A Parallel Island Model for Biogeography-Based Classification Rule Mining in Julia

Samuel Ebert
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
saebert@ncsu.edu

Effat Farhana
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
efarhan@ncsu.edu

Steffen Heber
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
sheber@ncsu.edu

ABSTRACT

In this paper, we present a distributed island model implementation of biogeography-based optimization for classification rule mining (island BBO-RM). Island BBO-RM is an evolutionary algorithm for rule mining that uses Pittsburgh style classification rule encoding, which represents an entire ruleset (classifier) as a single chromosome. Our algorithm relies on biogeography-based optimization (BBO), an optimization technique that is inspired by species migration pattern between habitats. Biogeography-based optimization has been reported to perform well in various applications ranging from function optimization to image classification. A major limitation of evolutionary rule mining algorithms is their high computational cost and running time. To address this challenge, we have applied a distributed island model to parallelize the rule extraction phase via BBO. We have explored several different migration topologies and data windowing techniques. Our algorithm is implemented in Julia, a dynamic programming language designed for high-performance and parallel computation. Our results show that our distributed implementation is able to achieve considerable speedups when compared to a serial implementation. Without data windowing, we obtain speedups up to a factor of 9 without a loss of classification accuracy. With data windowing, speedups up to a factor of 30 were obtained with a small loss of accuracy in some cases.

CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; *Supervised learning by classification*; *Rule learning*;

KEYWORDS

Evolutionary algorithms, biogeography-based optimization, island model

ACM Reference Format:
Samuel Ebert, Effat Farhana, and Steffen Heber. 2018. A Parallel Island Model for Biogeography-Based Classification Rule Mining in Julia. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208262

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208262

1 INTRODUCTION

The goal of classification is to identify the correct class of new data points based on a given training data set with known class labels. Classification rules are rules of the form IF **P** THEN **C**, or **P**→**C**, where **P** is the conjunction of attribute tests and **C** is the assigned class label. Rule-based classification is a classification approach that uses a set of classification rules to assign a class label to new data points. Rule-based classifiers can often achieve a similar accuracy to popular black-box classifiers such as support vector machines [12] or neural networks [38], and have the additional benefit of being extremely easy to use, explain, and interpret. This makes them particularly suitable for applications where both predictive accuracy and comprehensibility are important. Examples of such applications include credit risk evaluation [26], software defect prediction [24], and disease classification [25]. There are various approaches for classification rule induction, including algorithms that rely on association rules [23], decision tree-based methods [32], and algorithms that use rough sets [29]. One of the most successful rule induction paradigms is genetics-based machine learning (GBML). GBML-based methods offer a balance between exploring the search space and improving the promising solutions simultaneously [14]. However, these methods are often computationally expensive and require a high number of fitness function evaluations. This becomes particularly problematic for large data sets.

Due to the rapidly-increasing availability of cluster computing and general-purpose computing on graphics processing units (GPUs), parallel computing has emerged as a powerful tool to reduce running time and to make large data sets tractable for GBML-based approaches. Several methods exist for the parallelization of GBML algorithms. The most straightforward is the master-slave approach, which computes fitness evaluations in parallel while otherwise running the same algorithm as a serial approach [9]. A closely-related approach makes use of GPUs for this fitness computation [43]. Other methods of parallelization change the structure of the algorithm beyond simply parallelizing the fitness evaluation. These non-embarrassingly parallel methods are generally divided into two classes: fine-grained and coarse-grained. Fine-grained methods, such as cellular evolutionary algorithms, maintain a single population with individuals only being able to interact with individuals in their neighborhood in the population. These are often distributed in a manner that places each individual on its own processor core [10]. Coarse-grained methods, also referred to as island models, maintain several subpopulations in parallel that evolve independently

and exchange information at periodic intervals [40]. Results in the literature report that these island model implementations can show significant and sometimes even superlinear speedups [3, 4, 11, 22]. In addition to speedups, the island model has shown the potential to increase the quality of the final solution, as discussed in [42]. See [37] for a general overview of the structure and implementation of island model genetic algorithms.

In this paper, we evaluate performance of island models in the context of GBML. We have selected BBO-RM [13], which generates classification rules via biogeography-based optimization (BBO). BBO is a modern EA that was first proposed by Dan Simon [35]. BBO-RM mines rules within the Pittsburgh based paradigm [27]. In Pittsburgh-style encoding, one chromosome represents an entire ruleset. As the rules of a ruleset evolve simultaneously, this representation is capable of accounting for interactions between classification rules. As a consequence, the resulting rulesets of Pittsburgh-style systems often perform better than classifiers that consist of individually optimized rules [14]. Unfortunately, the good performance of the Pittsburgh-style encoded rulesets is often combined with high computational costs. To address this challenge, we have parallelized BBO-RM using a distributed island model. To further increase performance, our implementation uses the dynamic high-performance programming language Julia. Introduced in 2012, the Julia language is designed with an emphasis on performance, parallelization, and technical computing [6]. Julia uses native just-in-time compilation based on a low-level virtual machine [21]; benchmarks place the language in a similar class as static-typed languages such as C and C++. To facilitate additional performance increases, Julia supports concurrent, multi-core, and distributed computing [2].

The contributions of this paper are as follows: we develop an island model for biogeography-based rule mining island BBO-RM. We explore the effect of different migration topologies and data windowing on the performance of our island model implementation. We utilize the performance and parallelization features of a modern programming language, Julia. We report our experimental results in the context of eight benchmark datasets from the UCI machine learning repository [8]. For evaluation metrics, we report accuracy and elapsed time to compare the performance of the single population and island models.

2 RELATED WORK

Island model evolutionary algorithms have been successfully applied to a wide range of complex problems. One of the earliest advances in this field was the GENITOR II algorithm, proposed in 1990 by Whitley and Starkweather [42]. The algorithm discussed used a distributed population of 10 islands arranged in a ring. The authors compared the performance of this island model implementation to the serial GENITOR algorithm and found that the island model consistently outperformed the serial one [39]. In addition to the performance advantages, the authors noted a greater robustness of the distributed version, citing an ability to find high-quality solutions without the need for extensive parameter tuning.

Whitley et al. considered the application of island model genetic algorithms to linearly separable problems [41]. The authors hypothesized that island models could have an advantage for these problems due to the greater search diversity provided by multiple islands; each island could work toward solving a different subproblem of the linearly-separable problem. The authors reported mixed results in tests of this framework; the island model was reported to outperform the single-population on a four-bit deceptive function but did not show the same advantages on other functions. The authors noted that among the different configurations of island models tested, the best performance was achieved by dividing the given population size into high numbers of islands and small population sizes.

Several authors have investigated the impact of migration policy and migration topology on the performance of island model genetic algorithms. Guan and Szeto tested a wide variety of random migration topologies and found that the best performance was reached when the connectivity level in the graph was between 40 and 70 percent [16]. Direct comparisons of migration topologies have shown mixed results; depending on the details of the algorithms and problems tested, different authors have reported that the star [5], torus [20], or modified ring and hypercube [31] performed best. Sekaj tested several topologies of the island model on multiple benchmark functions and found that the best-performing topologies varied based on the function being optimized [33]. For smooth functions, a dense topology with frequent migration performed best, while for more difficult functions, a sparser topology with less frequent migration performed best. The paper suggested that this was due to the greater preservation of genetic diversity provided by a more isolated model, which was necessary to solve difficult functions. These mixed results in the literature suggest that there is not a single optimal migration configuration that can be relied on to perform well on all problems.

Although a great deal of work has been done in the field of island model genetic algorithms, most research has focused on the application of these algorithms to the optimization of parametric benchmark functions. Their application to GBML has not been as extensively studied. However, several papers have made contributions to this field. Ishibuchi et al. proposed an island model implementation of a Pittsburgh-style GBML system for fuzzy rule mining [18]. The work describes a system of training data rotation between islands to further speed up execution. The authors found that the island model was able to greatly reduce execution time on the datasets tested. The island model has also been applied for Michigan GBML. Sharma and Saroj found that the island model led to a considerable increase in classification accuracy compared to a single population in a Michigan GMBL system [34]. Srinivasa et al. found that an island model Michigan GBML system with adaptive population sizing performed favorably when tested against several other machine learning techniques [36]. This body of previous research into island model genetic algorithms shows the effectiveness of these techniques for a wide range of problems.

3 BBO-RM ALGORITHM

This section presents a brief overview the BBO algorithm and the biogeography-based rule miner BBO-RM.

3.1 Overview of BBO

BBO is an evolutionary algorithm (EA) motivated by species migration patterns through time and space. BBO assumes an archipelago where each island represents a candidate solution of an optimization problem. The key difference between BBO and other EAs is BBO's use of a migration scheme instead of a crossover operation. In BBO, solutions exchange species (genes) via multi-parent, fitness proportionate probabilities instead of the two-parent, fixed crossover rate policy used by traditional EAs. Islands with high fitness tend to resist changes, while less-fit islands tend to receive species from good islands and improve their fitness. Mutation and elitism are applied similarly to other EAs, as shown in Fig. 4a. BBO has been successfully employed in benchmark function optimization as well as many real-life problems, including economic load dispatch [7] , flexible job shop scheduling [30], image classification [28], and many others [17].

3.2 BBO-RM Classifier

BBO-RM uses Pittsburgh-style encoding. Each chromosome corresponds to a candidate solution (list of classification rules). Different solutions might contain different numbers of rules. However, there are lower and upper bounds for the number of rules in a solution. Each rule R contains a certain number $attR \in [attMin, attMax]$ of attribute tests. For discrete features, attribute tests have the form $Attribute = Value$ or $Attribute \neq Value$. For continuous features, tests are of the form $Attribute \in [Lower\_Bound, Upper\_Bound]$.

Population initialization is performed via a mixed approach that uses both randomly generated rules (20% of all rules with equal proportion of class labels) and seeded rules [19]. The seeding procedure selects a training example via sampling without replacement and generalizes it as a rule. For continuous values, the bounds take values of $[seed - intervalLength/2, seed + intervalLength/2]$, where the variable $intervalLength$ is randomly initialized with values chosen from a uniform distribution with range between 25% and 75% of the domain size. The class label of randomly generated rules is set to the class with the highest frequency among the data points in the training set that cover the rule.

The mutation operation involves selecting a ruleset with a fixed mutation probability, selecting a rule in this ruleset with uniform probability, and then mutating one attribute of the rule. Mutation alters bounds of a numerical attribute. For a nominal attribute, a value is randomly chosen from the domain.

BBO-RM uses an accuracy-based fitness function. The classifier is a sorted rule list where the first matching rule determines the class label. The default class is the majority class of the training dataset. A data point x is assigned the label of the default class if no rule covers x.

4 DISTRIBUTED BBO-RM ALGORITHM

4.1 Island Model

Algorithm 1 provides an overview of the island model implementation of BBO-RM. This framework incorporates the BBO-RM algorithm outlined in the previous section into a distributed island model. Our implementation consists of nine islands, each running an independently-initialized instance of the BBO-RM algorithm. Each island communicates with the others in the system at set intervals, exchanging information through migration and data windowing. At the conclusion of the evolution process, each island sends its best individual to the master process and the master process chooses the best of these individuals as the final solution. These processes are discussed in detail in their respective sections.

Algorithm 1 Overview of Island BBO-RM

```
1:  for each island in parallel do
2:      Initialize population of n individuals
3:      Compute fitness of each individual
4:      while generations < maxGenerations do
5:          Save the k elite members in the population
6:          Calculate immigration and emigration rates
7:          Perform migration between candidate solutions
8:          Apply mutation operator with given probability
9:          Compute fitness of each population members
10:         Replace the k worst individuals with the k elites from
                the previous generation;
11:         if generations mod migrationInterval == 0 then
12:             Send a copy of the best individual to a neighboring
                    island;
13:             Receive new individuals from neighboring islands
14:             if Windowing == true then
15:                 Send training data slice to a neighboring island
16:                 Receive new training data from a neighboring
                        island;
17:             end if
18:         end if
19:     end while
20:     Send best individual to the master process
21: end for
```

4.2 Migration

Our island model uses a synchronous migration strategy, where all islands exchange information after the same number of generations. A parameter for migration interval dictates the frequency of this information exchange. During the migration step, each island sends a copy of its best individual to a neighboring island. This migrated individual replaces the worst individual in the target population. This migration scheme was chosen based largely on the findings of CantÃž-Paz in [11], which found that a best-replace-worst migration scheme led to the fastest convergence of several possible strategies.

We test our island model on several different migration topologies, which control the possible communication links between islands. We have chosen to study three topologies: the ring, grid, and fully-connected topologies. Each topology is represented by a graph with islands as vertices and communication links as edges; each island can communicate only with islands adjacent to it in the graph. The choice of topology affects the process used to choose a destination for migrants. In the fully-connected and grid topologies, islands send their migrants to a randomly-selected adjacent island. In the ring topology, each island has designated neighbors and will always send to and receive from the same respective islands. As a consequence of this design, at each migration step, islands in

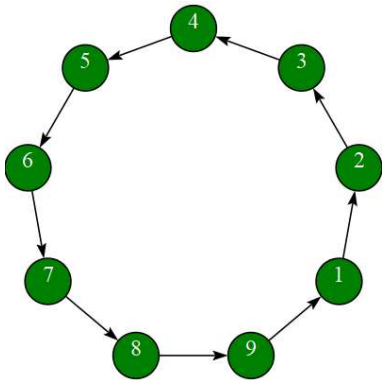

Figure 1: Ring Migration Topology

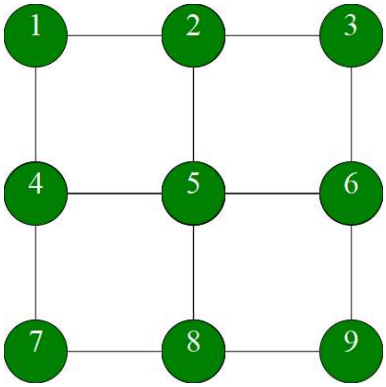

Figure 2: Grid Migration Topology

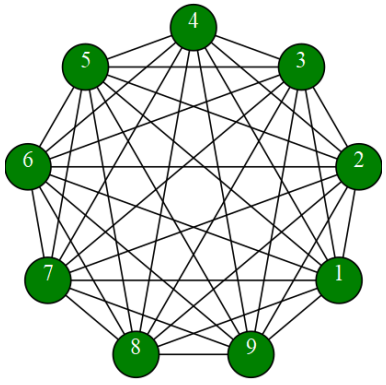

Figure 3: Fully-connected migration topology

the grid and fully-connected topologies can receive up to as many migrants as their degree in the graph, while islands in the ring topology will receive exactly one new migrant. In the event that an island receives n migrants, these migrants replace the n worst individuals in the island's population. The three migration topologies are shown in Figures 1, 2, and 3, respectively. Fig. 4 depicts the relation between the serial BBO-RM algorithm and our island BBO-RM implementation.

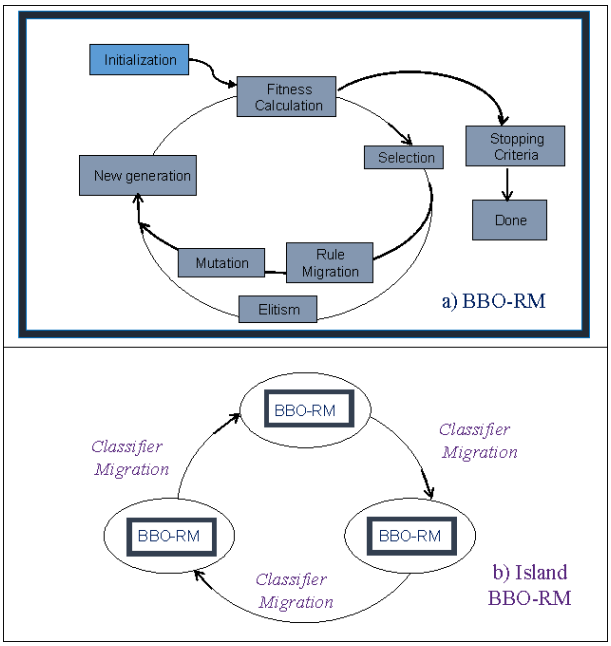

Figure 4: Illustration of the BBO-RM algorithm (a) and its relation to Island BBO-RM (b)

4.3 Data Windowing

To further reduce execution time, we have implemented a system of training data windowing. Our windowing process is implemented based on the one described in [18]. In our implementation, the original training data is divided into nine stratified, mutually-exclusive slices, each of which is assigned to an individual island. At each migration step, islands rotate their assigned window of training data according to their neighbors in the graph. The authors of [18] noted that rotating data along the same link as the direction of chromosome migration caused performance to degrade. To avoid this problem, the authors of [18] set training data rotation to be in the opposite direction of migration. Our work uses the same strategy in the ring topology. For the grid topology, we have derived a graph that ensures that training data can never move along the same link as a migrating individual. Since this problem cannot be avoided in a fully-connected topology, we use the same windowing topology for the grid and fully-connected topologies. These data windowing topologies are shown in Fig. 5.

Fig. 6 provides an example of our use of Julia's parallelization features. The figure shows a code snippet that sketches our island model implementation of BBO-RM and illustrates how the individual island sub-populations are launched. We have used the single program multiple data (SPMD) mode from Julia's DistributedArrays package, which facilitates the execution of functions in parallel on all processor cores. The package presents functionality similar to that found in MPI [1]. Within the functions shown for migration and windowing, our implementation uses the *sendto* and *recvfrom* functions from the SPMD package to communicate between islands.

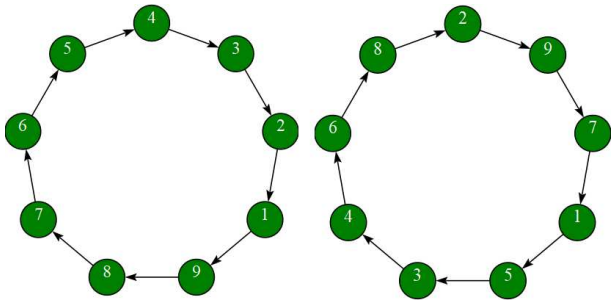

Figure 5: Data windowing topologies used with the ring (left) and fully-connected and grid (right) migration topologies

```
addprocs(9)

@everywhere function islandBBORM(train, test, trainingDataSlices, popSize,
                pMutate, topology, usesWindowing, maxGen, migrationInterval)
  neighbors = setNeighbors(topology)
  trainingData = getWindow(train, trainingDataSlices)
  population, minCost, avgCost, minParValue, maxParValue,
                      attTypes, elitePopulation, island = init(... )

  for generation in 1:maxGen
    saveElites(population, elitePopulation)
    migrationWithinPopulation(population, island)
    mutation(population, island)
    restoreIslandToPopulation(population, island)
    calculateFitness(population)
    restoreElites(population, elitePopulation)
    computeCost(population)
    sortByCost(population)

    # Perform migration and data windowing between islands
    if generation % migrationInterval == 0
      barrier()
      migrationBetweenIslands(neighbors, topology)
      dataWindowing(neighbors, topology)
      computeCost(population, trainingData)
      sortByCost(population)
    end
  end
  ...
end

# Read in train and test data and partition training data
...

# Run the function in parallel on all all 9 cores using the grid topology
spmd(islandBBO, trainData, testData, trainDataSlices, 25,
                                  0.6, "G", true, 50, 10)
```

Figure 6: Code snippet detailing our use of Julia for parallel computing

5 EXPERIMENTAL SETUPS

5.1 Datasets

We have tested our implementation on eight datasets from the UCI Machine Learning Repository [8]. We have selected datasets with a wide variety of characteristics. For example, we have chosen datasets with binary and multiclass datasets, datasets with a mix of continuous and categorical attributes, and datasets with only numerical attributes. Our datasets range in size from roughly 2000 to 58000 instances and 6 to 36 attributes. All datasets and their characteristics are shown in Table 1. Missing values in the adult dataset were removed in preprocessing.

5.2 Parameter Settings

Parameters were used to set the mutation rate, population size, number of elites, migration interval, and generation count. For both the single-population and island model tests, we use a mutation rate of 0.6 and a generation count of 50. The value for mutation rate was chosen to match the value in [13], while the value for generation count was chosen based on our observations of typical convergence times in BBO-RM. We use a population size of 25 per island in the island models and a size of 225 for the single-population model. As our goal was to compare the performance of the island models to the single population of the same overall size, we chose these values to provide a sufficient population size for each of the islands while still ensuring that the single-population size did not become excessive. In the island models, we perform migration and data windowing every 10 generations.

Table 1: Comparison of percentages.

Dataset	n	R	N	C
Shuttle	58000	9	0	7
Adult	48842	8	6	2
Bank Marketing	45211	7	9	2
Occupancy	20560	6	0	2
Magic	19020	10	0	2
Satellite	6435	36	0	7
Page Block	5473	10	0	5
Segment	2310	19	0	7

5.3 Experimental Setup

Experiments were performed on the three migration topologies discussed in Section 4.2. For each topology, we tested performance with and without data windowing. We also tested the single-population model with the same total population size. For all configurations, we performed two repetitions of 10-fold cross-validation for a total of 20 observations. All reported results are the average of these 20 values. All experiments were run on a high-performance computing cluster running CentOS Linux release 7.2.1511 (Core) using Julia version 0.6.2. The serial version ran on a single core, while all island models used nine cores.

5.4 Results

Table 2 shows accuracy and time measurements for all configurations. 95% confidence intervals were computed for all values using the t-statistic and are displayed below the averages.

Overall, these results show that the island models are able to achieve considerable speedups while maintaining comparable accuracy to the single-population version. For five of the eight datasets, the best-performing island model had a higher accuracy than the single population model. For island models without windowing, speedups ranged from 3.7 to 8.9 compared to the serial version. For models with windowing, speedups ranged from 17.8 to 30.2. The highest speedups were seen on the segment, page and bank marketing datasets. Speedup values for all configurations are plotted in Fig. 7. Most island models without windowing achieved a very similar accuracy to the single-population model. Models with windowing tended to perform slightly worse than those without

Table 2: Comparison of percentages.

Dataset	W	Single-Population		Fully-Connected		Grid		Ring	
		Accuracy (95% CI)	Time (s) (95% CI)	Accuracy (95% CI)	Time (s) (95% CI)	Accuracy (95% CI)	Time (s) (95% CI)	Accuracy (95% CI)	Time (s) (95% CI)
Shuttle	0	94.56 (93.43, 95.69)	18310.9 (18203, 18419)	94.84 (93.72, 95.97)	3718.36 (3436, 4001)	95.00 (94.01, 95.99)	4193.91 (3990, 4397)	94.99 (93.80, 96.17)	3783.54 (3543, 4023)
	1	-	-	94.47 (93.57, 95.38)	851.55 (813.7, 889.4)	94.72 (93.75, 95.69)	931.51 (890.5, 972.5)	93.62 (92.54, 94.70)	883.72 (839.0, 928.4)
Adult	0	94.56 (93.43, 95.69)	18310.9 (18203, 18419)	94.84 (93.72, 95.97)	3718.36 (3436, 4001)	95.00 (94.01, 95.99)	4193.91 (3990, 4397)	94.99 (93.80, 96.17)	3783.54 (3543, 4023)
	1	-	-	94.47 (93.57, 95.38)	851.55 (813.7, 889.4)	94.72 (93.75, 95.69)	931.51 (890.5, 972.5)	93.62 (92.54, 94.70)	883.72 (839.0, 928.4)
Bank	0	94.56 (93.43, 95.69)	18310.9 (18203, 18419)	94.84 (93.72, 95.97)	3718.36 (3436, 4001)	95.00 (94.01, 95.99)	4193.91 (3990, 4397)	94.99 (93.80, 96.17)	3783.54 (3543, 4023)
	1	-	-	94.47 (93.57, 95.38)	851.55 (813.7, 889.4)	94.72 (93.75, 95.69)	931.51 (890.5, 972.5)	93.62 (92.54, 94.70)	883.72 (839.0, 928.4)
Occupancy	0	94.56 (93.43, 95.69)	18310.9 (18203, 18419)	94.84 (93.72, 95.97)	3718.36 (3436, 4001)	95.00 (94.01, 95.99)	4193.91 (3990, 4397)	94.99 (93.80, 96.17)	3783.54 (3543, 4023)
	1	-	-	94.47 (93.57, 95.38)	851.55 (813.7, 889.4)	94.72 (93.75, 95.69)	931.51 (890.5, 972.5)	93.62 (92.54, 94.70)	883.72 (839.0, 928.4)
Magic	0	94.56 (93.43, 95.69)	18310.9 (18203, 18419)	94.84 (93.72, 95.97)	3718.36 (3436, 4001)	95.00 (94.01, 95.99)	4193.91 (3990, 4397)	94.99 (93.80, 96.17)	3783.54 (3543, 4023)
	1	-	-	94.47 (93.57, 95.38)	851.55 (813.7, 889.4)	94.72 (93.75, 95.69)	931.51 (890.5, 972.5)	93.62 (92.54, 94.70)	883.72 (839.0, 928.4)
Satellite	0	94.56 (93.43, 95.69)	18310.9 (18203, 18419)	94.84 (93.72, 95.97)	3718.36 (3436, 4001)	95.00 (94.01, 95.99)	4193.91 (3990, 4397)	94.99 (93.80, 96.17)	3783.54 (3543, 4023)
	1	-	-	94.47 (93.57, 95.38)	851.55 (813.7, 889.4)	94.72 (93.75, 95.69)	931.51 (890.5, 972.5)	93.62 (92.54, 94.70)	883.72 (839.0, 928.4)
Page	0	94.56 (93.43, 95.69)	18310.9 (18203, 18419)	94.84 (93.72, 95.97)	3718.36 (3436, 4001)	95.00 (94.01, 95.99)	4193.91 (3990, 4397)	94.99 (93.80, 96.17)	3783.54 (3543, 4023)
	1	-	-	94.47 (93.57, 95.38)	851.55 (813.7, 889.4)	94.72 (93.75, 95.69)	931.51 (890.5, 972.5)	93.62 (92.54, 94.70)	883.72 (839.0, 928.4)
Segment	0	94.56 (93.43, 95.69)	18310.9 (18203, 18419)	94.84 (93.72, 95.97)	3718.36 (3436, 4001)	95.00 (94.01, 95.99)	4193.91 (3990, 4397)	94.99 (93.80, 96.17)	3783.54 (3543, 4023)
	1	-	-	94.47 (93.57, 95.38)	851.55 (813.7, 889.4)	94.72 (93.75, 95.69)	931.51 (890.5, 972.5)	93.62 (92.54, 94.70)	883.72 (839.0, 928.4)

windowing. One major outlier in both categories was the segment dataset. For this dataset, all island models performed considerably worse than the single population, and the island models with windowing performed several percentage points better than those without windowing.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented an effective island model implementation of biogeography-based classification rule mining. The performance of our implementation shows that our island model implementation is able to achieve speedups ranging from approximately 4 to 9 without compromising accuracy on most datasets. We have also presented a scheme for data windowing that provides even larger speedups ranging from approximately 18 to 30, at the cost of a small loss in accuracy on some problems. We have compared the performance of the island model on several migration topologies and have observed that for our datasets the choice of migration topology does not have a major effect on classification accuracy. There does appear to be some difference between topologies with regard to speedup, but the results vary based on the problem tested and do not appear to follow any identifiable trend. Some of these differences may be explained by the stochastic nature of the algorithm, as all configurations were run with different seed values for the random number generator.

Our work also demonstrates the effectiveness of the Julia language for parallel problems; the language features allowed our algorithm to be implemented completely in a high-level language without sacrificing the performance and parallel capabilities of low-level languages such as C. Although the speedups of Julia compared to other languages vary considerably depending on the computational task, in our previous comparison of function optimization through serial BBO in Julia and MATLAB (data not shown) we observed speedups of roughly a factor of 4.

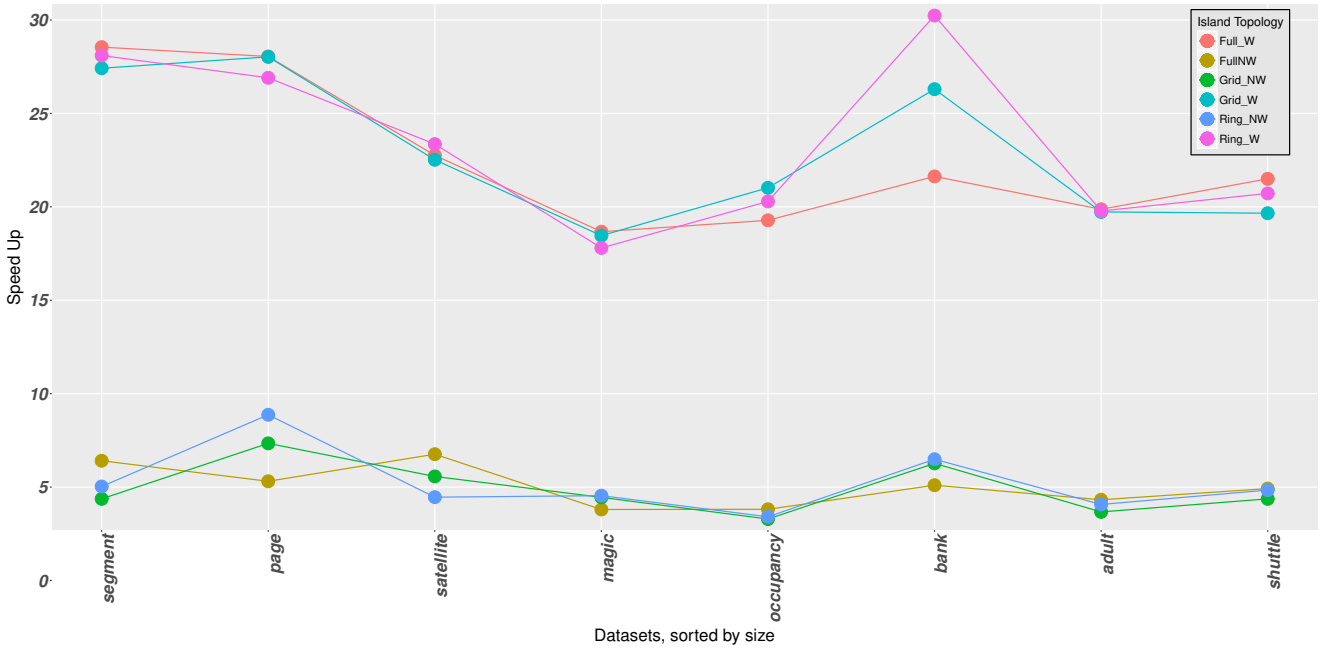

Figure 7: Speedup values for all configurations

Our results point towards several open avenues for future work. We plan to investigate the impact of heterogeneous parameter settings for elitism count, mutation rate, and rule length in the different sub-populations. Previous research has shown that such an approach can lead to greater robustness and sometimes even an increase in solution quality [15]. In addition, we plan to investigate alternative windowing schemes that do not involve data forwarding between islands. Such schemes could have the potential to further reduce execution time by reducing the communication overhead of sending training data between islands. Finally, we also plan to investigate the causes of unusual performance on certain datasets - for example, on the segment dataset, island model implementations with data windowing performed significantly better than islands models without data windowing, unlike the observed results for all other datasets. Currently, the impact of algorithm parameters and dataset characteristics on classifier performance is not well-understood.

ACKNOWLEDGMENTS

This work was supported in part by a Research Experiences for Undergraduates stipend from the North Carolina State University College of Engineering.

REFERENCES

[1] [n. d.]. Julia DistributedArrays. https://github.com/JuliaParallel/DistributedArrays.jl
[2] [n. d.]. The Julia Language. https://julialang.org
[3] Enrique Alba. 2002. Parallel evolutionary algorithms can achieve super-linear performance. *Inform. Process. Lett.* 82, 1 (2002), 7 – 13. https://doi.org/10.1016/S0020-0190(01)00281-2
[4] Enrique Alba and JosÃĪ M. Troya. 2001. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems* 17, 4 (2001), 451 – 465. https://doi.org/10.1016/S0167-739X(99)00129-6 Workshop on Bio-inspired Solutions to Parallel Computing problems.
[5] I. R. Andalon-Garcia and A. Chavoya. 2012. Performance comparison of three topologies of the island model of a parallel genetic algorithm implementation on a cluster platform. In *CONIELECOMP 2012, 22nd International Conference on Electrical Communications and Computers*. 1–6. https://doi.org/10.1109/CONIELECOMP.2012.6189871
[6] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. 2012. Julia: A Fast Dynamic Language for Technical Computing. *CoRR* abs/1209.5145 (2012). arXiv:1209.5145 http://arxiv.org/abs/1209.5145
[7] A. Bhattacharya and P. K. Chattopadhyay. 2010. Biogeography-Based Optimization for Different Economic Load Dispatch Problems. *IEEE Transactions on Power Systems* 25, 2 (May 2010), 1064–1077. https://doi.org/10.1109/TPWRS.2009.2034525
[8] C.L. Blake, E. Keogh, and C.J. Merz. [n. d.]. UCI repository of machine learning databases. www.ics.uci.edu/mlearn/MLRepository.html
[9] Erick Cantu-Paz. 1997. Designing Efficient Master-Slave Parallel Genetic Algorithms.
[10] Erick Cantú-Paz. 1998. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis* 10, 2 (1998), 141–171.
[11] Erick Cantú-Paz. 2001. Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms. *Journal of Heuristics* 7, 4 (01 Jul 2001), 311–334. https://doi.org/10.1023/A:1011375326814
[12] Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. Cambridge University Press, New York, NY, USA.
[13] Effat Farhana and Steffen Heber. 2017. Biogeography-based Rule Mining for Classification. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. ACM, New York, NY, USA, 417–424. https://doi.org/10.1145/3071178.3071221
[14] Alex A. Freitas. 2002. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
[15] Y. Gong and A. Fukunaga. 2011. Distributed island-model genetic algorithms using heterogeneous parameter settings. In *2011 IEEE Congress of Evolutionary Computation (CEC)*. 820–827. https://doi.org/10.1109/CEC.2011.5949703
[16] Wang Guan and Kwok Yip Szeto. 2013. Topological Effects on the Performance of Island Model of Parallel Genetic Algorithm. In *Advances in Computational Intelligence*, Ignacio Rojas, Gonzalo Joya, and Joan Cabestany (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 11–19.
[17] Weian Guo, Ming Chen, Lei Wang, Yanfen Mao, and Qidi Wu. 2017. A Survey of Biogeography-based Optimization. *Neural Comput. Appl.* 28, 8 (Aug. 2017), 1909–1926. https://doi.org/10.1007/s00521-016-2179-x

[18] Hisao Ishibuchi, Shingo Mihara, and Yusuke Nojima. 2013. Parallel Distributed Hybrid Fuzzy GBML Models With Rule Set Migration and Training Data Rotation. *Trans. Fuz Sys.* 21, 2 (April 2013), 355–368. https://doi.org/10.1109/TFUZZ.2012.2215331

[19] Cezary Zygmunt Janikow. 1992. *Inductive Learning of Decision Rules from Attribute-based Examples: A Knowledge-intensive Genetic Algorithm Approach.* Ph.D. Dissertation. Chapel Hill, NC, USA. UMI Order No. GAX92-07958.

[20] Jörg Lässig and Dirk Sudholt. 2010. Experimental Supplements to the Theoretical Analysis of Migration in the Island Model. In *Parallel Problem Solving from Nature, PPSN XI*, Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 224–233.

[21] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04).* Palo Alto, California.

[22] Shyh-Chang Lin, W. F. Punch, and E. D. Goodman. 1994. Coarse-grain parallel genetic algorithms: categorization and new approach. In *Proceedings of 1994 6th IEEE Symposium on Parallel and Distributed Processing.* 28–37. https://doi.org/10.1109/SPDP.1994.346184

[23] Bing Liu, Wynne Hsu, and Yiming Ma. 1998. Integrating Classification and Association Rule Mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98).* AAAI Press, 80–86. http://dl.acm.org/citation.cfm?id=3000292.3000305

[24] Yi Liu, Taghi M Khoshgoftaar, and Naeem Seliya. 2010. Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Transactions on Software Engineering* 36, 6 (2010), 852–864.

[25] Xavier Llora, Anusha Priya, and Rohit Bhargava. 2009. Observer-invariant histopathology using genetics-based machine learning. 8 (03 2009), 101–120.

[26] David Martens, Bart Baesens, Tony Van Gestel, and Jan Vanthienen. 2007. Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research* 183, 3 (2007), 1466 – 1476. https://doi.org/10.1016/j.ejor.2006.04.051

[27] Zbigniew Michalewicz. 1996. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.).* Springer-Verlag, London, UK, UK.

[28] V. K. Panchal, Parminder Singh, Navdeep Kaur, and Harish Kundra. 2009. Biogeography based Satellite Image Classification. *CoRR* abs/0912.1009 (2009). arXiv:0912.1009 http://arxiv.org/abs/0912.1009

[29] Zdzislaw Pawlak. 1998. ROUGH SET THEORY AND ITS APPLICATIONS TO DATA ANALYSIS. *Cybernetics and Systems* 29, 7 (1998), 661–688. https://doi.org/10.1080/019697298125470

[30] Seyed Habib A. Rahmati and M. Zandieh. 2012. A new biogeography-based optimization (BBO) algorithm for the flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology* 58, 9 (01 Feb 2012), 1115–1129. https://doi.org/10.1007/s00170-011-3437-9

[31] M. RuciÅĎski, D. Izzo, and F. Biscani. 2010. On the impact of the migration topology on the Island Model. *Parallel Comput.* 36, 10 (2010), 555 – 571. https://doi.org/10.1016/j.parco.2010.04.002 Parallel Architectures and Bioinspired Algorithms.

[32] Steven L. Salzberg. 1994. C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning* 16, 3 (01 Sep 1994), 235–240. https://doi.org/10.1007/BF00993309

[33] Ivan Sekaj. 2004. Robust Parallel Genetic Algorithms with Re-initialisation. In *Parallel Problem Solving from Nature - PPSN VIII*, Xin Yao, Edmund K. Burke, José A. Lozano, Jim Smith, Juan Julián Merelo-Guervós, John A. Bullinaria, Jonathan E. Rowe, Peter Tiňo, Ata Kabán, and Hans-Paul Schwefel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 411–419.

[34] Priyanka Sharma and Saroj. 2015. Discovery of Classification Rules Using Distributed Genetic Algorithm. *Procedia Computer Science* 46 (2015), 276 – 284. https://doi.org/10.1016/j.procs.2015.02.021

[35] Dan Simon. 2008. Biogeography-based optimization. *IEEE transactions on evolutionary computation* 12, 6 (2008), 702–713.

[36] KG Srinivasa, KR Venugopal, and Lalit M Patnaik. 2007. A self-adaptive migration model genetic algorithm for data mining applications. *Information Sciences* 177, 20 (2007), 4295–4313.

[37] Dirk Sudholt. 2015. Parallel evolutionary algorithms. In *Springer Handbook of Computational Intelligence.* Springer, 929–959.

[38] Sun-Chong Wang. 2003. *Artificial Neural Network.* Springer US, Boston, MA, 81–100. https://doi.org/10.1007/978-1-4615-0377-4_5

[39] Darrell Whitley. 1989. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, 116–121.

[40] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85.

[41] Darrell Whitley, Soraya Rana, and Robert B Heckendorn. 1999. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology* 7, 1 (1999), 33–47.

[42] Darrell Whitley and Timothy Starkweather. 1990. Genitor II: A distributed genetic algorithm. *Journal of Experimental & Theoretical Artificial Intelligence* 2, 3 (1990), 189–214.

[43] Man-Leung Wong, Tien-Tsin Wong, and Ka-Ling Fok. 2005. Parallel evolutionary algorithms on graphics processing unit. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 3. 2286–2293 Vol. 3. https://doi.org/10.1109/CEC.2005.1554979