Improving the Scalability of Distributed Neuroevolution Using Modular Congruence Class Generated Innovation Numbers

Joshua Karns
Rochester Institute of Technology
Rochester, New York, USA
josh@mail.rit.edu

Travis Desell
Rochester Institute of Technology
Rochester, New York, USA
tjdvse@rit.edu

ABSTRACT

The asynchronous master-worker model is a classic method used to distribute evolutionary algorithms, as it can allow for decoupling of population size from the number of available processors while at the same time being naturally load balanced. While easy to implement, it suffers from an unavoidable choke point: the master process, which must process all results and generate tasks for workers. This work investigates a method for improving the performance of distributed neuroevolution algorithms, which commonly use such a model, that involves offloading costly crossover and mutation operations to the worker processes. To accomplish this, a novel modular congruence class based strategy for generating unique innovation numbers was developed, which requires no additional communication overhead. Experimental results designed to stress test the master process were generated using the Evolutionary eXploration of Augmenting Memory Models (EXAMM) neuroevolution algorithm, after discovering in preliminary results that it suffered from a bottleneck preventing scalability past 432 cores in a high performance computing environment. The results show a statistically significant improvement in throughput (genome evaluations per second) and scalability past 864 cores using this offloading method. Further, this methodology is generic and could be applied to any neuroevolution algorithm which utilize NEAT-inspired innovation numbers.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**; **Parallel algorithms**.

KEYWORDS

neuroevolution, scalability, distributed computing

ACM Reference Format:
Joshua Karns and Travis Desell. 2021. Improving the Scalability of Distributed Neuroevolution Using Modular Congruence Class Generated Innovation Numbers. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449726.3463202

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463202

1 INTRODUCTION

Evolutionary algorithms are a class of optimization algorithms which have been successfully applied to a wide variety of problems in many domains. EAs lend themselves well to parallelization, and the topic of distributed evolutionary algorithms has seen a resurgence in recent years. Gong *et al.* attribute this boom of interest to the rise of big data which has increased the size and complexity of real world optimization problems, necessitating scalable EAs [6].

Given their ease of use, flexibility, and good track record it should be no surprise that they have also been applied to the domain of neural architecture search (NAS). Often called neuroevolution (NE), evolutionary NAS algorithms search for optimal artificial neural network (ANN) architectures and their associated weights. NE presents a particularly difficult challenge as the search space of ANNs is potentially unbounded. Additionally, NE algorithms tend to be computationally expensive as evaluating genomes typically requires training the generated neural networks for a number of epochs, or at least performing a number of forward passes in order to estimate their performance [2, 3]. Due to this, having scalable distributed computing strategies to perform NE becomes especially important to generate results in a reasonable amount of time.

Still, NE is an active field of study that has found significant success. Much of the current work in the field stems from a landmark neuroevolution algorithm known as Neuroevolution of Augmenting Topologies (NEAT) [15]. One such example is an algorithm called CoDeepNEAT [11]. CoDeepNEAT simultaneously evolves a population of small module ANNs along with a population of blueprints which are graphs where the nodes point to one of the module ANNs. Combined with evolution of hyperparameters, CoDeepNEAT has achieved performance comparable with state of the art human designed models in image recognition tasks and natural language processing tasks [11]. A genetic algorithm that is not a relative of NEAT can be found with NSGA-Net, a multi-objective genetic NAS algorithm [8]. By optimizing for minimal error in addition to minimizing computational complexity, Lu *et al.* were able to find networks that are on par with the state of the art while producing networks that use significantly less computational resources [8]. Other NAS algorithms have found success without the use of evolutionary methods. Liu *et al.* used a differentiable parameterization of the neural architecture search space in their so that it can be explored using gradient descent, calling their algorithm DARTS [7]. Their approach achieved high performance on benchmark datasets while also taking significantly less time than other state of the art approaches to NAS.

Another example of a NEAT inspired algorithm, which is also the subject of this paper is an algorithm called Evolutionary eXploration of Augmenting Memory Models (EXAMM). EXAMM is a distributed

Islands	Pop/Island	Cores	Average Time (s)	Average Evals/s
1	10	108	378.212200	0.106
1	10	216	571.062600	0.14
1	10	432	405.518000	0.395
1	10	864	507.160700	0.631
1	50	108	361.143200	0.111
1	50	216	417.388111	0.192
1	50	432	330.796444	0.484
1	50	864	427.459500	0.749
1	100	108	392.112900	0.102
1	100	216	376.900200	0.212
1	100	432	326.879800	0.489
1	100	864	484.685400	0.66
5	10	108	346.810200	0.115
5	10	216	395.542600	0.202
5	10	432	399.689900	0.4
5	10	864	450.462700	0.71
5	50	108	367.362600	0.109
5	50	216	405.988100	0.197
5	50	432	414.928800	0.386
5	50	864	862.423778	0.371
5	100	108	380.968667	0.105
5	100	216	510.093500	0.157
5	100	432	805.925100	0.199
5	100	864	1732.306900	0.185

Table 1: Previously unpublished results testing the scalability of EXAMM. These results were never published because EXAMM scaled poorly with population size. In these results, candidate RNN genomes were each trained for 10 epochs, highlighting the need to have a faster master process to scale the algorithm.

neuroevolution algorithm used to evolve RNNs for time series data prediction [12]. EXAMM uses an island based speciation strategy [1, 12], which splits the population up into a fixed number of islands each with a fixed population. The overall population size can be increased by increasing either the number of islands or increasing the population size on each island. This strategy is employed using an asynchronous master-worker model: a single master process distributes tasks to workers and processes the results, shown in Figure 1.

Any algorithm that uses a master-worker model (which is the case for most distributed NE algorithms) may be subject to a bottleneck caused by the master process. If the master cannot generate tasks and process results at least as fast as the workers execute the tasks, then the algorithm will experience a drop in both parallel efficiency and speedup. Still, there are many works that explore the benefits of using asynchronous master-worker models for EAs [4, 5, 14, 16]. A preliminary examination of EXAMM's scalability, shown in Table 1, suggests that EXAMM suffers from a bottleneck when scaled to a large number of cores which was compounded as the size and number of islands was increased. The root cause of poor scaling with respect to population was determined to be logging of population statistics for each island that occurred every time the master received a result from a worker. These statistics required the master to look at every genome on every island, which explains why increasing the number of islands or number of genomes on an island was slowing things down.

When the population size remained somewhat small, these results show reasonable scalability as the number of cores is increased. With a population of 1 island and 10 members per island, going from 108 cores to 864 cores leads to a nearly 6x speedup. Although not linear, a 6x speedup with 8x the cores is still good.

Turning off any unnecessary logging is an easy way to speed up the master which improves scalability, however there was still room for improvement in the algorithm itself. The process of generating new genomes in EXAMM is relatively expensive compared to an ordinary EA because the genome is an RNN - effectively a directed acyclic graph (DAG). All genome operators (mutation and crossover) are graph-based operations which are significantly more expensive than typical mutation or crossover operations that would be found in a standard genetic algorithm. The results in this paper found that performing a mutation or crossover operation accounted for around 30% of the time spent generating genomes for workers.

This work presents a novel way to delegate the computation associated with performing a mutation or crossover operation to the worker processes. Normally, this would be trivial, however EXAMM employs system of innovation numbers (inspired by the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [15]) in its genome representation. Each time a new node or edge is added to the network via mutation, that new component will be associated with a unique number (an integer). This greatly simplifies the process of performing crossover and reliably produces more coherent child networks [15], but using this strategy makes performing mutation operations on a worker node more challenging. Allowing workers to assign new innovation numbers during mutation operations necessitates a method that will guarantee that each integer uniquely used as an innovation number for a given architectural innovation in a neural network.

In this paper, a modification to EXAMM which performs mutation operations on workers (rather than on the master) while ensuring unique usage of innovation numbers among all workers is investigated. In order to accomplish this, this work presents a novel strategy based on modular congruence classes which allows workers to independently generate unique non-conflicting innovation numbers without requiring any communication overhead. The method is tested on EXAMM and performance is compared to the vanilla version of EXAMM. Further, the hyperparameters of EXAMM used in the experiments are set to minimize the amount of computation done by workers thus requiring fewer workers to bottleneck.

2 METHODOLOGY

2.1 Evolutionary eXploration of Augmenting Memory Models

Evolutionary eXploration of Augmenting Memory Models (EXAMM) is a distributed asynchronous neuroevolution algorithm, which is distributed using the Message Passing Interface (MPI) [10] for use on high performance computing systems. EXAMM is parallelized using a master-worker model, which is summarized by Figure 1. The master process maintains a population of genomes

organized using an island speciation strategy [1]. It will generate new genomes for workers upon request via a mutation or crossover operation, selecting genomes to be modified with a round-robin schedule. Once a genome has been evaluated, it will be sent back to the master who will attempt to insert it into the population. Workers repeatedly request a new genome from the master, train it (using backpropagation), evaluate it, then finally send the genome back to the master along with its fitness value. More information about the specifics of EXAMM, including the mutations it performs and the method of crossover it uses, can found in the paper by Desell *et al.* where EXAMM was conceived [12].

Inspired by NEAT, EXAMM uses a system of innovation numbers in its genome representation to preserve innovation and overall structure. Innovation numbers were conceived to solve the competing conventions problem, which is illustrated in Figure 2. Innovation numbers solve this problem by assigning a unique number to all nodes and edges in the network, including those added to a genome through mutation. This way, when performing crossover network components with the same innovation numbers can be

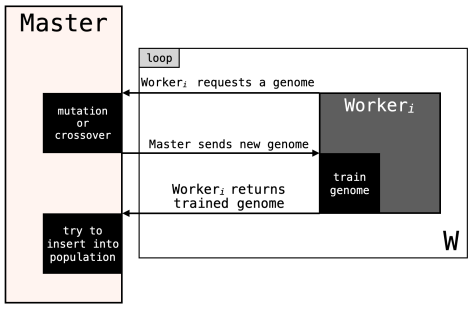

Figure 1: High level overview of the asynchronous model of distributed computation used by EXAMM, for some number of workers w.

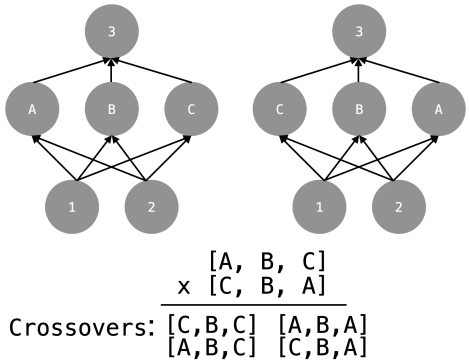

Figure 2: A basic case of the competing conventions problem, as originally illustrated in the paper detailing NEAT [15]. Two parent networks $[a, b, c]$ and $[c, b, a]$ are shown along with four possible child networks. Both parents represent the same underlying functionality in a different way, yet some of their children lose some of that information due to node ordering.

treated as the same component. However, this genome representation presents a key challenge to delegating mutations to the workers, as it necessitates a strategy to prevent workers from assigning the same innovation number to different components, preferably with no communication with the master process. If two workers were to assign the same innovation number to two (or more) different nodes (or edges), the genome encoding used by EXAMM would be broken - innovation numbers are supposed to refer to unique components, and many of the genome operations done by EXAMM rely on this property to execute correctly and would break if it were not upheld.

2.2 Proposed Method

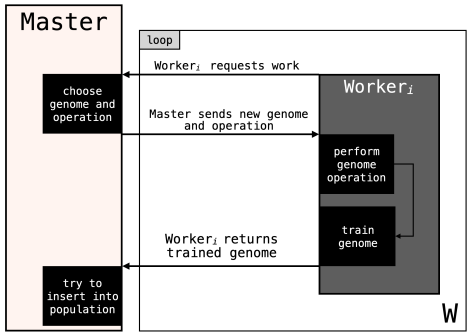

Figure 3: High level overview of EXAMM with the proposed modifications, for some number of workers w. Rather than performing the operation (mutation or crossover), the master just selects which operation to perform and the workers execute it. In the case of a crossover, the master will send two genomes to the worker.

Moving the crossover to the workers is fairly straightforward. The master can select and send the two parent genomes, and the worker can perform the crossover. Since crossover doesn't involve creating new nodes or edges, it really is as simple as change the location of a function call. However, since mutation involves the creation of new components and thus the creation of new innovation numbers, this must also be paired with a strategy to ensure unique innovation numbers are used. Modular arithmetic offers an elegant method to "allocate" different innovation numbers to workers at the start of the program.

Each worker process is assigned a congruence class in modulo w, where w is the number of workers. A congruence class modulo w is a set of integers that are all equivalent mod w, specifically the congruence class of an integer a is the set $\{a + wk | \forall k \in \mathbb{Z}\}$. Since all of the congruence classes modulo w are mutually exclusive from one another (*i.e.*, the union of all of them is the empty set), and there are exactly w different congruence classes mod w, restricting each worker to a different congruence class for innovation number assignment will guarantee there is no overlap. Thus, all a worker has to do in order to obtain the next innovation number is add w to its previously generated innovation number.

Figure 3 illustrates the proposed modifications to EXAMM. The number of communication steps is the same as in vanilla EXAMM

(shown in Figure 1), but the potentially costly step of performing a mutation or crossover operation has been moved to the worker. Reducing the amount of work the master has to do each time it generates a genome should improve the maximum throughput of the master (*i.e.* how many genomes it can generate and process per second) and therefore the scalability of the algorithm.

3 EXPERIMENTS

In order to compare the scalability of the original version of EXAMM and EXAMM with the proposed modifications, experiments were ran for both versions on Rochester Institute of Technology's high performance research computing cluster using varying numbers of processors [13]. Each experiment was also ran with a range of population sizes. Experiments each used 108, 216, 432, or 864 cores, corresponding to 3, 6, 12, and 24 cluster nodes of 36 cores each. Each number of cores was ran with a population of 1, 4, 16, and 64 islands where each island had 64 members, for a total of 32 experiments which were each repeated 10 times.

The hyperparameters given to EXAMM were chosen to maximize susceptibility to a bottleneck. In particular, the number of epochs of backpropagation used to train the genomes was 0. Normally, the workers in EXAMM use backpropagation to optimize the genome weights before calculating the fitness; this is also where workers tend to spend most of their computational resources. While removing backpropagation will result in generated networks with poor accuracy, it also means workers evaluate genomes as fast as possible to stress test the performance of the master process. This means the algorithm will experience a bottleneck sooner than it would under normal circumstances, since the workers will be requesting genomes from the master much more frequently. As this bottleneck was already found in preliminary results, as presented in Figure 1, where genomes were being trained for 10 backpropagation epochs, optimizing the speed of the master process has become important for current use cases, and would also be important for smaller data sets which can train faster.

In each experiment, a total of 10,000 genomes were evaluated on a data set consisting of 10 days worth of per-minute readings of 12 different sensors from a coal fired power plant. The data is split amongst 12 time series, each of which has 14,402 rows.

Since backpropagation was effectively disabled, only the validationing portion of the data set is looked at, as this is what EXAMM uses to determine the genome fitness and backpropagation is never used which is the only time training data is used. The validationing dataset here makes up two of the 12 files for a total of 28,804 rows, and the training dataset is composed of the remaining 10 files for a total of 144,020 rows. So for this work, with backpropagation disabled, workers only performed a forward pass to evaluation the fitness of the genome on the validationing dataset. As an interesting aside, due to EXAMM's Lamarckian weight inheritance strategy [9], EXAMM can actually evolve fairly well performing RNNs without any backpropagation, which is another justification for needing faster master performance.

That being said, in an actual use case of EXAMM where backpropagation is enabled, for each specified epoch of candidate RNN training (a user defined hyperparameter), at least one forward pass and one packward pass needs to be done for every epoch on the training data. Additionally, EXAMM will evaluate the RNN on the validation data for every epoch, as it selects fitness and weights for the network returned from the master from the epoch which performed best on the validation data. Due to using momentum and the noisy nature of stochastic backpropagation, a monotonic decrease of RNN fitness is not guaranteed during backpropagation, so this will return the best version of the RNN found. Assuming a forward pass and backward pass take roughly the same amount of time, workers perform at least $B \cdot (2 \cdot T + V)$x more work when performing backpropagation where E is the number of backpropagation epochs being done, and T and V are the time to perform a forward pass on the training and validation datasets, respectively.

Both versions of EXAMM used in the experiments performed some logging each time a evaluated genome was received from a worker, including information about each island. Thus, the time it takes to insert a genome should scale linearly with the number of islands in the population. Due to this, we would also expect to see a bottleneck sooner in experiments that have more islands.

4 RESULTS

Table 4 contains a summary of the findings. As expected, the time it takes for the master to prepare a genome for a worker is greater in the original version of EXAMM. In fact, the experimental version of EXAMM took more than 30% less time on average. The time required to insert a genome into the population appears to be quite similar between the two versions of EXAMM, and also takes more time as the number of islands increases which is also expected. The actual throughput (Average Evals / s in the table) is also better in the experimental group, especially as the number of cores is increased. Highlighting scalability issues, the throughput for the original version of EXAMM when using 432 cores is nearly identical to throughput when using 864 cores – indicative of a bottleneck limiting performance. Similarly, the amount of time required to run the experiments went down as the number of cores was increased but the effect was more significant in the experimental group.

Statistical significance tests for the throughput and genome preparation time are shown in Table 2. These results were calculated using the Mann-Whitney U-Test, comparing the populations of experiments for both versions of EXAMM. The tests found that the difference in genome preparation time was strongly significant in every case, where as the throughput results only begin to be statistically significant when 432 cores are used, which was where the original EXAMM implementation started to suffer from a bottleneck.

The throughput is also visualized with box plots in Figures 5 and 4. In Figure 5 a roughly asymptotic trend can be seen - doubling the number of cores sees diminishing returns. Further, the number of islands seems to have a negative impact on the throughput, particularly with more cores; this is to be expected though and was mentioned in Section 3.

The variance of the throughput and time to run each experiment is fairly large, as can be seen in the σ (standard deviation) columns of Table 4. Figures 5 and 4 provide a more in depth look at this, which is shown by the high number of outliers and large interquartile ranges. While the exact reasons for the high variance is not yet known (and an area of futher examination), it is likely a combination

of factors including but not limited to the low number of times the experiments were repeated (only 10 each) and the fact that the high performance computing cluster used to run the experiments is a shared computing system, performance may be helped or hindered based on what other activities are being ran on the cluster and overall cluster utilization.

Cores	Islands	Genome p-value	Throughput p-value
108	1	**0.00009**	0.06061
108	4	**0.00009**	0.13643
108	16	**0.00009**	0.42505
108	64	**0.00009**	0.26018
216	1	**0.00009**	0.45486
216	4	**0.00009**	0.06061
216	16	**0.00009**	0.33879
216	64	**0.00009**	0.23625
432	1	**0.00009**	**0.00009**
432	4	**0.00009**	0.20275
432	16	**0.00009**	0.28538
432	64	**0.00009**	**0.00141**
864	1	**0.00009**	**0.00141**
864	4	**0.00009**	**0.00009**
864	16	**0.00009**	**0.03778**
864	64	**0.00009**	**0.00141**

Table 2: P-values from statistical significance tests on the results. The Mann-Whitney U test was used here, and statistical significance is determined using an α value of 0.05; statistically significant findings are in bold. The value being tested in the "Genome" column is the amount of time it takes for the master to create a genome to be sent to a worker.

Group	Islands	Cores	Probe Time (ms)	Recv. Time (ms)
Experimental	1	108	0.1024	0.0529
Experimental	1	216	0.0170	0.0492
Experimental	1	432	0.0022	0.0469
Experimental	1	864	0.0013	0.0466
Experimental	4	108	0.1319	0.0538
Experimental	4	216	0.0089	0.0473
Experimental	4	432	0.0022	0.0465
Experimental	4	864	0.0012	0.0452
Experimental	16	108	0.0743	0.0496
Experimental	16	216	0.0082	0.0493
Experimental	16	432	0.0023	0.0461
Experimental	16	864	0.0012	0.0461
Experimental	64	108	0.0367	0.0462
Experimental	64	216	0.0026	0.0446
Experimental	64	432	0.0020	0.0443
Experimental	64	864	0.0010	0.0466
Original	1	108	0.0649	0.0523
Original	1	216	0.0029	0.0481
Original	1	432	0.0003	0.2294
Original	1	864	0.0001	0.0485
Original	4	108	0.0895	0.0525
Original	4	216	0.0020	0.0485
Original	4	432	0.0006	0.0451
Original	4	864	0.0001	0.0477
Original	16	108	0.0478	0.0528
Original	16	216	0.0018	0.0477
Original	16	432	0.0004	0.0786
Original	16	864	0.0001	0.0466
Original	64	108	0.0231	0.0476
Original	64	216	0.0019	0.0438
Original	64	432	0.0007	0.0459
Original	64	864	0.0001	0.0463

Table 3: This table contains additional results investigating the time spent sending and receiving genomes. The probe time is the average amount of time spent waiting by the master for either a result or a work request from a worker. The receive time is the average amount of time it took to receive an evaluated genome from a worker.

5 DISCUSSION

This work presents an examination of the scalability and performance of the Evolutionary eXploration of Augmenting Memory Models (EXAMM) neuroevolution algorithm. After finding that EXAMM's master process suffered from a bottleneck, a strategy was developed to offload mutation and crossover operations to worker processes. In order to accomplish this, a novel modular congruence class based methodology for generating non-conflicting innovation numbers across distributed worker processes was developed, which requires no communication between the master or workers for generation of new innovation numbers.

Experimental results support the idea that this method can improve the scalability of a NEAT-like distributed neuroevolution algorithms. The results show similar performance between the original and the experimental versions of EXAMM for a small number of cores, but after scaling the number of cores to 432 they begin to diverge, as this is when the original implementation began to significantly suffer from a performance bottleneck. This divergence is also reflected by statistical significance tests as all experiments using 864 cores were found to be statistically significant but all experiments using less than 432 cores were not. Figure 4 illustrates this well: the performance of the original version of EXAMM is virtually identical with 12 and 24 nodes (432 and 864 cores respectively) while the experimental version enjoys a performance bump, allowing EXAMM to scale to a significantly larger number of worker processes.

While this methodology provided a significant scalability increase for EXAMM, there still remains room for improvement as the speedup still had some distance from a linear improvement. For example, going from 108 cores to 864 uses 8 times the cores yet only yielded less than a 3 times increase throughput. However, this was done with experiments that were designed to stress test the performance of the master process, with the workers not performing any computationally expensive backpropagation. A more realistic usage of EXAMM will see an even more significant increase in throughput, as each worker would be doing multiple epochs of backpropagation for each genome evaluated. Adding in just one epoch of backpropagation over the entire data set would significantly increase the amount of time it would take for a worker to process a genome from the master, which would in turn will allow for even more scalability.

Group	Islands	Cores	Ins. Time (ms)	Genome Time (ms)	Average Time (s)	Time (s) σ	Average Evals / s	Evals / s σ
Experimental	1	108	0.0938	0.0866	5.1861	1.1386	2003.482	352.457
Experimental	1	216	0.0688	0.0771	2.7025	0.4487	3808.620	660.444
Experimental	1	432	0.0569	0.0706	2.0547	0.1508	4893.533	363.886
Experimental	1	864	0.0613	0.0640	2.0269	0.3256	5037.771	658.523
Original	1	108	0.0780	0.1210	4.5136	0.6032	2252.037	274.058
Original	1	216	0.1056	0.1059	3.1714	1.3090	3533.893	945.502
Original	1	432	0.0955	2.0093	2.7397	0.6759	3820.363	690.999
Original	1	864	0.0500	0.0987	2.2849	0.0745	4381.045	137.715
Experimental	4	108	0.0676	0.0924	5.7141	1.2159	1843.706	447.010
Experimental	4	216	0.0667	0.0795	2.5292	0.4862	4080.393	667.042
Experimental	4	432	0.0962	0.0724	2.4361	0.5439	4269.469	752.692
Experimental	4	864	0.0477	0.0646	1.8678	0.1121	5372.972	318.601
Original	4	108	0.0717	0.1293	5.1239	0.4180	1964.180	154.472
Original	4	216	0.0619	0.1147	2.5621	0.1552	3916.380	220.662
Original	4	432	0.0802	0.1021	2.5477	0.7673	4138.950	717.466
Original	4	864	0.0659	0.0983	2.4146	0.3306	4202.267	446.514
Experimental	16	108	0.0847	0.0812	4.7145	1.1115	2235.575	492.414
Experimental	16	216	0.1031	0.0802	2.9785	0.7918	3529.944	666.991
Experimental	16	432	0.0880	0.0700	2.3257	0.3673	4411.264	710.381
Experimental	16	864	0.0893	0.0664	2.3087	0.4375	4458.710	681.007
Original	16	108	0.0842	0.1248	4.5715	0.4830	2214.325	255.534
Original	16	216	0.0762	0.1104	2.6399	0.1916	3807.890	274.927
Original	16	432	0.0981	0.4515	2.4445	0.1506	4104.361	219.994
Original	16	864	0.0649	0.0950	2.3516	0.1083	4261.752	203.003
Experimental	64	108	0.1396	0.0668	3.7038	0.4798	2745.740	358.732
Experimental	64	216	0.1220	0.0623	2.5333	0.3592	4025.903	558.467
Experimental	64	432	0.1064	0.0610	2.3660	0.2224	4258.120	337.258
Experimental	64	864	0.1054	0.0587	2.3881	0.2689	4237.215	447.744
Original	64	108	0.1430	0.0973	3.7680	0.7791	2733.082	386.833
Original	64	216	0.1066	0.0900	2.6443	0.2431	3810.400	313.027
Original	64	432	0.1030	0.0912	2.6446	0.0758	3784.411	108.946
Original	64	864	0.1068	0.0895	2.6892	0.2779	3752.108	325.245

Table 4: Results of each experiment. "Ins. Time" is the average amount of time it took for the master to insert a genome, and "Genome Time" is the amount of time it takes for the master to select a genome to be sent to the worker (and perform mutation or crossover in the case of the original group), this does not include the time it takes to send the genome(s).

5.1 Future Work

Table 1 contains results from previous experiments looking at how EXAMM scaled with population size and core count. In addition to showing that offloading mutation and crossover to worker processes provided a strong improvement in scalability, an additional area of slowdown has been identified (namely unnecessary logging). Additional experiments should be run to investigate the scalability of EXAMM on a real world dataset as well as the impact of population size and core count on network performance, without any logging, as well as methods to offload logging (perhaps to a concurrent thread or process) to further reduce its impact. Moreover, future experiments should include backpropagation: the experiments here demonstrate improved performance of the master process in a worst-case scenario (workers evaluate genomes significantly quicker than they do in a normal use case), but demonstrating material improvement in scalability during a more realistic use case of EXAMM would make for significantly stronger results.

In addition to this, any algorithm that uses a master-worker model is inevitably going to fall victim to a bottleneck caused by the master. In the case of EXAMM, the next step for improving performance is to separate the master (which currently manages all island populations) into multiple processes, which of which could manage a subset of islands, as well as one overall master process operating in a hierarchical manner. This can be leveraged to effectively distribute both the communication and computational load of the master process to significantly raise the throughput ceiling. Each island could be assigned to one of the multiple island processes that will act as its manager, and each of these island processes would have their own set of worker processors that train and evaluate genomes. Each island process would only have to communicate with the master process to send it new island-local best genomes and to receive the best genomes from other islands. For potentially further improvements, communication with the master could be unnecessary all together, as seen in the taxonomy of distributed EAs given by Gong *et al.* in their 2015 work which

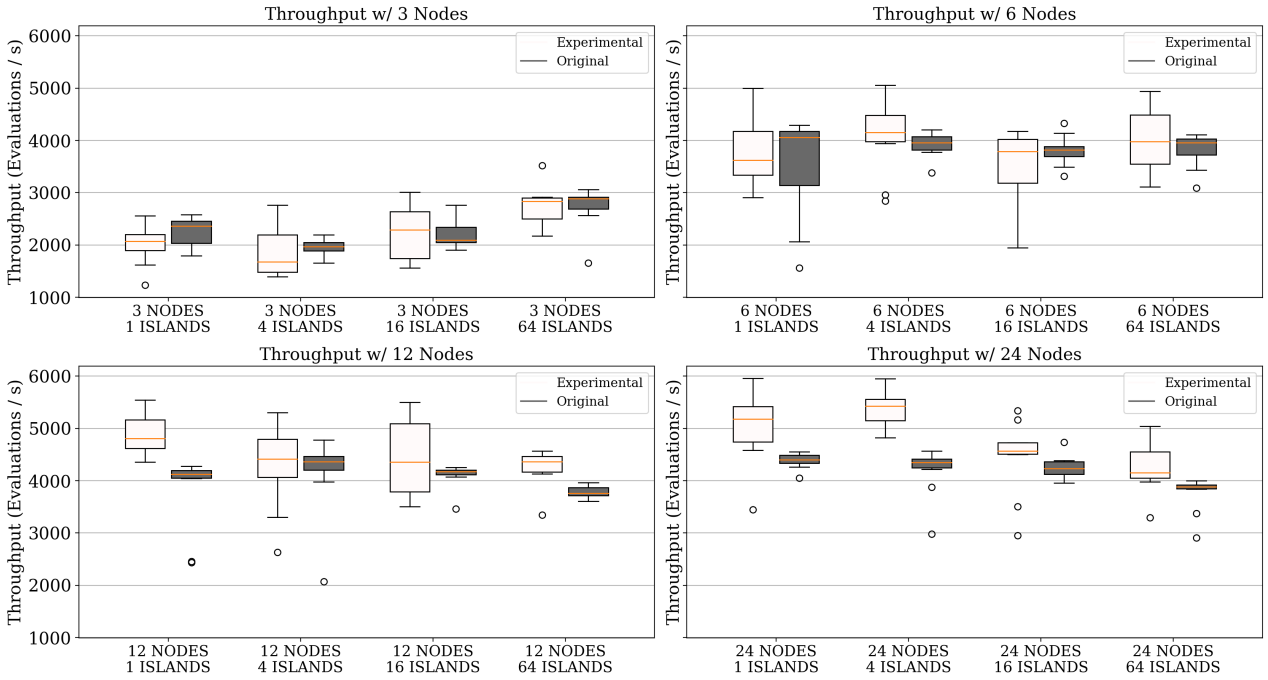

Figure 4: Throughput values separated by the number of nodes they used. The experimental group is shown in white, and the original group in gray.

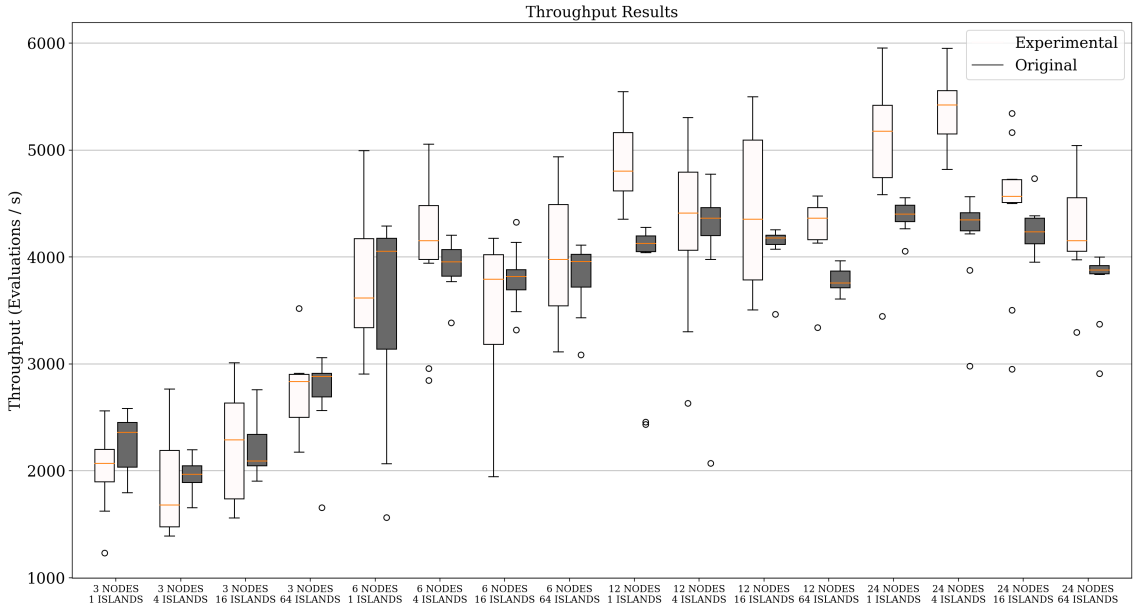

Figure 5: This table contains the same results as Table 4, but everything is lined up horizontally to make the overall throughput trend more clear.

discusses examples of such algorithms [6]: islands can directly communicate with one another, rather than using a master process as an intermediary.

In addition to these methods to improving scalability of EXAMM, the effect of multiple islands and numbers of worker processes has not been well studied for distributed neuroevolution algorithms.

While it may be possible to greatly increase the throughput of genome evaluation, this may not necessarily result in a corresponding improvement in the convergence rate of the neuroevolution algorithm. Studying this would help paint a more full picture of how far neuroevolution algorithms can really scale.

REFERENCES

[1] Alba, E., and Tomassini, M. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 6*, 5 (2002), 443–462.

[2] Camero, A., Toutouh, J., and Alba, E. Random error sampling-based recurrent neural network architecture optimization. *Engineering Applications of Artificial Intelligence 96* (2020), 103946.

[3] Camero, A., Wang, H., Alba, E., and Bäck, T. Bayesian neural architecture search using a training-free performance metric. *Applied Soft Computing* (2021), 107356.

[4] Depolli, M., Trobec, R., and Filipič, B. Asynchronous master-slave parallelization of differential evolution for multi-objective optimization. *Evolutionary computation 21*, 2 (2013), 261–291.

[5] Desell, T. *Asynchronous Global Optimization for Massive-Scale Computing*. PhD thesis, Rensselaer Polytechnic Institute, 2009.

[6] Gong, Y.-J., Chen, W.-N., Zhan, Z.-H., Zhang, J., Li, Y., Zhang, Q., and Li, J.-J. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing 34* (2015), 286–300.

[7] Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search, 2019.

[8] Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., and Banzhaf, W. Nsga-net: Neural architecture search using multi-objective genetic algorithm, 2019.

[9] Lyu, Z., ElSaid, A., Karns, J., Mkaouer, M., and Desell, T. An experimental study of weight initialization and lamarckian inheritance on neuroevolution. *The 24th International Conference on the Applications of Evolutionary Computation (EvoStar: EvoApps)* (2021).

[10] Message Passing Interface Forum. MPI: A message-passing interface standard. *The International Journal of Supercomputer Applications and High Performance Computing 8*, 3/4 (Fall/Winter 1994), 159–416.

[11] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. Evolving deep neural networks, 2017.

[12] Ororbia, A., Elsaid, A. A., and Desell, T. Investigating recurrent neural network memory structures using neuro-evolution, 2019.

[13] Rochester Institute of Technology. Research computing services, 2019.

[14] Scott, E. O., and De Jong, K. A. Understanding simple asynchronous evolutionary algorithms. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII* (2015), pp. 85–98.

[15] Stanley, K. O., and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary computation 10*, 2 (2002), 99–127.

[16] Zăvoianu, A.-C., Lughofer, E., Koppelstätter, W., Weidenholzer, G., Amrhein, W., and Klement, E. P. On the performance of master-slave parallelization methods for multi-objective evolutionary algorithms. In *International Conference on Artificial Intelligence and Soft Computing* (2013), Springer, pp. 122–134.