Final Project

CPSC424

Jake Brawer

May 11, 2017

1 Background

For my final project I wrote compared a serial genetic algorithm (GA) to two different parallel approaches. At one level the goal of this project is simply to see which method results in the fastest convergence to the global maxima. However, this project also interests me due to it's relationship to past work I've done in "evolvability" of physical robots[1] and because I generally find evolutionary biology fascinating. Evolvability is the propensity for a system evolve adaptively [2]. Generally when people discuss this "propensity" they do it with regards with regards to a system's genotype-phenotype (G->P) map. For example, trying to evolve pure C source code (e.g. by changing and/or deleting characters randomly and evaluating the performance of mutated code) will almost certainly result in unusable very quickly. However if you tried to evolve a binary representation of this same code things might go a little differently.

In addition, there is also a sense in which evolvability is also partially determined by the system's environment, that is, by the contextual factors that constrain and impinge upon the system. In a sense this is what I am looking at with my project. As i will discuss in more detail in a subsequent section, the main difference between my parallel code and serial code is that in the serial code the genomes are free to recombine with any other genome in the population, while in the parallel code recombination is node-specific. Therefore, what I am really looking at here, all other things being equal, is

[1] Brawer, J., Hill, A., Livingston, K., Aaron, E., Bongard, J., & Long Jr, J. H. (2017). Epigenetic Operators and the Evolution of Physically Embodied Robots. Frontiers in Robotics and AI, 4, 1.

[2] Pigliucci, M. (2008). Is evolvability evolvable?. Nature Reviews Genetics, 9(1), 75-82.

how the structure of the environment affects the evolution of the system. Indeed, there is a really fascinating isomporphism here between the methods of parallelization and the biology underlying the evolutionary mechanics. That is,concepts like speciation, gene flow, and inbreeding depression –concepts part and parcel of the evolution of biological systems–organically fall out of some of the parallel GA approaches.

The implications of all this is that there are two ways to judge the performance of the algorithms here: 1) How fast they are (measured by wall time) and 2) how evolvable it is (measured by the number of generations). These two measures are likely highly correlated but definitely distinct; the parallel code, for example, may evolve the optimal solution in fewer generations, but the overhead required for parallelizing the code may make the overall run time long.

2 The Algorithm

2.1 Serial Algorithm

The serial algorithm is a pretty classic implementation of GA. The algorithm is attempting to evolve the solution the maximizes the function:

$$f(x_1, x_2, \ldots, x_n) = 1 - \sum_{k=1}^{n} \left(\sum_{i=1}^{n} (i^k + \beta) \left((\frac{x_i}{i})^k - 1 \right) \right)^2$$

The maximum of this function is attained when $\forall i, x_i = i$. The most immediate difference is that here the gene strings are comprised of double precision values rather than the typical binary string representation. I did it this way simply because I thought it was a little more interesting and challenging. However there is an accuracy tradeoff here; given the volatile nature of floating point arithmetic it is unlikely any algorithm will actually converge on the exact answer.

Here I use crossover and mutation to recombine the genes. Parents were selected using tournament selection. This involves randomly selecting k genes and comparing their fitnesses, selecting the two most fit individuals (i.e. individuals with the highest f value) out of the k. The offspring's genes are formed, in part, by concatenating contiguous regions of the parents' genes (these regions may not be of equal length, but they always create an n sized genome). Additionally each gene in the offspring's genome has `MUTATION_PROB` probability of being mutated by a value between [-1,1]. The same number of genomes is maintained across generation.

2.2 Island Hopping

The island hopping algorithm is a parallel GA mention in the *Genetic Algorithms Handout*. Conceptually the island hopping algorithm is similar to the serial algorithm but with a few key differences. Here, the population of genomes are spread amongst the nodes creating p species, where p is the number of nodes. Migration between subpopulations occurs every 10 generations, wherein each node sends there most fit genome to every other node, and replaces, replacing their least fit individuals with the newly received individuals.

Conceptually there are benefits and trade-offs to this approach. On one hand, you are splitting the computation amongst many nodes, which in theory should result in some sort of speedup. On the other hand, each node has smaller population size and thus are liable to incur the evolutionary depressive effects of inbreeding. However, the relative isolation of each node's population may result in a wider search of the phenotypic space, and thus may be less likely to converge to a local optimum.

2.3 Stepping Stone

The stepping stone algorithm is another parallel algorithm that differs from the island hopping algorithm in one key way: migration. That is, rather then a node sending its most fit genome to all other nodes, it sends it only to a single adjacent node. The benefit of the approach is that populations are a little more isolated and therefore are searching the phenotypic space a more readily. Again, though, less migration means potentially more inbreeding, which can attenuate the search process.

3 Results

3.1 Serial Algorithm

n	β	Pop Size	$\text{Total}_{\text{gens}}$	$\text{Total}_{\text{time}}$ (s)	Max fitness
4	50	40	100000	5.87	-8.97
4	0.5	40	100000	5.68	0.99
5	$1E9$	40	100000	9.87	$-4E13$
5	$1E7$	40	100000	9.91	$-2E11$

3.2 Island

n	β	procs	Pop Size	$\text{Total}_{\text{gens}}$	$\text{Total}_{\text{time}}$ (s)	Max fitness
4	50	2	40	100000	3.55	-1230
4	0.5	2	40	100000	5.68	0.85
5	$1E9$	2	40	100000	5.623	$-8E17$
5	$1E7$	2	40	100000	5.57	$-2E13$

3.3 Stepping Stone

n	β	Pop Size	Procs	$\text{Total}_{\text{gens}}$	$\text{Total}_{\text{time}}$ (s)	Max fitness
4	50	40	2	100000	3.18	0.73
4	0.5	40	2	100000	3.16	0.87
5	$1E9$	40	2	100000	5.48	$1E13$
5	$1E7$	40	2	100000	5.44	$-2E6$

3.4 Discussions

Due to issues with Omega, I did not get to run the type or breadth of tests I would have liked. Indeed, for practical reasons, I was unable to run the algorithm for longer than 100000 generations, and therefore all the algorithms were stopped prematurely. As a result, it is hard to get a good sense of how evolvable each algorithm was. However, It does appear that the stepping stone algorithm performed slightly better than the Island algorithm, fitness wise, hinting at the possibility of greater evolvability.

From a raw speed standpoint its difficult to draw any firm conclusions as well. Clearly the parallel algorithms were a little faster than the serial algorithm, but that is likely because each process had few genomes to work with than the serial algorithm. It's possible that it takes more generations for these algorithms to attain maxima, in which case this slight speed increase may not matter so much.

4 Conclusions/Extensions/Challenges

It seems like there are two general approaches to parallelizing GAs. Approach 1), which is on display here, involves parallelization in the biological sense of parallel evolutionary trajectories of multiple species. Approach 2) involves parallelizing the work done by a single evolving population. The former seems like its based more in evolutionary theoretic concepts, while the latter in high performance computing concepts. It would have been interesting to

compare algorithms from these two camps, and determine if there are any fundamental differences.

One slight oversight I made that I am realizing only now involves the initial population seed. In the parallel code, each node has a slightly different seed for their random number generator. This means that the parallel and serial alogrithms start off with slightly different initial populations. Ultimately this is not a huge deal but it is possible to reach the maxima quicker due to a "warmer start."

5 Running the Code

Make code and load modules with the following command:

```
sh setup.sh
```

To run the serial code use the following command:

```
./serial <n> <b> <pop size> <max gens>
```

Where <n> is the number of items in the input vector to f, <b> is your β value, <pop size> is the number of genomes in a given generation, and <max gens> is the maximum number of gens you would like to run the algorithm for.

Run the parallel code with the following command:‘

```
mpiexec -n <p> ./<parallel alg> <n> <b> <pop size> <max gens>
```

Here <p> is the number of processes, and <parallel alg> is either `island` or `stepping_stone`. Note: each processes will get <pop size> / <p> genomes.