 Conclusions

In this paper, we explored different patterns of the delays that occur in the island-based evolutionary algorithm running on the HPC infrastructure. Apparently, these phenomena can affect the computation using metaheuristics. Although this may not be so crucial as in the case of distributed simulation, we are convinced that by proper managing of such deyals we can increase the efficiency of the system (e.g. by dropping too delayed messages).

In the future, we will develop dedicated emigration/immigration operators and explore how they affect the algorithm. Such proposed operators will, e.g. neglect delayed migrants, or incorporate some partial information to the population carried by those.

Acknowledgement

The research presented in this paper received support from the Polish NCN Projects no. 2019/35/O/ST6/00571 (SB) and 2020/39/I/ST7/02285 (AB).

For the purpose of Open Access, the author has applied a CC-BY public copyright license to any Author Accepted Manuscript (AAM) version arising from this submission. This research received partial support from the funds assigned to AGH University of Science and Technology by the Polish Ministry of Education and Science (AB, SB).

We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2023/016415.

References

1. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation **6**(5), 443–462 (Oct 2002)
2. Alba, E., Luque, G., Luna, F.: Parallel Metaheuristics for Workforce Planning. Journal of Mathematical Modelling and Algorithms **6**(3), 509–528 (Sep 2007). https://doi.org/10.1007/s10852-007-9058-5
3. Biełaszek, S., Byrski, A.: On parameters of migration in pea computing. Progress in Polish artificial intelligence research 4, : Łódź University of Technology Press, 2023, nr 2437 p. 491–493 (2023)
4. Biełaszek, S., Piętak, K., Kisiel-Dorohinicki, M.: New Extensions of Reproduction Operators In solving LABS Problem Using EMAS Meta-Heuristic. In: Nguyen, N.T., Iliadis, L., Maglogiannis, I., Trawiński, B. (eds.) Computational Collective Intelligence. pp. 304–316. Lecture Notes in Computer Science, Springer International Publishing, Cham (2021)
5. Cantu-Paz, E.: On the Effects of Migration on the Fitness Distribution of Parallel Evolutionary Algorithms. No. UCRL-JC-138729 (Apr 2000), `https://www.osti.gov/biblio/791479`
6. Cantu-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA, USA (2000)
7. Cantú-Paz, E., Goldberg, D.E.: Efficient parallel genetic algorithms: theory and practice **186**, 221–238 (Jun 2000). https://doi.org/10.1016/S0045-7825(99)00385-0, iSSN: 00457825 Issue: 2-4 Journal Abbreviation: Computer Methods in Applied Mechanics and Engineering
8. Crainic, T.G., Hail, N.: Parallel Metaheuristics Applications. Parallel Metaheuristics p. 447 (2005), `https://www.academia.edu/65945140/Parallel_Metaheuristics_Applications`
9. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley (1989)
10. Lazarova, M., Borovska, P.: Comparison of parallel metaheuristics for solving the TSP. p. 17 (Jan 2008). https://doi.org/10.1145/1500879.1500899
11. Misaghi, M., Yaghoobi, M.: Improved invasive weed optimization algorithm (IWO) based on chaos theory for optimal design of PID controller. Journal of Computational Design and Engineering **6**(3), 284–295 (Jul 2019). https://doi.org/10.1016/j.jcde.2019.01.001
12. Nowostawski, M., Poli, R.: Parallel genetic algorithm taxonomy. In: 1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No.99TH8410). pp. 88–92 (Aug 1999)
13. Ruciński, M., Izzo, D., Biscani, F.: On the impact of the migration topology on the island model. In: Parallel Computing, Issues 10–11, October–November 2010, Pages 555-571. vol. 36 (1993)
14. Schaefer, R., Byrski, A., Smolka, M.: The island model as a markov dynamic system. Int. J. Appl. Math. Comput. Sci. **22**(4), 971–984 (2012). https://doi.org/10.2478/V10006-012-0072-Z, `https://doi.org/10.2478/v10006-012-0072-z`
15. Skolicki, Z., De Jong, K.: The importance of a two-level perspective for island model design. pp. 4623–4630 (Oct 2007). https://doi.org/10.1109/CEC.2007.4425078
16. Skolicki, Z., De Jong, K.: The influence of migration intervals on island models. pp. 1295–1302 (Jun 2005)

17. Sudholt, D.: Parallel Evolutionary Algorithms. In: Kacprzyk, J., Pedrycz, W. (eds.) Springer Handbook of Computational Intelligence, pp. 929–959. Springer Handbooks, Springer, Berlin, Heidelberg (2015)
18. Turek, W., Stypka, J., Krzywicki, D., Anielski, P., Pietak, K., Byrski, A., Kisiel-Dorohinicki, M.: Highly scalable erlang framework for agent-based metaheuristic computing. J. Comput. Sci. **17**, 234–248 (2016). https://doi.org/10.1016/J.JOCS.2016.03.003
19. Vose, M.D.: What are Genetic Algorithms? A Mathematical Prespective. In: Evolutionary Algorithms, pp. 251–276. The IMA Volumes in Mathematics and its Applications, Springer, New York, NY (1999)
20. Wolpert, D., Macready, W.: Macready, W.G.: No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation 1(1), 67-82 **1**, 67–82 (May 1997)