Evaluating a Parallel Evolutionary Algorithm on the Chess Endgame Problem

Wayne Iba  
Westmont  
955 La Paz Road  
Santa Barbara, CA 93108  
IBA@WESTMONT.EDU

Kelsey Marshman  
Westmont  
955 La Paz Road  
Santa Barbara, CA 93108  
KMARSHMAN@WESTMONT.EDU

Benjamin Fisk  
Westmont  
955 La Paz Road  
Santa Barbara, CA 93108  
BFISK@WESTMONT.EDU

Abstract—Classifying the endgame positions in Chess can be challenging for humans and is known to be a difficult task in machine learning. An evolutionary algorithm would seem to be the ideal choice. We describe our implementation of a parallel island model and evaluate it in the context of the Chess Endgame data set from the UCI machine learning repository. We are mainly interested in impact of parallelization upon runtime and accuracy. Thus, we compare the system's performance under a number of varied conditions, including population size, number of islands, number of neighbors, migration rate, and migrant selection strategy. These results show the system to be useful from an efficiency standpoint, and point to opportunities to better understand the behavior and properties of the island-model evolutionary algorithm.

I. INTRODUCTION

Genetic and evolutionary algorithms have two especially appealing features: they have been shown to work well in domains where an adequate theory has not been developed and they can be naturally implemented on parallel hardware. This combination makes GAs particularly powerful tools. In this paper, we are primarily interested in parallelizing GAs using a standard island model [4,5]. Specifically, we explore the empirical behavior of several strategies for selecting individuals that migrate between islands. As a test bed, we use the Chess Endgame domain from the UCI repository [1]. The Chess Endgame domain is a challenging classification task in its own right, but for our purposes it need only be sufficiently complex to reveal the effects of the different migrant selection strategies and other factors we varied in our implementation of the island model.

II. ISLAND MODEL

Genetic and evolutionary algorithms have a number of advantages to their credit. Certainly, they are effective methods at solving extremely challenging problems. Although they can require extensive time to converge on an acceptable solution, they have a natural parallelization that promises to offset that characteristic. A third quality is the conceptual simplicity of the algorithm.

The standard extension of the GA to parallel hardware involves setting up multiple populations on different processors, which we may think of as islands. The standard GA is employed on each processor, evaluating individuals’ fitness, selecting survivors, breeding children, and mutating individuals. However, in addition to these regular operations, the island version selects a fraction of individuals that migrate to neighboring islands. Thus, not only must one specify the survival rate, mutation rate, population size, etc. as with the traditional GA, the user of the island model must also specify a migration rate for how many individuals leave, and a selection strategy for picking which individuals will migrate.

We implemented and tested a version of the island model on a 32-node cluster using MPI. Each node is a 2.7ghz Pentium 4 with 1 gigabyte of ram, connected to the other nodes and the front end through a gigabit switch. The cluster is running the Rocks operating system derived from RedHat Linux. For the purposes of conducting our experiments, we divided the cluster into smaller sub-clusters and implemented multiple experiments simultaneously.

III. IMPLEMENTATION

To address the questions of interest, we selected the Chess Endgame (King-Rook vs. King) domain from the UCI Machine Learning Repository [1]. In this domain, White has its King and a Rook, while Black has only its King with Black to play. Given a board state represented as the rank and file of the three pieces, a classifier must predict the number of White moves needed to checkmate Black assuming optimal play by both parties. The classifier must make predictions from 0 (already checkmate) to 16 moves, or identify that the game state leads to a draw. The practical relevance of this problem domain allows a game-search algorithm to search to the desired depth very efficiently without employing breadth-first or iterative deepening search. The depth is entirely determined by the static features of the current game state.

The data set from the UCI repository contains 28,056 instances of the 18 possible classes (0 to 16 moves plus draw). The instances are not distributed evenly across these 18 classes. Guessing the most frequent class yields 16% accuracy. Of course, our system does no analysis of this sort and starts out close to the expected accuracy of guessing randomly among the 18 classes (5.5%).
A. Representation and Classification

A significant problem in all learning problem domains involves selecting a representation that the learning method accommodates and that captures the relevant information needed to accurately classify test instances. Instances from the dataset have six features – the rank and file of the three remaining pieces – and the number of moves to checkmate. For the genetic algorithm discussed in this paper, we employ an 8-bit vector to represent a number between 1 and 8, where one of the eight bits has a value of 1 while the remaining bits are 0. We represent the rank and file of the three pieces using six of these 8-bit vectors. In addition, we augment each instance with an additional six 8-bit vectors representing the difference between the ranks and files of each pair of pieces. Thus, we represent each instance from the Chess Endgame domain as 12 8-bit segments, or 96 bits and a classification (0-16 or draw).

An important problem specific to genetic algorithms involves the representation of an individual’s genome. We decided to treat the classification task as a large conditional, or switch statement, where an instance is sequentially compared to patterns. Each pattern is associated with a distinct classification, and the first pattern that matches an instance determines the instance’s classification. Thus, our system represents an individual as a rule consisting of 17 96-bit segments (1632 bits); the 18th class is treated as an else or otherwise clause.

To see how patterns work, let us focus on a single feature such as the White-King rank represented as an 8-bit vector. If a pattern such as 01001111 is compared to an instance where the White King’s rank happens to be 2 (or any of the values 5 through 8), then the pattern is said to match. Of course, all 12 of the 8-bit feature patterns must match for the rule to issue a classification. If a given instance does not match all 12 of the first set of features, it is compared to the next set of 12, and so on, until a set matches or until it reaches the catch-all case.

B. Fitness and System Parameters

Given the above representation, we implemented a standard genetic algorithm with two-point crossover reproduction and point mutation. Based on the pattern matching scheme above, a given individual will always make a classification for any given instance. To measure fitness, we select a random subset of 180 instances from the UCI database as our data set. An individual’s fitness is evaluated by the percentage of this sample that is correctly classified.

From one generation to the next, some individuals survive and others perish. A fixed portion of the current generation is selected to survive into the next generation. Individuals survive according to a proportional fitness selection process. The lost individuals are replaced with offspring created from crossover reproduction. To this end, we select (with replacement) parents from the entire previous generation, again using proportional fitness selection. During crossover reproduction, no special treatment was given to case or feature boundaries within an individual’s genetic code. That is, crossover could take place anywhere in the 1632-bit sequence.

We implemented three strategies for selecting individuals to migrate: random, best-fit proportional, and worst-fit proportional selections. Contrary to some implementations (e.g., [3]), we transport migrants between islands rather than copy or clone them. Note, individuals are marked for migration prior to breeding; thus, a migrant may leave parts of its genetic material behind as it migrates.

Previous work has addressed the role played by the strategy for replacing existing individuals with immigrants [3]. We ignored this factor settling instead for a single fixed replacement strategy. The arriving immigrants take the place of randomly chosen offspring in the newly formed generation; that is, survivors from the previous generation remain.

IV. Experiments

For the experiments reported here, we did not vary the generational survival or mutation rates. After each generation, 20% of the population was selected for survival according to a fitness proportional scheme. Our mutation rate of $6.13 \times 10^{-6}$ per bit, yields an expected point mutation of one bit position flip for every 100 individuals. The numbers and graphs reported here reflect average fitness across the population for six repetitions of a given condition.

A. Speedup from Parallelizing the Island Model

The potential for parallelization is a significant characteristic of GAs. However, parallelizing algorithms is a notoriously hazardous process. First, parallel programs are very difficult to debug; but a potentially more important problem concerns the communication between the individual processes that make up the total system. If not for communication costs, we might expect a parallelization speedup equal to the number of processors used to run parts of the program simultaneously. However, in some cases communication costs can wipe out any advantages gained from the parallel hardware. In the case of the island model, we expect the communication costs to be low given that relatively few individuals migrate at each generation.

Our first hypothesis was that the parallel implementation should complete a given number of generations in a fraction of the time for the standard GA, where the fraction is determined by the number of islands over which the population is distributed. We ran our implementation on the Chess Endgame domain for 1000 generations under several different conditions. We evaluated two population sizes – 1600 and 9600 – both as single monolithic populations and divided over 8 islands (200 and 1200 individuals each). In addition, we also tested the larger population distributed over 24 islands of 400 individuals each. For all parallelized implementations each island had four neighbors and migrated one individual to each. Finally, as a check on the communication costs between processes, we compared the random migration and the no-migration conditions (only for the 24 island model case).
Table 1 shows the average run times (in minutes) for these conditions of the parallel and serial versions of the program for two different population sizes. In the first data column, we show runtime for a total population of 1600 distributed over 8 islands, as well as a single population of 1600 individuals. The second and third columns show data for a population of 9600, distributed over 8 islands with 1200 individuals on each and over 24 islands with 400 on each.

For all three conditions, we observe a significant speedup but do not achieve the superlinear speedup reported by others [2,3]. In each condition, we observe about 80% of the ideal speedup (i.e., a speedup factor exactly dividing the runtime by the number of processors). By comparing the runtime for the 24 island condition with the 24 island version without migration, we note that migration introduces a significant cost of approximately 33%. This will be one area where we focus attention in future work. While our hypothesized speedup was generally confirmed, we expected to achieve that speedup without a loss in accuracy; in fact, we might expect even better accuracy with the island model.

B. Benefits in Learning Rate from Island Model

In addition to the time savings through an implementation on parallel hardware, we wanted to verify that the endgame classification accuracy was at least as good as the serial version – if not better. That is, it has been reported that the island model with migration performs better than a single island having a population equal to the sum of the islands’ populations [3]. So when we compared the accuracies for the runs reported above, we might have expected to see an advantage in learning rate for the island model. Thus, we also hypothesized that migration between islands should yield an improvement in accuracy (at the run time costs and benefits observed in Table 1), and that the island version with migration should learn more rapidly than the single island runs.

To start, we checked to see if migrating individuals had any benefit at all. We compared the learning curves for the 24-island model without any migration to the same island layout with random migrant selection. In the migration case, each island sent one migrant to each neighbor every generation.

In the first experiment, we saw that a single island with a large population held an advantage over many islands with
smaller populations. Figure 2 compares the results of the eight-island condition to the 24-island model. Again, larger populations distributed over fewer islands appears to come out ahead, but the differences are much smaller and are not statistically significant.

Continuing to search for advantages of multiple islands, we considered that the number of migrants may need to be larger in order to achieve or exceed the performance of a single large population. In our next experiment, we varied the migration rate so that now a total of 16 individuals are selected to migrate, with each of the four neighbors receiving four immigrants. We kept the total population fixed at 9600 and distributed the individuals evenly over 24 islands.

Figure 3 shows our results from this experiment. The larger number of migrants appears slightly better, but again, the differences are not statistically significant. On reflection, as the number of migrants increases, the system as a whole should more closely approximate a single large population. In this light, the improvement resulting from sending four migrants per island confirms the previously observed advantage of a single large population.

Next we turned towards the topology, or connectedness, of the islands. In our model, an island has a fixed number of neighbors to which individuals may migrate. Another way to increase the mixing effect from migration is to increase the number neighboring islands. We ran our system under two more conditions where each island had two and eight neighbors. Together with the data from the earlier run, we can compare 24 islands with populations of 400 individuals under the three cases: two, four and eight neighbors; each neighbor would receive one immigrant.

Figure 4 shows the learning curves for these conditions. As might be expected the situation with less connectedness (two neighbors) did not perform as well as those with more (four and eight neighbors). Focusing on just the two and eight neighbor conditions, the difference between the learning curves appears to be significant at the 95% confidence level. However, there is no perceptible difference between the eight neighbors getting one migrant each and the four neighbors getting four migrants each.

Many factors could explain this result. In our closing discussion, we consider some of the factors and tests that we will perform in the future in order to gain a clearer understanding of the advantages of the island model. But before closing, let us turn our attention to the migrant selection strategies.

C. Impacts of Migrant Selection Strategy

Previous work has demonstrated varied benefits from alternative strategies for selecting individuals to migrate [3]. While selecting random individuals and selecting highly fit individuals have been the standard strategies, we wondered if selecting unfit individuals might have a positive impact. One explanation for the benefit from the island model in the first place has to do with the genetic diversity that the separate islands isolate. But rather than being truly isolated, the migration allows genetic information to flow between populations. When highly fit individuals migrate, it seems analogous to the situation where adjacent neighbors are giving hints in a problem solving exam. The hints might not always be needed, nor useful, but if everyone is giving their best hints we expect the collective performance on the exam to improve.

However, if the goal is to foster genetic diversity, then sending weak individuals might serve that purpose even better than migrating fit individuals. Instead of the problem-solving exam, consider the following scenario: several companies are trying to create the same product. Some designers in Company A are recommending an unpopular course of action. They become shunned, and leave for Company B, where their “contrarian” input is more welcome. The analogy to EAs is not perfect, but motivated us to evaluate the performance of this strategy as a means to get out of local optima.
Figure 5 shows the results of this experiment. We compared the selection of strong (highly fit) and weak (unfit) individuals to the performance of the random selection of migrants presented earlier. All three learning curves represent 24 islands of 400 individual populations with each island having four neighbors. As the graph indicates, there were no significant differences between the conditions. Thus, we were unable to gather supporting evidence for our third hypothesis – selecting weak individuals as migrants would provide an overall benefit.

D. Generalization and Overfitting

In all of the experiments described to this point, we used a sample of 180 chess endgame problems to evaluate the fitness of individuals. The chess endgame domain from the UCI repository contains a total of 28,056 instances. Preferably, a learning method will form a model from a small sample of a larger population, but the formed model will make accurate predictions or classifications on unseen cases. When the model accurately classifies the training data but has high error on unseen data, the model has over-fit the data.

In order to check how well our evolving individuals were generalizing to the larger set of end-games, we periodically evaluated the population with respect to the entire set of 28,056 test cases. For the purposes of this test, we continued the evolution for 4,000 generations. The population consisted of 24,000 individuals distributed over 24 islands. As before, each island had four neighbors to which one individual migrated on each generation.

Figure 6 shows the improvement in fitness for the 180 sample test cases and for the entire dataset consisting of 28,056 instances. First, we note that the larger population when continued to 4,000 generations reaches 75% accuracy, a 10% gain over our earlier results. Certainly, an ongoing concern for our work is to improve the asymptotic accuracy on the endgame dataset. But the generalization to unseen instances is surprising in several ways. First, although we expect the overall accuracy to be lower than on the sampled set, the magnitude of the difference is disappointing. But second, it is surprising that the traditional pattern with overfitting is not manifest here; rather than decreasing accuracy on the full set with further evolution, the accuracy holds steady at 25%. And third, it is surprising that over 7,000 test cases are accurately classified based on seeing only 180 training cases. Finally, it is surprising that after only 600 generations the population has nearly reached 24% and gains only a single percentage point of accuracy over the next 3,400 generations.

V. DISCUSSION

We implemented a parallel island model of the standard genetic algorithm and tested it on the Chess Endgame domain. The problem domain proved to be very challenging; under the conditions and number of generations we tested, we never achieved perfect accuracy. The approximate accuracy for all our runs out to 1000 generations centered around 65%. But note, this is significantly better than random chance given that there are 18 possible classes. (In current experiments not included here, the most-fit individual exceeds 95% accuracy with an average population fitness over 95% after 3000 generations with very large populations.)

With respect to speedup in run time, we achieved a general speedup as a result of distributing a population across island processors. We observed a significant speedup from parallelization but did not approach the superlinear speedup that others have observed. However, we found no significant differences in learning rate between the parallel island version and a single population of equivalent size. Neither did we find a significant difference between any of the migrant selection strategies we evaluated.

Several possible factors may be at work in these regards. First, the Chess Endgame domain may be too challenging to allow the benefits to be perceived. In other words, we may
be observing a ceiling effect of sorts. Second, with respect to speedup, we may be suffering from our implementation of migration between islands. Islands are synchronized by generation, and thus small variations between processors may create ripple effects that drain the benefit from parallelization. Third, we may have to find a better combination of starting parameters, such as population size, survival rate, migration rate, number of islands, number of neighbors, etc. We might also need a more powerful crossover reproduction operator. Given the size of an individual’s genome, two-point crossover may be too conservative in terms of exploring the space of individuals. Fourth, it may be the case that the learning advantage of the island model over the serial version may not apply to the Chess Endgame domain; in that case, we would want to understand the factors at work and be able to characterize domains as to their applicability in this regard. Finally, it may be the case that we already are observing real effects and we just need to run more trials in order to shrink our confidence intervals. For example, Figure 4 is highly suggestive of significant differences. In subsequent tests, we intend to explore all of these possibilities. Certainly, parallel implementations of evolutionary algorithms hold great promise, but in what situations that promise can be had is still unclear.

ACKNOWLEDGEMENTS

We gratefully acknowledge the Fletcher Jones Foundation for the generous grant that provided our 32-node Cluster computer. The authors thank Joel Stewart and Daniel Rufener for their participation during the initial implementation of the system that is presented in this paper. We also thank the originators and maintainers of the UCI Machine Learning Repository, as well as those who have contributed data sets.

REFERENCES


