The impact of asynchrony on parallel model-based eas

Guijt, Arthur; Thierens, Dirk; Alderliesten, Tanja; Bosman, Peter A.N.

DOI
10.1145/3583131.3590406

Publication date
2023

Document Version
Final published version

Published in
GECCO 2023 - Proceedings of the 2023 Genetic and Evolutionary Computation Conference

The Impact of Asynchrony on Parallel Model-Based EAs

Arthur Guijt
Arthur.Guijt@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands

Dirk Thierens
D.Thierens@uu.nl
Utrecht University
Utrecht, The Netherlands

Tanja Alderliesten
T.Alderliesten@lumc.nl
Leiden University Medical Center
Leiden, The Netherlands

Peter A.N. Bosman
Peter.Bosman@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
Delft University of Technology
Delft, The Netherlands

ABSTRACT

In a parallel EA one can strictly adhere to the generational clock, and wait for all evaluations in a generation to be done. However, this idle time limits the throughput of the algorithm and wastes computational resources. Alternatively, an EA can be made asynchronous parallel. However, EAs using classic recombination and selection operators (GAs) are known to suffer from an evaluation time bias, which also influences the performance of the approach. Model-Based Evolutionary Algorithms (MBEAs) are more scalable than classic GAs by virtue of capturing the structure of a problem in a model. If this model is learned through linkage learning based on the population, the learned model may also capture biases. Thus, if an asynchronous parallel MBEA is also affected by an evaluation time bias, this could result in learned models to be less suited to solving the problem, reducing performance. Therefore, in this work, we study the impact and presence of evaluation time biases on MBEAs in an asynchronous parallelization setting, and compare this to the biases in GAs. We find that a modern MBEA, GOMEA, is unaffected by evaluation time biases, while the more classical MBEA, ECGA, is affected, much like GAs are.

CCS CONCEPTS

• **Mathematics of computing** → **Evolutionary algorithms**; • **Theory of computation** → **Parallel computing models**.

KEYWORDS

Genetic Algorithms, Model-Based Evolutionary Algorithms, Linkage Learning, Parallel Algorithms, Asynchronous Algorithms

ACM Reference Format:
Arthur Guijt, Dirk Thierens, Tanja Alderliesten, and Peter A.N. Bosman. 2023. The Impact of Asynchrony on Parallel Model-Based EAs. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3583131.3590406

GECCO '23, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0119-1/23/07.
https://doi.org/10.1145/3583131.3590406

1 INTRODUCTION

For the optimization of many real-world problems the assessment of the quality of a solution (evaluation) involves a time-consuming process, e.g., the training of a neural network. Problems with such evaluation functions are called expensive optimization problems.

More often than not, the amount of wall time spent should be minimized. Rather than using the resources to perform these evaluations sequentially, it is preferred to use the resources simultaneously, to run evaluations in parallel. As evaluations are also commonly independent, the parallelization potential is significant, allowing us to save significant amounts of time.

However, the generational nature of an EA may limit how parallelizable the algorithm is. EAs often follow a loop of generating offspring, evaluating them, and performing selection. Selection is only performed once all offspring solutions are evaluated, and new solutions are only generated and evaluated once selection has been performed. This describes a standard generational scheme. Alternatively, the population can be altered incrementally. For example, by generating a single solution at a time and attempting to introduce it into the population immediately after evaluation, resulting in a steady-state scheme.

In a parallel EA, unless very large population sizes are used, this limited number of solutions per generation means that processors may run out of solutions to evaluate until the next generation starts, i.e., when new offspring will be generated. Before continuing, these processors must wait on the other processors to ensure all evaluations have finished. This waiting on other processors is also called *synchronization*.

Synchronizing is however only required if one wishes to have the exact same behavior as the sequential implementation of the EA. Alternatively, one can also asynchronously sample, evaluate, and apply selection, as is the case for the asynchronous steady-state GA [15]. This may be beneficial as waiting wastes computational resources. For example, in a situation with many computing nodes, a slower node or evaluation may stop all other nodes from progressing, introducing a bottleneck into the optimization process, see Figure 1 for an illustrative example. Furthermore, in the case of node or network failures, synchronization will even lock up the optimization process indefinitely. In the remainder of this work we will refer to approaches employing synchronization as *synchronous* approaches, and approaches that forgo this as *asynchronous*.

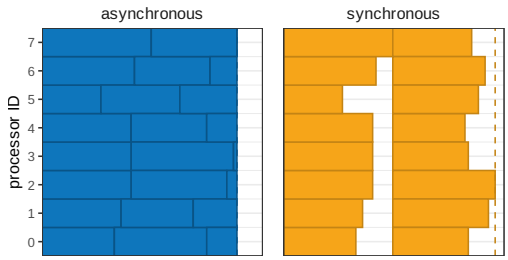

Figure 1: Resource consumption of synchronous and asynchronous approaches under heterogeneous evaluation times. Ideally, without synchronization the target solution can be found earlier (in this example: the most expensive solution of the second 'generation').

However, not synchronizing does not guarantee better overall performance of the EA. For example, while in [3] the asynchronous configuration outperformed the synchronous configuration, in [21] performance degraded when using an asynchronous approach. Evaluation time biases were investigated in [14–16], in which was shown that there exists an evaluation time bias, i.e., the distribution of the population is biased such that it is correlated with the corresponding evaluation times, such that this bias is not explained by fitness based selection. They note that this bias depends on both the distribution of evaluation times and the number of processors. More specifically, a preference towards both short and long evaluation times on a flat fitness landscape was observed.

Even so, it is difficult to determine based on current literature, when asynchronous execution of an EA is problematic. This is in part due to how comparisons are often performed. First, the selection procedure is often altered when switching from synchronous to asynchronous, making it impossible to distinguish effects caused by asynchrony from those caused by steady-state selection and variation. Selection and variation are key aspects of an EA, and should also be considered as an additional influence. Furthermore, in order for effects of time biases to be interpretable, the time distributions of evaluations need to be known as well.

More generally, for EAs the population size is important as well. Different approaches may require different population sizes to perform best, especially if variation and selection are different. When switching between synchronous and asynchronous the population size should therefore be tuned again to avoid giving preferential treatment to the approach for which the population size was tuned.

Given all of this, we are particularly interested in the impact of selection and variation on the behavior of the EA. Together they induce a bias towards higher fitness solutions in the population. An oversight in how these operators work could very well induce, preserve, or halt evaluation time biases too.

In this work, we will explicitly also consider Model-based Evolutionary Algorithms (MBEAs). Through the use of linkage learning (LL) in MBEAs, variation can be performed based on inferred variable dependencies. This can result in significant performance improvements. To our knowledge, no prior work has studied the impact of asynchronous parallelization on MBEAs. Yet this is of interest, LL infers the structure of a problem through the use of the population. If the composition of the population is based not only on the fitness, but also the evaluation time associated with these solutions, then this will also affect the structure learning process. Therefore, while LL is known to improve performance, this could be disrupted by biases, such as evaluation time biases.

Our research questions are therefore:

1. How does selection affect performance, the ability to find a solution with target fitness, under various evaluation time distributions in an asynchronous setting?
2. How does variation affect performance under various evaluation time distributions in an asynchronous setting?
3. More specifically, how are MBEAs like GOMEA and ECGA affected by the evaluation time distribution when made (a)synchronous parallel?

The remainder of this work is structured as follows. First, in Section 2 we will describe the EAs used in this work. Following that, in Section 3, the artificial benchmark functions and the evaluation time distributions used, in addition to a real-world NAS benchmark are described. The remainder of experimental considerations is described in Section 4. We discuss the results in Section 5 and conclude in Section 6.

2 APPROACHES

In this work, we include a Simple GA as described in Subsection 2.1. To study how linkage learning (LL) and evaluation time biases interact, we use both a classic MBEA named the Extended Compact Genetic Algorithm (ECGA) and a modern MBEA named the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) as described in Subsections 2.2 and 2.3 respectively.

2.1 Genetic Algorithm (GA)

In previous works, selection in GAs is often altered simultaneously with the (a)synchronous nature of the approach. For example, in [15], the synchronous configuration uses a generational selection scheme, whereas the asynchronous configuration is steady-state. A steady-state configuration exhibits different behavior compared to GAs employing a generational selection scheme [17]. We therefore also investigate a 'synchronous' steady-state variant and an asynchronous 'generational' approach in addition to the original two configurations. Pseudocode for these approaches can be found in the supplementary material.

For the synchronous steady-state approach we operate in batches of $|P|$ offspring, generating each offspring at the start of the evaluation, and synchronizing until all $|P|$ offspring solutions are sampled and evaluated. Steady-state selection is then performed once the evaluation of an offspring solution is finished. For this we opt to randomly select a solution from the population and replace this solution if it is worse than the newly evaluated offspring solution. This ensures that the population's average fitness cannot decrease, and thus stops any bias contrary of improvement to fitness from taking over the population. When using generational selection in an asynchronous setting, every solution that completes evaluation is added into a selection pool first, similar to the approach described in [16]. Once this pool has reached the prerequisite size, we apply the generational selection operator, replacing the entire population

with the end result. We perform generational selection using a Parent + Offspring (P+O) tournament of size 4, where the number of offspring is equal to the population size. Tournaments are created based on shuffling, then by splitting the population in blocks of the tournament size, repeating as often as necessary to select P solutions.

For recombination, we use Uniform Crossover (UX) and Two-point Crossover (TPX). In addition to Subfunction Crossover (SFX) for problems for which subfunction information is available. While UX is included as a baseline, TPX and SFX are included as they suit the structure of at least one of the artificial benchmark functions described in Subsection 3.1. Specifically, in SFX each block of variables that forms a subfunction is exchanged with $p = 0.5$, perfectly mixing the blocks of the concatenated DT function, whereas TPX is especially suited for the ANKL problem due to its sequential adjacent structure. These operators should showcase how the behavior changes when a well-suited operator is used from the start and throughout the search, also providing an idealized reference for approaches employing LL.

2.2 Extended Compact Genetic Algorithm (ECGA)

ECGA is one of the first approaches employing LL. Unlike GOMEA, explained in the next subsection, this approach still has separate recombination / variation and selection steps. This allows us to use exactly the same selection procedures as for the GA.

With ECGA, a marginal product model (MPM) is learned using a metric based on model complexity and population compression after applying selection [9]. For this selection step, we apply tournament selection of size 4. In the resulting model every variable is grouped disjoint subsets, thereby modeling each subset of variables jointly. For example, given MPM $\mathcal{M} = \{\{0, 1\}, \{2, 3\}\}$ and population (after selection) $P_M = 0011, 1100$, allows us to sample 00 and 11 with, for each subset of variables, equal likelihood. Therefore, allowing us to sample 0000, 0011, 1100, and 1111.

Learning a model is costly. Therefore, performing continuous updates of this model for every solution sampled is too computationally expensive to be benchmarked properly. As such, for the asynchronous configuration the model is updated only when $|P|$ (population size) evaluations finish. Thereby updating the model with the same frequency as the generational approach (generationally). While this reduced update frequency should not have too significant an impact on the learning of the MPM, solutions are sampled using out-of-date frequency estimates.

2.3 Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA)

While ECGA utilizes LL, it does so differently from most modern MBEAs. Modern model-based approaches like P3 [7, 8], DSMGA-II [1, 11], and GOMEA [5, 19] all use an incremental change and accept/deny mechanism, reminiscent of local search. This allows these approaches to utilize non-disjoint models, like the Incremental Linkage Set (ILS) in DSMGA and the Linkage Tree (LT) in GOMEA [19] and P3 [7]. In this work we will be using the LT, which is constructed through UPGMA hierarchical clustering applied to normalized mutual information (NMI) of the variables in the population. Each of these models are often represented as a Family of Subsets (FOS), i.e., a list of subsets containing variables to which variation should be applied jointly. For the LT the resulting tree is flattened such that each node in the tree corresponds to a subset of variables.

Variation and selection are performed using Gene-pool Optimal Mixing (GOM). In GOM changes are made to subsets of variables as defined by a Family of Subsets, sampled from the population. After each change solutions immediately compete against their parent. Because of this, changes are evaluated more directly, preventing changes to other variables from being a source of noise for assessing the quality of the current change [5]. If no change was made to a solution through all steps in GOM, or the strict non-improvement stretch of a solution $(1 + \lfloor \log_2(|P|) \rfloor)$ was reached, Forced Improvements (FI) are applied. FI is GOM where the donor is the current best solution (elitist). If FI fails to improve a solution, the solution is replaced with the current elitist. While this integrated variation and selection operator prevents us from changing the selection operator, this combined recombination and selection process is interesting in itself.

2.3.1 Synchronous. The synchronous approach closely follows the sequential version of GOMEA. Every generation starts with learning a linkage model. Then, an offspring population is made, initially containing a copy of each individual in the population. GOM and potentially FI are then scheduled to be applied in parallel on each of these offspring solutions, leaving the population from which is sampled during GOM, unchanged. When processors are available, and there are still offspring solutions left on which GOM needs to be applied, GOM is applied to of the offspring solutions in parallel. Each application of GOM is scheduled continuously until all involved evaluations have completed. Once GOM (and FI) complete the processor is freed up again. A generation ends once all solutions had GOM applied to it once.

2.3.2 Asynchronous. When making GOMEA asynchronous, there is no longer a generational offspring population which is generationally improved. Instead, GOM is scheduled to be applied after initialization, and after completing GOM, using a queue. Once a processor has finished its current task, the next task from the queue gets executed. At the start of (asynchronous) GOM, if GOM has been applied $|P|$ (population size) times, a new FOS is learned. Furthermore, a copy of the population is made. This effectively turns every application of GOM into its own mini-generation. Consequently, at the end of GOM we copy the generated offspring to the population from which is sampled. We label this configuration "a/e", for *asynchronous end*. However, this configuration leads to significant use of out-of-date information: none of the accepted changes of solutions undergoing GOM are visible to other solutions. As such we also evaluate an alternative configuration that copies the offspring to the shared population at *intermediate* stages during GOM, not just at the end. This configuration is labelled "a/i".

3 PROBLEMS

Previous work has indicated that heritable heterogeneous evaluation times can negatively influence the behavior of an asynchronous EA [15]. Heritable heterogeneous evaluation times concern variations in evaluation time associated with the genotype itself, rather

than external factors like machine load. A good example of a problem with this property is training a neural network, which we will discuss in Subsection 3.2.

However, both the fitness landscape and evaluation times of such a problem are generally complex, which may complicate analysis. We will therefore first describe benchmark functions with varying kinds of landscapes and structure, for which we will also vary the evaluation times associated with the solutions, in Subsection 3.1.

3.1 Artificial Benchmark Functions

In [15] it is indicated that evaluation times need to be heritable, and as such we will focus on this aspect. Similarly, evaluation times need to be preserved under variation too, as otherwise no time bias can develop. Furthermore, in [15] settings in which the evaluation cost was perfectly positively and negatively correlated were investigated - and found no significant difference in performance. As any bias contrary to the improvement of fitness would be unlikely to survive selection, performance of an EA is likely not impacted. We therefore select a distribution based on the genotype which is not a simple function of the fitness.

Furthermore, as the results in [14] indicate a bias towards the extreme evaluation times, we will control where in our benchmark functions these extremes are located. The evaluation time setting in this work is expressed as a ratio $a : b$, where b is the cost of the optimum s^* and a is the cost of the bitwise complement $\bar{s^*}$ - i.e., the solution with all bits flipped. The evaluation time $E(s)$ of a solution s, given the normalized Hamming distance $H(s,\ s^*)$ between this solution s and the optimum s^* is then:

$$E(s) = H\left(s,\ s^*\right)\ a + \left(1 - H\left(s, s^*\right)\right)\ b \tag{1}$$

We choose to use the ratios 100 : 1, 10 : 1, 2 : 1, 1 : 1, 1 : 2, 1 : 10 and 1 : 100, that is, ranging from a cheaper optimum to a more expensive optimum.

3.1.1 Concatenated Deceptive Trap (DT). The first problem we will use is the concatenated DT function [4, 20]. It consists of n blocks of size k which are concatenated together, forming a full string of length $\ell = nk$. In this work, we will only consider the case for $k = 5$. In order to evaluate the function, first the number of '1' bits within the block b is counted. Following that, the DT function is applied to the unitation of each block 0 through $n - 1$ and aggregated by summation:

$$DT(u) = \begin{cases} k & u = k \\ k - u - 1 & \text{otherwise} \end{cases} \tag{2}$$

$$f(x) = \sum_{b=0}^{n-1} DT\left(\sum_{i=bk}^{bk+k-1} x_i\right) \tag{3}$$

As there are many search paths indicating that more zeroes is better, high-fitness solutions consisting of zeroes are easiest to find. The solution consisting of all zeroes is referred to as the deceptive attractor. Yet, the needle in a haystack where $u = k$, has better fitness: k as opposed to $k - 1$ for the deceptive attractor. It is therefore all ones that is actually the optimum to this function. In order to solve this problem in a scalable manner, variables should be exchanged at the level of these blocks [18], for example by recognizing block structure by using linkage learning.

The complement of the optimum to this problem consists of all zeroes, and is the solution consisting of only attractors. We have defined the range of evaluation times depending on these two solutions. A preference towards cheaper solutions could therefore make a cheap-to-evaluate attractor even more attractive.

3.1.2 Adjacent NK-Landscapes (ANKL). While the (additively decomposable) concatenated DT function is difficult to solve with a local searcher, it is separable. This makes the problem easy to solve if the separability is known, or when this separability can be inferred. We therefore also consider the non-separable problem of ANKL. This problem consists of overlapping blocks consisting of k adjacent variables with some stride s from block to block. In this work, we consider $k = 5$ and $s = 2$. For each block i a randomly generated function f_i is defined that maps each genotype of this block to a value ranging from $[0, 1]$. Given random functions f_0 through f_{n-1}, where x is a genotype of length $\ell = sn + k - 1$:

$$f(x) = \sum_{i=0}^{n-1} f_i\left(x_{si}, \ldots,\ x_{si+k-2},\ x_{si+k-1}\right) \tag{4}$$

Due to this more complex overlapping structure as well as the random nature of the subfunctions, it will be more difficult for the linkage learning approaches to configure the linkage model appropriately. As a result, such approaches require more generations to obtain a suitable model. This may therefore provide more time for any evaluation-time biases to steer the population towards or away from the optimum, potentially causing structure to be found earlier, or preventing the structure from being found at all.

3.2 NASBench 301

While benchmark functions are interesting and useful for analysis, they are not necessarily representative of practical problems. Real-world problems often contain elements that simpler benchmark functions do not account for.

We will therefore apply the aforementioned approaches to the Neural Architecture Search benchmark NASBench 301 [22]. NASBench 301 is a benchmark for neural architecture search applied to the DARTS search space. In this search space, each network is trained from scratch. In order to make this less expensive as a benchmark, the benchmark provides both a surrogate model for the fitness and the corresponding evaluation time. This allows for an efficient simulation of a run on this problem without requiring a GPU for hours.

We use version 0.9 of the XGB surrogate model for performance and LGB model for runtime. As noisy objective functions are not a subject of research for this work, we have disabled noise for the performance model. For this experiment only the runtime model is used as an evaluation time distribution. The runtime model does not contain noise.

4 EXPERIMENTAL SETUP

For all problems, a run using a specific population size is stopped if it has reached the target value, or if it has converged, i.e., when all genotypes in the population are identical. As there will be configurations and problem pairs for which the target value is not reached within a reasonable amount of time, for each set of problems, we

will also be limiting the amount of time spent on a full bisection run. If bisection was terminated prematurely we will select the smallest successful population size found by bisection within the time limit.

Statistical tests are performed using the Mann-Whitney U-test [6, 13] with $p = 0.05$ with Holm-Bonferroni [10] correction where applicable. All experiments are carried out on a machine with two AMD EPYC 7282 16-Core Processors 2.8GHz, with 252 GB of RAM. Source code can be found at https://github.com/8uurg/Impact-of-Asynchrony-on-MBEAs

4.1 Artificial Benchmark Functions

Prior work [14–16] investigated the distribution of evaluation times. However, if an approach is influenced such that the distribution of evaluation times is biased, this does necessarily indicate a negative impact on the performance of an approach, like the time required to reach the optimum. We will focus on performance under various evaluation time configurations.

However, using standard performance metrics is problematic. When using the number of evaluations required to reach a target fitness, synchronous approaches are favored as waiting does not incur any penalty, whereas utilizing these resources with additional evaluations does count as additional evaluations. For our first experiment, we will be changing the distribution of evaluation times, and compare all investigated distributions. Using the wall time in this comparison is problematic as the total wall time spent will change with the distribution. For example, scaling all evaluation times by 10 will not change the evaluation times with respect to one another. As such a run will proceed identically, except with evaluation times that are 10 times larger. For more complex distributions it would be hard to say whether the approach is actually negatively affected by the change in evaluation times, or whether the solutions to be evaluated are more costly.

We instead choose a measure that stays constant if behavior is the same, yet still indicates when performance worsens. For this we use the minimally required population size to reach a target value as determined by bisection, and compare how the minimally required population size scales across varying evaluation time distributions.

This measure works for a convergent EA as in this case the population acts as a buffer against diversity loss. As the minimally required population size is the smallest population size for which a problem is solved, an increase indicates the need for a larger diversity buffer, in turn indicating the loss of important diversity compared to the previous configuration. If the same approach is used, and the only difference between two configurations is the evaluation time distribution, an increase in the minimally required population size is an indication that the approach is sensitive to a harmful evaluation-time bias.

We will perform these first experiments with the number of processors equal to $|P|$ (the population size) as this maximizes the degree of parallelization proportional to the number of evaluations performed. Additionally, given this setting, the evaluations with a fast evaluation time will also be the first to complete, which is not guaranteed with low degrees of parallelization.

For these experiments, each configuration will be evaluated for 100 random seeds, with each bisection run limited to 1 hour. This is orders of magnitude more than what most approaches need, and ensures even the worst performing algorithms have a chance.

4.2 NASBench 301

While comparing how approaches scale across different time distributions will allow us to study the impact of evaluation time biases, a practical problem often only has a single evaluation time distribution associated with it. Furthermore, the amount of computational hardware available is not unbounded in practice. If this were not the case, one could evaluate the entire search space in parallel at the cost of the most expensive solution. A more practical goal for parallelization for a specific problem is to minimize the amount of wall time spent to hit a target fitness given a limited amount of resources. As we only regard a single evaluation time distribution, aforementioned concerns do not apply to the NASBench 301 experiment.

Therefore, for the NASBench 301 experiment we will run experiments for the simulated wall time required to reach a target value. Here, first a range bounding this minimal evaluation time is determined, followed by a modified golden section search [12], detailed further in the supplementary material. This is to find the population size that minimizes the corresponding wall time to ensure that each approach can be compared fairly.

For this experiment the number of processors is restricted to 64. Each configuration is evaluated for 20 different random seeds for at most 16 hours, due to the more costly nature of using the surrogate over a benchmark function. As in preliminary experiments this time limit was found to be insufficient due to wasting a significant amount of time on small population sizes, we additionally require an improvement to be found every $2e + 10P$ evaluations, where e is the number of evaluations issued at the last improvement and P is the population size.

5 RESULTS AND DISCUSSION

First, we will discuss the results on the artificial benchmark functions (Table 1 for DT and Table 2 for ANKL). After this, we will discuss the results on NASBench 301.

5.1 Artificial Benchmark Functions

From Figure 2 it is apparent that asynchronous configurations experience an evaluation time bias that leads to a change in behavior. Specifically, the required population size is lower when the optimum is cheaper, i.e., is faster to evaluate, and larger when the optimum is more expensive, i.e., takes longer to evaluate. Simultaneously, synchronous approaches are invariant to the distribution of evaluation times investigated. This is in line with what would be expected based on the results in literature [14–16].

However, the extent of the differences in minimally required population size is highly dependent on the crossover used. When the most suitable crossover is used, i.e., SFX for DT and TPX for ANKL, the differences between the timing settings are small. At the same time, when an unsuitable crossover is used, differences in required population size range *across orders of magnitude*. In the worst case, the problem is not solved within the allotted time for more expensive optima, as such evaluation time biases can negatively impact the performance of an approach.

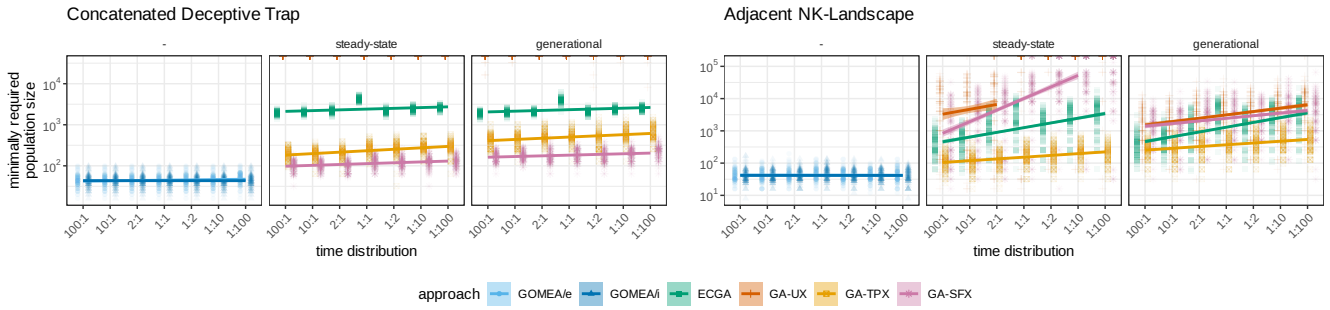

Figure 2: Correlation between time distribution and minimally required population size, with to the left the optimum being cheaper and to the right the optimum being more expensive than its complement. All runs are plotted as a point with opacity. Only asynchronous configurations are shown. For the creation of the regression lines we assume that failed runs used the maximum population size tested and are not drawn if more than half of the runs did not find the optimum. (Left: Concatenated DT $l = 50$, $k = 5$, Right: ANKL $l = 40$, $s = 2$, $k = 5$)

Table 1: Median minimally required population size on DT for $\ell = 50$, $k = 5$. Extended table in supplementary material.

selection	-			steady-state								generational							
approach	GOMEA			ECGA		GA						ECGA		GA					
cx	LL-LT			LL-MPM		UX		TPX		SFX		LL-MPM		UX		TPX		SFX	
(a)sync	a/i	a/e	s	a	s	a	s	a	s	a	s	a	s	a	s	a	s	a	s
timing																			
100:1	44	44	44	1944	4216	-	-	192	262	102	132	1906	4046	-	-	434	546	176	190
10:1	44	44	44	2038	4216	-	-	196	262	102	132	2022	4046	-	-	450	546	174	190
2:1	44	44	44	2166	4216	-	-	208	262	104	132	2042	4046	-	-	512	546	164	190
1:1	44	44	44	4110	4110	-	-	264	264	130	130	4052	4052	-	-	496	496	182	182
1:2	44	44	44	2110	4118	-	-	256	244	120	122	2060	4050	-	-	482	508	168	188
1:10	44	44	44	2394	4118	-	-	286	244	132	122	2316	4050	-	-	584	508	198	188
1:100	44	52	44	2562	4118	-	-	288	244	128	122	2534	4050	-	-	634	508	234	188

Table 2: Median minimally required population size for ANKL for $\ell = 40$, $s = 2$, $k = 5$. Extended table in supplementary material.

selection	-			steady-state								generational							
approach	GOMEA			ECGA		GA						ECGA		GA					
cx	LL-LT			LL-MPM		UX		TPX		SFX		LL-MPM		UX		TPX		SFX	
(a)sync	a/i	a/e	s	a	s	a	s	a	s	a	s	a	s	a	s	a	s	a	s
timing																			
100:1	40	40	44	454	4004	7034	16260	120	206	968	11516	382	3704	1968	3682	268	408	1578	2990
10:1	36	40	48	506	4004	8156	16260	114	206	3602	11516	512	3704	1982	3682	254	408	1752	2990
2:1	40	40	44	1426	4004	16380	16260	124	206	7160	11516	1214	3704	3468	3682	396	408	3634	2990
1:1	48	48	44	2992	2992	-	17098	178	178	23488	9540	4012	4012	4076	3962	386	386	3442	3058
1:2	40	42	48	1784	3760	-	15648	194	182	49152	10848	1756	2848	4066	3630	444	444	2872	3310
1:10	40	40	48	2924	3760	-	15648	228	182	57344	10848	2802	2848	6260	3630	516	444	5046	3310
1:100	40	40	48	3722	3760	-	15648	218	182	-	10848	3654	2848	8148	3630	512	444	4096	3310

Table 3: Median of minimally required amount of time to find a solution with an accuracy of 95.2727 or higher on NASBench 301 and corresponding median population size. Sample count is even, median is midpoint average.

selection	GOM			steady-state						generational					
approach	GOMEA			ECGA		GA				ECGA		GA			
cx	LL-LT			LL-MPM		UX		TPX		-		UX		TPX	
(a)sync	a/i	a/e	s	a	s	a	s	a	s	a	s	a	s	a	s
population size	63.5	49.5	58.5	-	-	252.5	368.5	1024	1110	3072	-	816	480.5	2048	2048
simulation time (s) × 10^6	1.36	1.68	1.65	-	-	0.98	1.40	4.26	4.49	7.73	-	1.52	0.98	4.43	4.77

The selection method can also impact how the approach scales across different evaluation time settings. For example, on ANKL the asynchronous steady-state GAs with UX and SFX degrade substantially, even to the point of failure, whereas the same GA, but with a (pseudo-)generational scheme degrades substantially less, by at most a factor 5. Yet, with the right crossover, steady-state actually has a smaller minimally required population size, and degradation between both configurations is at most a factor of 2.

We conjecture that this is caused by the following. In a steady-state approach the solution is immediately integrated into the population. Additionally, it is also immediately available to be used by crossover to generate new offspring. These offspring are now more likely to be fast evaluating solutions, depending on how well variation will preserve the evaluation time of a solution. This effect can accumulate over time, leading to premature convergence.

On the other hand, in a generational scheme these solutions are not immediately integrated into the population. Newly generated offspring are hence distributed according to the original distribution of solutions – with less evaluation time bias affecting them. Simultaneously, solutions with longer evaluation times will take longer for a processor to evaluate. During this time, no other solutions will be sampled for evaluation on this processor. In effect, the pool of currently evaluating solutions performs selection against short evaluation times. This will cause the amount of resources allotted to fast evaluating solutions to be lower. Finally, this results in less accumulation of evaluation time bias towards fast evaluating solutions, avoiding premature convergence towards such solutions.

Variation also has a notable impact on the impact of evaluation times. When variation aligns with the problem's structure, the different selection schemes unexpectedly scale similarly across evaluation time settings. Picking the right crossover operator will help with avoiding significant degradation for asynchronous approaches, even if the approach is in a steady-state configuration.

These results are promising for MBEAs because in MBEAs problem structure is automatically learned to inform its variation operator. One could therefore expect performance to always be reasonable, resulting in the precise selection scheme mattering less.

Before we discuss the results for ECGA, we repeat that ECGA's steady-state configuration is not truly steady-state, but only applies steady-state like selection. This is because ECGA only updates its model once $|P|$ (population size) solutions finish evaluating, as stated in Section 2.2. This results in an approach with behavior more closely resembling that of the generational approach. The differences between the two selection methods is negligible if the distribution updates less frequently, as is the case for ECGA.

First of all, there is the oddity that the constant time distribution requires a larger population size in order to find the optimum than the other asynchronous configurations for ECGA. The required population size is more in line with the synchronous configuration than the other time distributions. We explain this as follows. In an asynchronous setting with heterogeneous evaluation times, after the first few solutions finish evaluation, not enough solutions have finished evaluation yet to update the model. As such, the next solutions to be evaluated are still sampled from the initial random model. These solutions are therefore additional random solutions. Conversely, in the case where the evaluation time distribution is constant, all evaluations finish at the same time. This results in the model being updated. Therefore, the next solutions to be evaluated on the freed up processors, will be sampled using an *updated* model, potentially with some (evaluation time) bias.

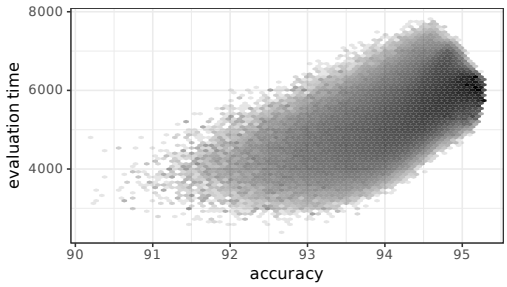

Figure 3: Objective (accuracy of the trained network) versus evaluation time sampled by runs of the approaches for NAS-Bench 301. Showing the experienced correlation between the two for the approaches.

After a model update, only the solutions that are currently being evaluated can originate from an older model. There are at most "number of processors" such solutions. In the case of the experiments above, this would be equal to the population size. This is reflected in the results: the minimally required population size is approximately twice as big, only for the constant evaluation time. The set of solutions that are currently evaluating may therefore act like an extension of the population itself.

When observing how ECGA's minimally required population size scales for different evaluation time settings in Figure 2, note that it scales worse compared to a GA with a suitable crossover for ANKL, and even with a non-suitable crossover with generational selection. Variation is more than the subsets of variables that are captured in the MPM model. How the model is used is just as important. In this case, the global sampling for each subset seems to be worse than the recombinative crossover of a GA. In conclusion, though not necessarily the fault of LL, LL within an MBEA is no guarantee that variation will perform well.

In contrast to ECGA and the GAs, GOMEA is found to be invariant to the evaluation time setting in most cases. There exists only a single time setting and problem combination for GOMEA for which the medians are statistically significantly different from the rest: 'a/e' on DT for 1:100, see Table 1. This is likely due to the combined variation and selection method used in GOMEA. As changes are made to individuals, they only compete against their parent. When parent and offspring are similar in evaluation time, this automatically results in niching behavior with respect to the evaluation times of solutions. In comparison to global selection, this approach stops solutions that evaluate quicker from taking over the population, in effect removing a large source of evaluation time bias.

5.2 NASBench 301

For NASBench 301 the distribution of the objective and evaluation time of a solution is shown in Figure 3. From the positive correlation and observations on the benchmarks one would expect to see asynchronous configurations to be outperformed by their corresponding synchronous approaches. Yet, if one refers to Table 3,

this is not the case. As a matter of fact, some of the best performing approaches are asynchronous. We explain this as follows.

Referring to Figure 3, one may note that the correlation is not perfect. The target solution for this problem is not actually the most expensive or least expensive solution, as was the case for the artificial benchmark functions. Additionally, an EA does not utilize all of these solutions simultaneously. If we look at the correlation from a particular fitness value onwards, i.e., truncating the population, the correlation decreases as this fitness threshold increases. Eventually, this correlation even becomes slightly negative for these data points: a setting which would actually be considered beneficial for steady-state asynchronous GAs based on our previous results.

For ECGA this is particularly notable. The approach itself is known to have trouble working with certain multi-modal functions [2], and it seems NASBench 301 is among these problems. Even so, the asynchronous approach has runs in which a solution with at least the target fitness was found, whereas the synchronous approach does not. This could be due to evaluation time biases helping the approach, much like what happens with the asynchronous steady-state GA for ANKL. It is therefore plausible that evaluation time biases may in part be responsible for the improvements in performance observed for asynchronous approaches, rather than only the improvements in throughput.

Furthermore, we have observed ECGA to prematurely merge FOS elements together on NASBench 301. As the model is unlikely to merge variables if they are not correlated, such a merge seems to only further reinforce correlation for this problem. Combined with the high selection pressure, this is likely to result in premature convergence to a single mode. In contrast, the linkage tree used in GOMEA does not suffer from this issue. The LT FOS always includes the univariate FOS elements: subsets with each variable on their own. Combined with the niching behavior described above, this significantly reduces the possibility of premature convergence no matter the source.

This further reinforces that an MBEA not only has to learn the right linkage from the population, but also use it in the right manner.

For GOMEA we would expect a difference between the time required for the synchronous and asynchronous approach. Since with GOM many similar variation steps are done to a single solution sequentially, the variance in evaluation times is potentially amplified. Furthermore, population sizes are relatively small compared to the GA and ECGA. We would therefore expect the throughput for asynchronous GOMEA to be considerably higher, and as such the time required to be lower.

However, no statistically significant difference in time between the *a*synchronous/e and *s*ynchronous approach is observed ($p = 0.52$, $U = 224$, 20 samples). Furthermore, a more detailed investigation of a single run does indicate that the amount of time spent idle for the synchronous approach is notable, on average 140059s per processor over the entire run. In contrast, there is a statistically significant difference for the time required between the asynchronous/i and synchronous approach ($p = 0.007 < 0.05$, $U = 100$, 20 samples). Effectively, when using outdated parents, offspring are more likely to have a lower fitness value. This degrades the performance of the approach, counteracting any gains in throughput.

For an asynchronous MBEA to actually gain performance compared to its synchronous counterpart, model updates should not be too infrequent and material with which is recombined should be recent. GOMEA could still be improved in this regard: each run of GOM keeps a copy of the population for sampling. As this population was created at the start of GOM, prior to the first evaluation, this population will gradually become outdated. Updating this population during GOM could result in recombination with more up-to-date solutions, potentially further improving performance.

6 CONCLUSIONS

Answering RQ 1, we have observed that steady-state asynchronous EAs are much more vulnerable to the biases induced by heterogeneous evaluation times, compared to asynchronous EAs using a generational scheme. Furthermore, answering RQ 2, when paired with a variation operator that is not competent in terms of improving fitness, the impact on an algorithm's capability to solve a problem can be severe. In contrast, when using well suited variation and selection operators, any differences between synchronous and asynchronous configurations become much smaller.

For the first time we have investigated the impact of heterogeneous evaluation times on parallel MBEAs and linkage learning (LL). The addition of LL promises to automatically align a variation operator with the structure of a problem. However, answering RQ 3, there are significant differences in the impact of evaluation times on the performance of ECGA, and GOMEA. While LL is not affected to the extent that evaluation time biases prevent the approach from finding high quality solutions, its performance greatly depends on how variation and selection are performed.

Finally, rather than negatively impacting the results, LL, when paired with the right variation and selection scheme can be a useful tool for obtaining good performance in general, even when an EA is asynchronous. GOMEA is an example of such an EA. GOMEA gets selection and variation right for all the problems evaluated, and is invariant to the choice between a synchronous or asynchronous configuration.

ACKNOWLEDGMENTS

This publication is part of the project "DAEDALUS - Distributed and Automated Evolutionary Deep Architecture Learning with Unprecedented Scalability" with project number 18373 of the research programme Open Technology Programme which is (partly) financed by the Dutch Research Council (NWO). Other financial contributions as part of this project have been provided by Elekta AB and Ortec Logiqcare B.V..

REFERENCES

[1] Ping-Lin Chen, Chun-Jen Peng, Chang-Yi Lu, and Tian-Li Yu. 2017. Two-Edge Graphical Linkage Model for DSMGA-II. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. Association for Computing Machinery, New York, NY, USA, 745–752. https://doi.org/10.1145/3071178.3071236

[2] Chung-Yao Chuang and Wen-Lian Hsu. 2010. Multivariate Multi-Model Approach for Globally Multimodal Problems. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10)*. Association for Computing Machinery, New York, NY, USA, 311–318. https://doi.org/10.1145/1830483.1830544

[3] Alexander W. Churchill, Phil Husbands, and Andrew Philippides. 2013. Tool Sequence Optimization Using Synchronous and Asynchronous Parallel Multi-Objective Evolutionary Algorithms with Heterogeneous Evaluations. In *2013 IEEE Congress on Evolutionary Computation*. 2924–2931. https://doi.org/10.1109/CEC.2013.6557925

[4] Kalyanmoy Deb and David E. Goldberg. 1994. Sufficient Conditions for Deceptive and Easy Binary Functions. *Annals of Mathematics and Artificial Intelligence* 10, 4 (1994), 385–408. https://doi.org/10.1007/BF01531277

[5] Arkadiy Dushatskiy, Marco Virgolin, Anton Bouter, Dirk Thierens, and Peter A. N. Bosman. 2021. Parameterless Gene-pool Optimal Mixing Evolutionary Algorithms. (2021). arXiv:2109.05259 http://arxiv.org/abs/2109.05259

[6] Michael P. Fay and Michael A. Proschan. 2010. Wilcoxon-Mann-Whitney or t-Test? On Assumptions for Hypothesis Tests and Multiple Interpretations of Decision Rules. *Statistics surveys* 4 (2010), 1–39. https://doi.org/10.1214/09-SS051

[7] Brian W. Goldman and William F. Punch. 2014. Parameter-Less Population Pyramid. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*. Association for Computing Machinery, New York, NY, USA, 785–792. https://doi.org/10.1145/2576768.2598350

[8] Brian W. Goldman and William F. Punch. 2015. Fast and Efficient Black Box Optimization Using the Parameter-less Population Pyramid. *Evolutionary Computation* 23, 3 (2015), 451–479. https://doi.org/10.1162/EVCO_a_00148

[9] Georges R. Harik, Fernando G. Lobo, and Kumara Sastry. 2006. Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA). In *Scalable Optimization via Probabilistic Modeling*, Janusz Kacprzyk, Martin Pelikan, Kumara Sastry, and Erick CantúPaz (Eds.). Vol. 33. Springer Berlin Heidelberg, Berlin, Heidelberg, 39–61. https://doi.org/10.1007/978-3-540-34954-9_3

[10] Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70. https://www.jstor.org/stable/4615733

[11] Shih-Huan Hsu and Tian-Li Yu. 2015. Optimization by Pairwise Linkage Detection, Incremental Linkage Set, and Restricted / Back Mixing: DSMGA-II. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. Association for Computing Machinery, New York, NY, USA, 519–526. https://doi.org/10.1145/2739480.2754737

[12] J. Kiefer. 1953. Sequential Minimax Search for a Maximum. *Proc. Amer. Math. Soc.* 4, 3 (1953), 502–506. https://doi.org/10.1090/S0002-9939-1953-0055639-3

[13] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether One of Two Random Variables Is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50–60. https://doi.org/10.1214/aoms/1177730491

[14] Eric O. Scott and Kenneth A. De Jong. 2015. Evaluation-Time Bias in Asynchronous Evolutionary Algorithms. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. Association for Computing Machinery, New York, NY, USA, 1209–1212. https://doi.org/10.1145/2739482.2768482

[15] Eric O. Scott and Kenneth A. De Jong. 2015. Understanding Simple Asynchronous Evolutionary Algorithms. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15)*. Association for Computing Machinery, New York, NY, USA, 85–98. https://doi.org/10.1145/2725494.2725509

[16] Eric O. Scott and Kenneth A. De Jong. 2016. Evaluation-Time Bias in Quasi-Generational and Steady-State Asynchronous Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. Association for Computing Machinery, New York, NY, USA, 845–852. https://doi.org/10.1145/2908812.2908934

[17] Gilbert Syswerda. 1991. A Study of Reproduction in Generational and Steady-State Genetic Algorithms. In *Foundations of Genetic Algorithms*. Vol. 1. Elsevier, 94–101. https://doi.org/10.1016/B978-0-08-050684-5.50009-4

[18] Dirk Thierens. 1999. Scalability Problems of Simple Genetic Algorithms. *Evolutionary Computation* 7, 4 (1999), 331–352. https://doi.org/10.1162/evco.1999.7.4.331

[19] Dirk Thierens and Peter A.N. Bosman. 2011. Optimal Mixing Evolutionary Algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*. Association for Computing Machinery, New York, NY, USA, 617–624. https://doi.org/10.1145/2001576.2001661

[20] L. Darrell Whitley. 1991. Fundamental Principles of Deception in Genetic Search. *Foundations of Genetic Algorithms* 1 (1991), 221–241. https://doi.org/10.1016/B978-0-08-050684-5.50017-3

[21] Mouadh Yagoubi and Marc Schoenauer. 2012. Asynchronous Master/Slave MOEAs and Heterogeneous Evaluation Costs. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO '12)*. Association for Computing Machinery, New York, NY, USA, 1007–1014. https://doi.org/10.1145/2330163.2330303

[22] Arber Zela, Julien Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. 2022. Surrogate NAS Benchmarks: Going Beyond the Limited Search Spaces of Tabular NAS Benchmarks. https://doi.org/10.48550/arXiv.2008.09777 arXiv:2008.09777 [cs]