Parallel Exhaustive Search vs. Evolutionary Computation in a Large Real World Network Search Space

Garnett Wilson, Simon Harding, Orland Hoeber, Rodolphe Devillers, and Wolfgang Banzhaf

***Abstract*— This work examines a novel method that provides a parallel search of a very large network space consisting of fisheries management data. The parallel search solution is capable of determining global maxima of the search space using brute force search, compared to local optima located by machine learning solutions such as evolutionary computation. The actual solutions from the best machine learning technique, called Probabilistic Adaptive Mapping Developmental Genetic Algorithm, are compared by a fisheries expert to the global maxima solutions returned by parallel search. In addition, the time required for parallel search, for both CPU and GPU-optimized solutions, are compared to those required for machine learning solutions. The GPU parallel computing solution was found to have a speedup of over 10,000x, in excess of most similar performance comparison studies in the literature. An expert found that overall the machine learning solutions produced more interesting results by locating local optima than global optima determined by parallel processing.**

I. Introduction

Evolutionary computation (EC) is a good approach for examining a very large search space by finding local optima as solutions when it is not reasonable to search the entire problem space. However, there may also be value in using EC even when brute force is made possible for a very large space, as is the case here through a novel parallel computation solution. While considering the parallelization of an EC solution, we discovered that a brute force search was also possible given the use of either GPUs or CPUs for parallel programming. Thus, instead of being restricted to looking for promising sub-optima in the space with EC or other machine learning techniques, we could also determine the global optimum of the search space with brute force in order to compare solutions. This work examines the end user utility and execution time of using a novel massively parallel processing solution to determine the global optima of a large network-based data set of fisheries catch samples in the North Atlantic, and compares it to the local optima solutions presented using a developmental GA known to provide the most useful solutions from a group including other EC algorithms and simulated annealing [1].

The remainder of this paper is organized as follows. Section 2 reviews related past work, and examines the nature of the real world problem space. Section 3 describes the both most useful EC algorithm according to an expert (called Probabilistic Adaptive Mapping Developmental Genetic Algorithm, or PAM DGA) and parallel computation. Section 4 discusses the execution performance results of the non-parallel and parallel (for both GPU and CPU) implementations, and Section 5 describes the feedback of the expert user for the different search methods. Conclusions follow in Section 6.

Garnett Wilson, Orland Hoeber, and Wolfgang Banzhaf are with the Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada (email: gwilson@mun.ca, hoeber@mun.ca, banzhaf@mun.ca).

Simon Harding is with the Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Lugano, Switzerland (email: simon@idsia.ch).

Rodolphe Devillers is with the Department of Geography, Memorial University of Newfoundland, St. John's, NL, Canada (email: rdeville@mun.ca).

II. Background and Problem Space

Many studies have examined the gain in speed that can be gleaned from using a Graphic Processing Unit (GPU) in evolutionary computation (EC), especially given a problem that involves analysis of a large amount of data [2]. There has also been previous work by Langdon on using a GPU to speed up genetic programming search of a large bioinformatics data set using thoughtful division of the search space to construct a solution [3]. Our work takes a slightly different angle than previous studies: rather than attempting to optimize EC algorithms to examine the large data set, we use a novel GPU-based solution to perform a brute force search to compare with EC algorithms in finding information interesting to experts in a real world application (fisheries management). In so doing, we examine the usefulness of located global optima over local optima to fisheries management officers. Another important aspect of the study was to have reasonable execution times for these solutions. To this end, we examine the performance of the EC algorithms against both GPU and CPU parallel computing solutions when searching a very large network space.

This work represents the latest investigations of an ongoing research project involving visual analytics research and application of EC-based search for fisheries management efforts using a very large network based on a spatiotemporal data set of annual bottom trawl survey catch data for the Atlantic cod (*Gadus morhua*). In previous work in this project, we investigated the application of a basic GA to search for large catch differences [4] and then improved performance in terms of finding interesting catch information through the use of two coevolutionary GAs in [1]. The data used in this study was collected by Fisheries and Oceans Canada for the fisheries management of the Newfoundland and Labrador, Canada region. To the authors' knowledge, this data set represents one of the largest network data set analyzed in the evolutionary computation literature. Nodes represent catch levels at a particular geographical location and an associated time span, where the mean catch over all data points in the given span of years is calculated to determine the level

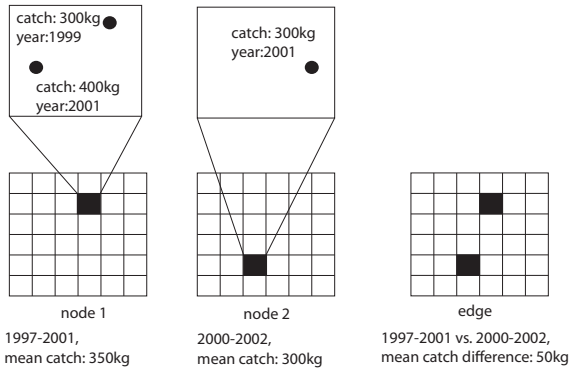

Fig. 1. Relationship between network structure and spatiotemporal visualization.

of catch for the node (see leftmost two grids of Figure 1). Edges in the network represent differences in the mean catch between locations over two time spans, and are shown in the rightmost grid of Figure 1. Edges in the network using two nodes with the same time span are excluded, since they do not reflect any difference because the entire geographical area is viewed at once in the visualization tool (application of the tool is discussed in Section 4). In virtue of not allowing the same time span (regardless of locations) in a node, loops (reflexive ties) are prohibited in the large graph.

The data collected covers an area of about 1,000,000 km^2 and a temporal range of years 1980 to 2005. The data set produces a very large network to be evaluated: The search space involves a node for every pair of locations x, y in a 30 x 30 grid and two year time span. The number of unique, unordered two year time spans for the 26 year period we examine (1980 to 2005, inclusive) is $\binom{26}{2}$, or 325 possibilities. The span of one year (e.g. 1996 to 1996) is also considered a possible time span of interest, so the number of possible time spans is thus a total of 325 + 26 = 351. Years of two time spans can overlap in each edge, but both nodes cannot refer to the same time span in a single edge. The area covered by the data set is divided as a 30 x 30 grid because it was selected as an appropriate resolution for viewing changes in preliminary experiments with an expert. Given the number of possible time spans, there will thus be 30^2 x 351 = 315,900 possible nodes to consider.

Nodes are average catch data over a particular area during a time span. We wish to consider the difference between nodes as absolute differences in those mean catches. Thus, the network to be considered consists of an undirected, weighted graph. The number of all unique edges existing in this search space is the number of possible pairings of nodes, with no time span compared to itself for a difference of 0, giving $n(n-1)/2 - t$ possibilities for n nodes and t time spans, or approximately 5.0 x 10^9 edges.

III. Search Methods

A. Probabilistic Adaptive Mapping Developmental Genetic Programming (PAM DGA)

In past work, the research project team compared four intelligent search methods: standard genetic algorithm, simulated annealing, co-evolutionary genetic algorithm and probabilistic adaptive mapping developmental genetic programming (PAM DGA). The authors found that the best performing algorithm was PAM DGA [1], considering the value of the networks to an expert end user. The PAM DGA algorithm uses two populations which evolve in parallel: genotype individuals and mapping individuals. Details and motivations behind PAM DGA are available in [5], with the algorithm discussed presently.

The fitness function used is the *modularity* (or *Q*) metric to provide a measure of the strength of connections between nodes in the network. In particular, densely connected subnetworks (communities) that are separated by sparse connections are rewarded. Newman adapted the modularity metric for weighted networks in [6], which is used here and is defined as

$$Q_w = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta_f(c_i, c_j) \qquad (1)$$

where A_{ij} is the weight of the edge from i to j, k_i of a node i in a weighted network is the sum of the weights of the edges connected to it ($k_i = \sum_j A_{ij}$), and $m = \frac{1}{2} \sum_{ij} A_{ij}$ is the total weight of the edges in the network. Q_w has an absolute value between 0 and 1, where a value of over 0.3 is typically considered to indicate favorable community division [6].

The typical community membership function is denoted as $\delta_f(c_i, c_j)$, where c_i is the community to which a node i is assigned. In the traditional community membership function, a node cannot be a member of more than one community (communities cannot overlap). In this work, the members of a community are time spans of two years. Previous work has shown that the most useful networks had no community overlap [1]. This work thus does not consider overlapped communities in the Q metric-based fitness function.

The first population consists of GA individuals. The genotype of each of these individuals is a chromosome of 20 gene sequences. Each separate gene sequence is an ordered set of 8 integers (genes) that correspond to an edge in the network. The genotype of each individual represents a graph, or list of edges. The first four integers in a gene sequence determine one node in the edge, and the last four integers determine the other node in the edge. Within each group of four integers that identify a node, the first two integers correspond to the x and y coordinates in the N x N grid and the last two integers correspond to the first and second years of a time span in the data set. Formation of genotypes and the genetic operators always ensure that the first and second years of the time span are ordered. Furthermore, in a set of eight integers that make up the gene sequence the two time spans (integers 3,4 and 7,8) cannot be the same. The absolute difference between the

average catch between the two time span and location pairs is the weight of the edge. A gene sequence corresponding to a network edge is

$$\overbrace{\overbrace{\underbrace{x_1, y_1}_{\text{location1}}\ ,\ \underbrace{t_1, t_2}_{\text{timespan1}}}^{\text{node1}}\ \overbrace{\underbrace{x_2, y_2}_{\text{location2}}\ ,\ \underbrace{t_3, t_4}_{\text{timespan2}}}^{\text{node2}}}^{\text{edge}} \tag{2}$$

where $t_2 \geq t_1$, $t_4 \geq t_3$ and $t_1, t_2 \neq t_3, t_4$. The second population in the PAM DGA algorithm is the mapping population. This second population consists of potential mappings of all time spans of two years to a community. As described in Section 2, there are 351 time spans that are each considered a community in the standard GA. A mapping individual consists of all ordered year pairings that constitute a time span. Upon initialization, each ordered pairing of years are given a randomly assigned community number from 1...351. As such, the mapping is redundant because more than one ordered pair of years (time span) can be a member of the same community and there will be 351 or less communities. By allowing the mapping of time span to community to be redundant, the GA search on the mapping population will emphasize particular sets of time spans (not necessarily sequential). That is, instead of each time span being its own community by definition, a number of collectively interesting time spans can be grouped in a community by the coevolutionary search in the mapping population.

The algorithm begins with initialization of genotype and mapping populations of size g and m, respectively. A probability table of size g x m is then created with cells initialized to $1/m$. For each round of a steady state tournament, 4 cells of the probability table are selected using roulette selection on the m axis. The selected cells correspond to four genotype/mapping pairings that are selected where the genotypes must be unique but the mappings can be chosen more than once. The pairings are evaluated for fitness, and the best two pairings are considered parents and are left unaltered. The best two pairings are also checked against the current best genotype/mapping pairing found so far in the tournament to determine if they will be identified as the new best. Once the current best genotype/mapping pairing is identified, the table cell corresponding to the two best genotype/mapping pairings is updated according to

$$P(g,m)_{new} = P(g,m)_{old} + \alpha(1 - P(g,m)_{old}) \tag{3}$$

and the other combinations in the same column are updated according to

$$P(g,m)_{new} = P(g,m)_{old} + \alpha(P(g,m)_{old}) \tag{4}$$

where g is the index of the genotype, m is the index of the mapping, α is the learning rate (corresponding to the emphasis of current table values over previous values), and $P(g,m)$ is the probability in cell [g, m] of the table. Updates by equations 3 and 4 result in all values in a column always having a sum of unity. A threshold value of γ is used to prevent premature convergence on a sub-optimal solution: Following the table update, if any cell in the probability table column corresponding to the winning genotypes exceed γ, all values in that column are then reset to $1/m$ so they sum to unity. The effect of noise addition and normalization is to effectively reset the chances of selection of all mappings with respect to the genotype handled by that table column. The last two ranked pairings are considered the children and are subject to genetic operations based on their respective associated thresholds. However, if either the genotype or mapping of the losing pairings is identified as the current best genotype or mapping found so far in the tournament, they are protected from both mutation and crossover. (The pseudocode for PAM DGA is provided in Figure 2.)

In addition to PAM DGA, the authors previously examined characteristics of the results of a GA, a standard coevolutionary GA, and simulated annealing. The pseudocode and detailed results for these algorithms are provided in [1], but they are mentioned only because we will examine their execution times in the results section to put PAM DGA and the parallel search technique (targeted for CPU and GPU) in perspective.

B. Massively Parallel Search

The machine learning algorithms described in the previous section search for local optima, or solutions in smaller areas of the search space. Since they explore the search space by moving from one possible solution to another that is not necessarily close by in the search space, the best solution may not be (but could be) the solution that maximizes the search criterion. On the other hand, if a brute force search over all possible solutions in the search space could be done, we could know the global optima of the entire space. As described in Section 3, the fisheries catch data search space appears to be so large as to be unreasonable to search in its entirety. Also, in terms of storage space alone, the entire data set would consume 593 GB based on our estimates.

By conceiving the search space in partitions and dividing the search as described in this section, we can provide a brute force search of the entire space. By so doing, we can determine the global maximum for the space. The key to performing this type of search is to place the arrays on a multicore processor sufficient to handle the parallel computation required, namely a GPU processor capable of massively parallel processing or a sufficiently fast multicore CPU processor. The search space can be envisioned as a grid of all time frames, with the time frame for each node of an edge given on either axes. Each element on the grid then corresponds to a comparison between two time frames, or comparison of two year pairs. For each year pair comparison (element on the grid), it is divided into a grid of the decided latitude-longitude resolution (in this case a resolution of 30x30). Here, there would simply be a 315,900x315,900 grid corresponding to compared timespans, each element being a 30x30 grid (which we will consider as 30 latitudinal

```
1  create size N genotype population & mapping population
2  initialize N x N probability table so each value = 1/N
3  while (tournament not done and solution not located
4    choose 4 genotype / mapping pairs (unique geno) using roulette selection
5    rank selected 4 genotype / mapping pairs
6    if new best genotype / mapping pair, replace best genotype / mapping pair
7    update probability table using Equation (3) & (4)
8    if (cell in best genotype column >= gamma)
9      set column values to 1 / N
10   for worst 2 genotype / mapping pairs
11    if (genotype != best genotype in best genotype / mapping pair)
12      replace genotype with parent, apply mutation & crossover
13     if (mapping != best genotype in best genotype / mapping pair)
14       replace mapping with parent, apply mutation & crossover
```

Fig. 2. Pseudocode for PAM DGA algorithm.

and 30 longitudinal divisions) for a geographic area. This conception of the search space results in duplication of all edge weight values (except for time spans compared to themselves). This repetition, as we will see, does not have a computational cost for parallel processing: Each piece of data provided to the GPU (or CPU) has unique elements not having been previously calculated (and those having been calculated previously just happen to be processed along with them). Furthermore, any conception of the search space that was attempted in order to eliminate the repetition prior to parallel computing far outweighed any benefit due to the CPU-side cost in the rearranging.

By altering the natural conception of the space, we can massage it into a form that is amenable to parallel (or massively parallel) processing shown in Figure 3. To do this, we first consider how large a section of the search space can be handled by the parallel processor at one time. In particular, the documentation for the Microsoft Research tool [7] indicates that most GPUs are limited to processing of 2-dimensional arrays of 64,000,000 elements, with no side over 8000 elements long. We found these restrictions to hold when using Accelerator with our chosen production GPU. The area of the search space sent to the GPU for processing each time was divided to be within this range. We chose to send arrays of no more than 5280x5280 elements (totaling 27,878,400 individual elements) in a 2-dimensional grid at one time. The smallest array sent is 5250 x 5250, thus containing 27,562,500 elements. (The grid of the largest size is in the lower right corner of Figure 3, with the other grids being slightly smaller.)

To produce a natural division of the search space that provides grids of this size, we consider that each element of the search space is actually an edge. Each edge is represented by two nodes, each having latitude, longitude, and time span attributes that collectively identify the node. Let us call each of these attributes *latitude1*, *longitude1*, and *timespan1* for the first node of the edge, and *latitude2*, *longitude2*, and *timespan2* for the second node of the edge. First we, divide the entire search space so we consider every instance of *latitude1* as a separate space, and then every instance of the space corresponding to each *latitude1*, *longitude1*

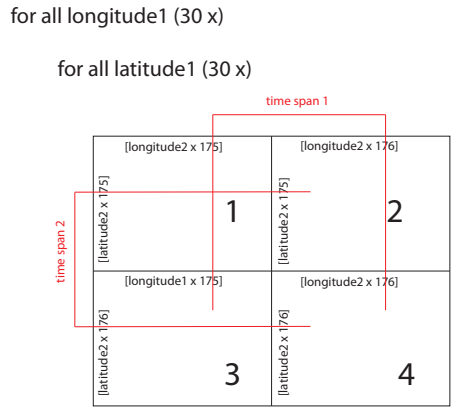

Fig. 3. Parallel computation conception of the search space.

combination as a space. This creates a nested loop with 900 iterations given 30 *latitude1* points and 30 *longitude1* points (for a 30 x 30 grid) to process four search spaces that are passed to the GPU, each of up to the stated size of 27,878,400 elements. The 351 possible time spans on each axis are split into two parts of 175 and 176. These four search spaces collectively contain nodes covering all values of *latitude2*, *longitude2*, and *timespan1* and *timespan2*. The attributes of the node corresponding to each element are also labeled in Figure 3.

The Accelerator tool (Version 2) code is used for the GPU parallel computing solution using Direct X under Accelerator bindings with C#. As mentioned at the start of this section, the size of the network to be considered is so large that it is not feasible to store all edges and access them as needed during execution. To eliminate this problem, we generate the edges as needed using the code in Figure 4 given information about the 315,900 possible nodes (the only information stored prior to determination of global optima).

In Figure 4, for each *latitude1* and *longitude1* (outer for loops) on lines 4 and 5, all elements in four smaller arrays (using inner for loop) are filled with catch weight differences with respect to *latitude2* (and *timespan1*) and *longitude2* (and *timespan2*) using weight difference calculated from the existing node data (lines 7-8 for GPU solution).

```
1  int maxLatitude = 30; int maxLongitude = 30;
3  // input node data in 4 vectors: weightDiffs1_1, 1_3, 2_1, 2_2
4  for (int longit1 = 0; longit1 < maxLongitude; longit1++)
5    for (int latit1 = 0; latit1 < maxLatitude; latit1++)
6      if using GPU
7        fill weightDiffs1_1 & weightDiffs1_2 vectors for longit2, time span 1
8        fill weightDiffs2_1 & weightDiffs2_2 vectors for latit2, time span 2
9        on GPU: determine weight difference for all grid points (see Figure 7)
10     if using CPU
11       fill input1 and input2 arrays for longit2, time span 1
12       fill input3 and input4 arrays for latit2, time span 2
13       for each input array start a thread (4 threads total)
14         in each thread determine vertical, horizontal maxes
15 find max across arrays 1,2 and 3,4 (rows) and 1,3 and 2,4 (columns)
16 for maximum values for all rows
17   find corresponding longit2, time span 1
18 for maximum values for all columns
19   find corresponding latit2, time span 2
```

Fig. 4. CPU-side code for GPU-based parallel solution.

```
1  // replicate weight vectors across rows, columns
2  FPA input1Vert = new FPA(weightDiffs1_1[longit1, latit1]);
3  FPA input2Vert = new FPA(weightDiffs1_2[longit1, latit1]);
4  FPA input3Vert = new FPA(weightDiffs1_1[longit1, latit1]);
5  FPA input4Vert = new FPA(weightDiffs1_2[longit1, latit1]);
6  FPA input1VertStretched = PA.Replicate(input1Vert, dim1);
7  repeat 6 for 3 other grids
8  FPA input1Hor = new FPA(weightDiffs2_1[longit1, latit1]);
9  FPA input2Hor = new FPA(weightDiffs2_1[longit1, latit1]);
10 FPA input3Hor = new FPA(weightDiffs2_2[longit1, latit1]);
11 FPA input4Hor = new FPA(weightDiffs2_2[longit1, latit1]);
12 FPA input1HorStretched = PA.Replicate(PA.Transpose(input1Hor), dim1);
13 repeat 12 for 3 other grids
15 // determine absolute difference between weight differences
16 FPA fpInput1 = PA.Abs(PA.Subtract(input1VertStretched, input1HorStretched));
17 repeat 16 for 3 other grids
19 // determine max in each row and column for [longit1, latit1].
20 FPA fpOutputVert1 = PA.MaxVal(fpInput1, 1);
21 repeat 20 for 3 other grids
22 FPA horFpInput1 = PA.Transpose(fpInput1);
23 repeat 22 for 3 other grids
24 FPA fpOutputHor1 = PA.MaxVal(horFpInput1, 1);
25 repeat 24 for 3 other grids
26 // move result from GPU back to CPU for further processing
27 maxesOfInputArrayVert1s[longit1, latit1] = evalTarget.ToArray1D(fpOutputVert1);
28 repeat 27 for 3 other grids
29 maxesOfInputArrayHor1s[longit1, latit1] = evalTarget.ToArray1D(fpOutputHor1);
30 repeat 29 for 3 other grids
```

Fig. 5. GPU-side code for GPU-based parallel solution.

These weight differences are appropriately separated into two vectors (*weightDiffs1_1*, *weightDiffs1_2*) for *latitude2*, *timespan1* and two vectors (*weightDiffs2_1*, *weightDiffs2_2*) for *longitude2*, *timespan2*. These vectors are then passed to GPU for processing, see Figure 5. For GPU processing, the vectors are first converted to vertical and horizontal floating point arrays (FPAs), an Accelerator-specific data type (lines 2-6 and lines 8-11, respectively, of Figure 5). These vectors are then replicated across the appropriate number of dimensions for each of the four sections of the grid (lines 6-7 and 12-13). Following that, the absolute difference in vertical and horizontal values for each of the four grids are determined (line 16-17). The result of this operation for each element is a catch difference with respect to two time spans, each with associated latitude and longitude locations. These replication and subtraction steps are shown in Figure 6.

Following the determination of the catch difference between two of {timespan, latitude, longitude} triples, the maximum of each row and column is determined. For each of the four search space sections, the maximum for each *latitude2*, *timespan1* with respect to each *longitude2*, *timespan2* possibility is determined (and vice versa) in lines 19-25. Collectively, recall that these four search spaces are in the context of a particular {*longitude1*, *latitude1*} location

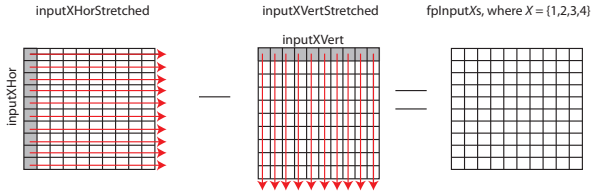

Fig. 6. Replication and subtraction of node-based weight values.

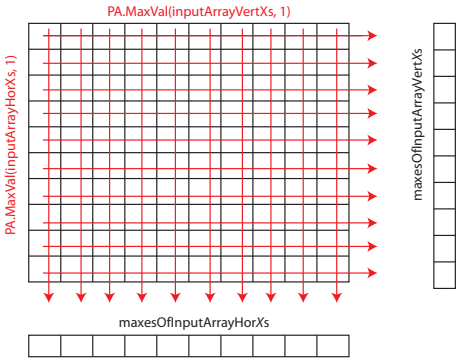

Fig. 7. Calculation of maximums across all rows and columns.

(see Figure 3). The maximum across each row and column and finally placed in the arrays *maxesOfInputArrayVertXs* and *maxesOfInputArrayHorXs* where $X = \{1, 2, 3, 4\}$, respectively, in lines 26-30 of Figure 5 and passed back to the CPU for further processing. This step is shown pictorially in Figure 7. Following the determination of the maximums, the maximum for each pair of row and column maxes in the four-array space is determined CPU-side (line 15 of Figure 4).

We should note that when determining the maxima that information is lost during this process. In particular, the maximum with respect to each row is stored in *maxesOfInputArrayVertXs*, but the column location of that maximum is lost (and vice versa for columns). However, this information can be easily determined since we need only search the original vertical and horizontal weight vectors for a difference that, combined with the difference in the location of the maximum resulting from *maxesOfInputArrayHorXs* or *maxesOfInputArrayVertXs*, respectively, produces the maximum in question. In particular, *weightDiffs1_1*, *weightDiffs1_2* are searched for *maxesOfInputArrayVertXs* and *weightDiffs2_1*, *weightDiffs2_2* are searched for *maxesOfInputArrayHorXs*. This final gathering for the time spans and locations corresponds to lines 16-19 in the CPU-side code of Figure 4.

The CPU solution operates in much the same way as the GPU solution, and only differs in lines 11-14 as shown in Figure 4 where the four arrays *fpInputX* are calculated CPU-side (lines 11-12) in the straightforward way, rather than by replication of vertical and horizontal vectors (lines 6-7 and 12-13 of Figure 5). These maxima are then calculated by dedicating each of four threads to determining the maxima for the four arrays (lines 13-14 of Figure 4). The remainder of the code operates in the same way CPU-side for CPU and GPU (lines 15-19 of Figure 4).

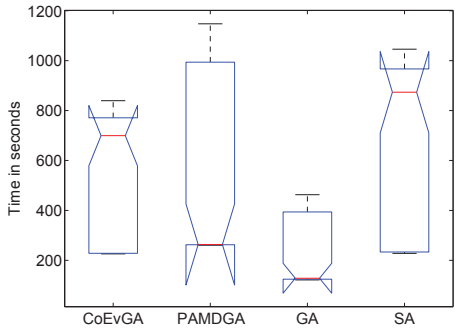

Fig. 8. Time in seconds for a solution to be located for each machine learning algorithm. Based on 50 trials.

IV. Results

A. Execution Time

For all of the machine learning algorithms, the same number of tournament rounds were completed to allow comparison of execution times. In particular, each of the EC algorithms is run for 100,000 rounds with 4 individuals being evaluated per round. This means that 400,000 individual evaluations are conducted per round. For equivalent computational effort, the SA is run for 400,000 evaluations. Figure 8 compares the time to solution for all the computational intelligence search algorithms in a boxplot based on 50 trials. Bottom, middle, and top of boxes indicate lower quartile, median, and upper quartile values, respectively. If notches of boxes do not overlap, medians of the two sets of data differ at the 0.95 confidence interval. The symbol '+' denotes points from 1.5 to 3 times the interquartile range, and 'o' denotes points outside 3 times the interquartile range.

The time for the search of the entire space using parallel computation for all global optima is shown in Figure 9 based on 50 trials. The time taken for evaluation of the four grids depicted in Figure 3 is timed. In particular, this corresponds to the execution time for the seeding of vectors and arrays for GPU and CPU, respectively, and the parallel processing associated with the GPU processing and CPU threading. The operations timed in Figure 7 is shown in Figure 4 on lines 6-9 for GPU and lines 10-14 for CPU. These sections of code were chosen to fairly represent the execution time differences because different CPU-side data preparation times are required prior to either GPU processing or threading on CPU due to the use of vectors and arrays, respectively. The CPU used for the time trials was an Intel® Core™ i7 870 @2.93GHz (which has four CPU cores) on the 64-bit Windows 7 Ultimate OS with 8GB of RAM. The graphics card used is an Asus® ENGTS450 DirectCU OC 850 MHz (overclocked) with 1GB GDDR5 video memory and a nVidia® GeForce™ GTS 450 GPU on board (which uses 192 cores). Both processors are considered current hardware at the time of this writing.

We can see from Figure 8 that, for the computational intelligence techniques examined, the time to solution after all rounds was 100 to 1200 seconds. Also, the PAM DGA al-

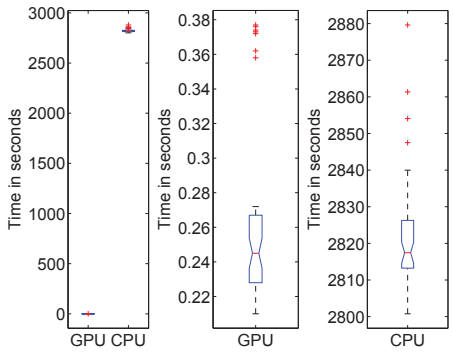

Fig. 9. Time in seconds for a solution to be located for parallel computation on GPU and CPU. Leftmost graph shows both plots, middle and right graphs are zoomed plots. Based on 50 trials.

gorithm that found the best solutions according to the expert, stretched across those execution times. In Figure 9, we can see that there is a large discrepancy between GPU and CPU execution times. The median time for execution of the code for the GPU solution was 0.245 seconds, compared to 2817.4 seconds for the multithreaded CPU solution, representing a speedup of 11,500x. This is a very significant speedup even considering GPU vs CPU claims in the literature that typically span from 10x to 1000x, and these often use a single thread CPU example [8]. This speedup is thus an order of magnitude greater than some of the higher claims, and this may be a result of compounding the normally cited speedups with the very large data set processed. Another possibility is that the speedup is due to efficient implementation of the GPU-based solution compared to the thread-based version using CPU. However, it is worth noting that comparable speedups of up to 7300x have been reported, also in a study using Accelerator, by Harding [9].

B. Expert Assessment

A fisheries analysis expert, examined the best solutions located by all machine learning algorithms (GA, SA, Co-evolutionary GA, and PAMDGA) and the global maximum search results. He rated each edge of the best networks, where these edges were visualized using GTdiff software (a visualization tool designed for this project, described in detail in [10]) as two temporal bins and one corresponding difference graph. In previous work, we investigated large changes of interest that resulted from PAM DGA [1]; here we contrast those previous best results according to the expert with the actual largest catch changes which can now be located thanks to parallel computing search. The first two grids show average catch in kg in each spatial grid element, and are ordered sequentially based on last year of the time spans (and if the last year is the same, they are ordered by the first year). The color scale spans from light yellow (lowest average catch) to brown (largest average catch). The third grid is the difference graph, which shows the difference in average catch between the two time spans as a positive (green) or negative (red) change. White indicates no change in catch, and the degree of saturation of green and red is used

TABLE I
RANKING OF DIFFERENCE GRAPHS

	No	Relevant	Salient	Differences
GA	10	6	1	17
SA	3	7	0	10
CoEvGA	6	6	1	13
PAMDGA	6	7	3	16
Parallel	13	6	1	20

to represent positive and negative differences, respectively. The final networks are shown as a 10 x 10 grid in GTdiff to allow for easier viewing of trends by the expert and in publication (as opposed to finer resolution on a large screen).

During the time period examined (1980 to 2005), there are established anomalous changes that are represented as large differences in catch over time that would be known to experts. A major event reported by biologists involved cod population levels that dropped suddenly in the early 1990s, leading to a moratorium on cod fisheries from 1992 to 1993. Other less significant changes are also known to experts during this time period. The three options for the rating of each difference graph by the expert were: No (meaning no difference relevant to fisheries officers appeared), Relevant (a difference relevant to fisheries officers appeared), or Salient (a special case of Relevant indicating that an important biological event was identified). The ranking of the difference graphs are shown in Table 1 for responses corresponding to each valued edge (actual difference) for best solutions from all computational intelligence algorithms [1] over 50 trials and for the global maximum search (multiple trials not applicable). The top difference graphs for the PAM DGA search are shown in Figure 10, and indicate known biological events including those involving the moratorium (these differences were reported with additional details in [1]).

The global maxima found by the parallel computation methods ranged in catch difference from 867,300 kg to 1,250kg between two time spans, and 115 unique maxima were determined. In order to provide the expert with a reasonable number of difference graphs to evaluate, we selected 20 of the maxima generated by the parallel algorithm for ranking by the expert (a convenient number close to the 16 evaluated for PAM DGA to provide results seen in Figure 10). We chose the top twenty based on a ratio of catch over the time span separating the last year of the first time span and the last year of the second time span. Interestingly, the results used time spans that involved comparison of a time span including the first year (1980) of the data to a year in the future, with the exception of 3 out of the top 20 results. The expert ranked these results, and found only one salient catch difference despite these difference graphs featuring the largest catch changes. This salient difference selected from the parallel search results is shown in Figure 11. He also found that, overall, the patterns were less interesting than those found by the computational intelligence algorithms that selected local optima. In particular, the graphs focused on the 1980s when larger catches occurred, but did not readily

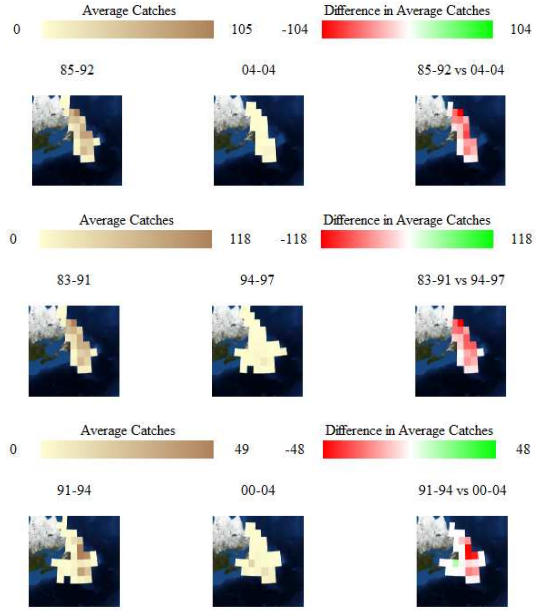

Fig. 10. Salient difference graphs selected by expert from the highest Q network produced by PAMDGA. Catches are shown in thousands of kg.

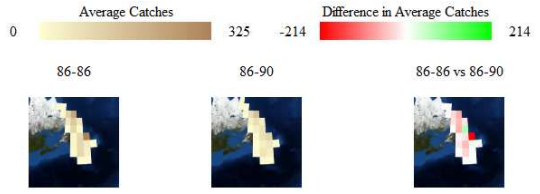

Fig. 11. Salient difference graphs selected by expert from the networks produced by parallel search. Catches are shown in thousands of kg.

pick out contrasts before and after the moratorium. The parallel search also located changes that involved overlapping time spans, which were deemed to be less helpful than the non-overlapping spans of the computational intelligence-based searches. What was interesting for the expert were not necessarily the maximum catch differences in the data set; and the expert noted that some extreme catches can be recorded but be isolated (not a pattern of interest).

V. Conclusions

This work presented a novel parallel computing solution for finding global optima in a very large network space. Furthermore, the location of large differences in this data set had real world implications, as the data represented scientific cod fisheries catch data used by fisheries officers for natural resources management purposes. The novel parallel approach was optimized for CPU and GPU use, taking advantage of the parallel computing capacity of both processor types. For the problem conception for this particular large data set, we found that the use of the GPU could provide a speedup of 11,500 times that of the CPU. This is an impressive speedup, even by GPU performance literature standards, and may be a result of usual speedup levels compounded by a larger data set. While a useful solution to parallel processing of a large data set is presented, the fisheries expert in this project found that there was greater value in finding local optima using evolutionary computation for this particular data set. In this study, the global optima tended to focus on a time period of abundant catches that were (for the most part) of less interest than the machine learning results. The value to users of local optima over global optima may differ from data set to data set, so there is likely a lot of opportunity to mimic the parallel GPU solution we present to evaluate other large network spaces quickly. Our study, however, also provides a real world example of how the exploration of a search space using evolutionary computation can be more valuable than simply finding global optima by brute force even when that option is possible.

Acknowledgment

The authors wish to thank Fisheries and Oceans Canada for making available the data used in the case study. This work was supported by a Strategic Projects Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) held by the last three authors.

References

[1] G. C. Wilson, S. Harding, O. Hoeber, R. Devillers, and W. Banzhaf, “Large network analysis for fisheries management using coevolutionary genetic algorithms,” in *GECCO*, 2011, pp. 1619–1626.

[2] W. Banzhaf, S. Harding, W. B. Langdon, and G. Wilson, “Accelerating genetic programming through graphics processing units,” in *Genetic Programming Theory and Practice VI*, 2009, pp. 1–19.

[3] W. Langdon, “Large scale bioinformatics data mining with parallel genetic programming on graphics processing units,” in *Parallel and Distributed Computational Intelligence*, ser. Studies in Computational Intelligence, F. de Vega and E. Cant-Paz, Eds. Springer Berlin / Heidelberg, 2010, vol. 269, pp. 113–141.

[4] G. Wilson, S. Harding, O. Hoeber, R. Devillers, and W. Banzhaf, “Detecting anomalies in spatiotemporal data using genetic algorithms with fuzzy community membership,” in *ISDA 2010: 10th International Conference on Intelligent Systems Design and Applications*. Chennai, India: Research Publishing Services, 2010, pp. 97–102.

[5] G. Wilson and M. Heywood, “Introducing probabilistic adaptive mapping developmental genetic programming with redundant mappings,” *Genetic Programming and Evolvable Machines*, vol. 8, pp. 187–220, 2007.

[6] M. E. J. Newman, “Analysis of weighted networks,” *Phys. Rev. E*, vol. 70, no. 5, p. 056131, Nov 2004.

[7] “An introduction to Microsoft Accelerator v2,” Microsoft Research, Tech. Rep. 2.1, June 2010.

[8] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, “Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU,” in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA ’10. New York, NY, USA: ACM, 2010, pp. 451–460.

[9] S. Harding and W. Banzhaf, “Fast genetic programming on GPUs,” in *Genetic Programming*, 2007, pp. 90–101.

[10] O. Hoeber, G. Wilson, S. Harding, R. Enguehard, and R. Devillers, “Exploring geo-temporal differences using GTdiff,” *IEEE Pacific Visualization Symposium*, pp. 139–146, 2011.