A Modified Compact Genetic Algorithm for the Intrinsic Evolution of Continuous Time Recurrent Neural Networks

John C. Gallagher
Department of Computer Science and Engineering
Wright State University, Dayton OH 45435-0001
jgallagh@cs.wright.edu

Saranyan Vigraham
Department of Computer Science and Engineering
Wright State University, Dayton OH 45435-0001
svigraha@cs.wright.edu

Abstract

In the past, we have extrinsically evolved continuous time recurrent neural networks (CTRNNs) to control physical processes. Currently, we are seeking to create intrinsic CTRNN devices that combine a hardware genetic algorithm engine on the same chip with reconfigurable analog VLSI neurons. A necessary step in this process is to identify a genetic algorithm that is both amenable to hardware implementation and is sufficiently powerful to effectively search CTRNN spaces. In this paper, we will propose and test several variations of the compact genetic algorithm (CGA) for searching these spaces. We will then benchmark the best variant using the De Jong functions, outline a hardware implementation, and discuss future plans to develop an integrated evolvable hardware device controller.

1. INTRODUCTION

The author has proposed the use of Continuous Time Recurrent Neural Networks (CTRNNs) as an enabling paradigm for evolving analog electrical circuits. In previous work we focused almost exclusively on extrinsically evolved CTRNNs that were created in simulation and only later implemented in hardware. We are currently interested in producing intrinsic CTRNN devices that evolve online as they are controlling physical processes. We feel that a necessary step in achieving this goal is to combine reconfigurable analog CTRNNs and a hardware evolutionary search engine onto a single VLSI device. Half of the problem, the feasibility of implementing CTRNNs in analog VLSI, has been addressed elsewhere [Gallagher and Fiore, 2000]. This paper will address the feasibility of searching CTRNN spaces with hardware based genetic algorithm that can be fabricated on the same chip with reconfigurable CTRNN hardware.

Several hardware based evolutionary algorithms (EAs) have been proposed in recent years [Kajitani, T., et.al., 1998] [Scott and Seth, 1995] [Yoshida and Yasuoka., 1999] One in particular, the Compact Genetic Algorithm (CGA) [Harik, Lobo, and Goldberg 1999] is especially well suited for efficient hardware implementation using common VLSI techniques [Aporntewan and Chongstitvatana, 2001]. The Compact Genetic algorithm, however, is a very weak evolutionary algorithm -- only equivalent to a first-order simple GA with uniform crossover and tournament selection [Harik, Lobo, and Goldberg 1999]. Although it is well suited to efficient hardware implementation, the standard CGA is not powerful enough to effectively evolve practical CTRNN device controllers. This paper will propose several simple variations of the CGA designed to increase the effectiveness of the search with respect to CTRNN devices. We will test the performance of the standard CGA and our variants against a benchmark locomotion control problem and show that, with simple modifications that do not significantly complicate hardware implementation, we can effectively evolve effective CTRNN control devices. Further, we will demonstrate that the performance of our modified CGA is, for this problem, superior to the simple genetic algorithms we have employed in the past.

2. THE BENCHMARK PROBLEM

Our benchmark problem is the control of legged locomotion in a simple, single-legged artificial agent. For the agent to walk, the leg must alternate swing and stance phases. A swing begins with the leg in its full backward position (at negative leg angle range limit) and the foot raised in the air. Then the leg rotates clockwise to swing the foot forward. A stance begins by placing the foot of a fully forward leg on the ground. Then the leg rotates counterclockwise to propel the body forward. Figure 1 shows the agent at the beginning of a stance phase. The leg contains three effectors and one sensor that returns the leg's angular position in radians. One effector governs the state of the foot (FT) and the other

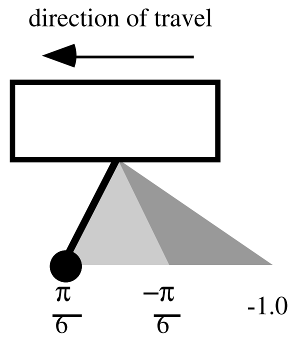

Figure 1: The Single Legged Agent

two generate clockwise (BS) and counter-clockwise torques (FS) about the leg's single joint with the body. The torques about each joint are summed, and depending on the state of the foot will either translate the body forward (foot down) or rotate the leg about its joint (foot up). Each leg has a limited range of angular motion ($\pm\pi/6$ radians - corresponds to the light gray wedge in Figure 1). A supporting leg may stretch outside of this range, but provides no transitional forces when doing so. The leg may not stretch outside an absolute limit (± 1 radians - corresponds to the dark gray wedge in Figure 1). This behavior is intended to model the reduced ability of a hyper-extended leg to provide propulsion. When the leg is lifted from the ground, the agent "falls" and its velocity is immediately set to zero.

The agent's behavior is controlled by a fully connected continuous time recurrent neural network (CTRNN) [Beer, 1995] with the following state equation:

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_i + \theta_i)$$

where y is the state of each neuron, τ is its time constant, w_{ji} is the strength of the connection from the j^{th} to the i^{th} neuron, θ is a bias term, and $\sigma(x) = 1/(1+e^{-x})$ is the standard logistic activation function. States were initialized to uniform random numbers in the range ± 0.1 and circuits were integrated using the forward Euler method with an integration step size of 0.1. In each evolved CTRNN, three units are motor neurons and provide control efforts to the forward swing (FS) and backward swing (BS) and FOOT (also called FT). The remaining neurons in each leg controller have no pre-specified role. For the experiments reported in this paper, the controlling CTRNNs receive no sensory input from the outside world.

3. COMPACT GENETIC ALGORITHM

In a compact genetic algorithm [Harik, Lobo, and Goldberg, 1999], the population is represented by a probability vector that codes the chance that each bit in an individual will be a one or a zero. Each generation is a single tournament between two individuals randomly generated from the global probability vector. The probabilities governing each bit are adjusted according to the result of the tournament. Tournaments are run until the probability vector converges. The basic CGA can be summarized as follows:

```
1. Initialize probability vector
   for i:=1 to l do p[i]=0.5

2. Generate two individuals from the vector
   a := generate(p);
   b := generate(p);

3. Let them compete
   winner, loser := evaluate(a,b);

4. Update the probability vector toward the better
   one
   for i := 1 to l do
      if winner[i] <> loser[i] then
         if winner[i] = 1 then p[i] +=1/n
         else p[1] -= 1/n

5. Check if the probability vector has converged
   for i = 1 to l  do
      if p[i] > 0 and p[i] < 1 then
            goto step 2

6. P represents the final solution
```

In the above description, `l` represents the number of bits in the genome and `n` represents the size of the simulated population. In this paper, we will systematically consider two modifications to the basic CGA. These are:

a) Elitism
We will implement elitism (the best individual seen to date stays in the population) by modifying step 2 of the basic CGA so that only the "losing" contestant of each tournament is randomly generated at the beginning of the next tournament. This ensures the best individual seen to date remains in consideration. We may more formally state this modification by rewriting step 2 of the standard CGA as follows:

2) Generate one individual from the vector

```
if fitness(a) > fitness(b) then
      b = generate(p);
else
      a = generate(p);
```

b) Mutation
We will implement mutation by adding a secondary tournament to each cycle that compares the performance of the current elite string with a mutated version of itself. If the mutated version wins, it replaces the old champion as the elite string for the next tournament. The bit positions that changed also have their probabilities returned to 0.5, or some other user selected value. Our implementation of mutation presumes elitism. We can more formally describe this modification by altering step 2 as described above and adding a new step as follows:

4.5) Mutate Champ and Evaluate

```
if fitness(a) > fitness(b)
        { c = mutate(a);
          evaluate(c);
          if fitness(c)>fitness(a) then
                { a = c;
                  prob_fix(p);
                }
        }
else    { c = mutate(b);
          evaluate(c);
          if fitness(c) > fitness(b) then
                { b = c;
                  prob_fix(p);
                }
        }
```

The `prob_fix()` routine resets `p[x]` to 0.5 for every position `x` that was mutated. Heuristically, this is meant to represent the fact that a mutation in that position seems like a good idea, but that we don't want to commit to it completely. Rather, we want to remain undecided (there's a fifty percent chance of either bit setting occurring) and let a history of evaluations determine which way that bit position should be set.

It should be noted that with the introduction of mutation, CGA convergence can no longer be guaranteed. One must modify the end condition appropriately, perhaps by setting a maximum number of tournaments to be run.

In later sections of this paper, we will refer to the standard CGA simply as "CGA". We will refer to a CGA with the elitism modification as "eCGA". We will refer to a CGA with both the elitism and mutation modifications as the "modified CGA", or the "mCGA".

Search Type	*Yield*	*Avg. Perform*
CGA	0%	0%
ECGA	33.7%	89.0%
MCGA	71.0%	92.9%

Table 1: Relative Performances of CGA Variants
Yield shows the percentage of runs that resulted in a CTRNN controller that was at least 80% of optimal. Avg. Performance shows the mean of the performances of all controllers that achieved better than 80% of optimal. Note that both the yield and relative quality is greater for mCGA than for eCGA. Also note that the standard CGA is totally ineffective for this problem.

4. PRELIMINARY COMPARISON OF CGA VARIATIONS

Our first set of experiments was designed to compare the relative efficacy of the three variations on the CGA described above. For these experiments, the parameter settings of a five neuron, fully connected CTRNN were encoded on a bit string genome using eight bits per parameter. Values were encoded as fixed-point binary numbers and were simply concatenated into a single string. For five neuron CTRNNs, there were 40 parameters resulting in a bit string of length 320. The fitness of a particular CTRNN was the amount of distance it caused the agent to walk in a fixed amount of time. No special scaling was applied to the fitness. One hundred three (103) searches each were run using CGA, eCGA, and mCGA. We found that the CGA failed terribly. Not one of the 103 runs of the CGA produced an agent capable of locomotion. Approximately 34% of the runs of eCGA produced agents capable of walking at a speed of at least 80% of optimal. 71% of the runs of mCGA produced agents capable of walking at a speed of at least 80% optimal. These results are summarized in Table 1. For this benchmark CTRNN search problem, mCGA is clearly superior. Further, mCGA compares very well to the simple genetic algorithm. Previous results using the simple GA for the same problem produced a yield of approximately 75% and an average performance of 91.1% of optimal.

5. mCGA and the CTRNN Benchmark

The preliminary benchmark results of the last section suggest that, of the CGA variants discussed, mCGA is the best suited to searching CTRNN spaces. Further, those benchmarks suggest that mCGA is, for searching CTRNN spaces, at least as effective as the standard simple genetic algorithm. Our second set of experiments was aimed at producing a

Arch Set	*Parameters*	*Bit Length*	*mCGA Yield*	*sGA Yield*	*mCGA Avg*	*sGA Avg*
CPG3	18	144	38.8%	40.4%	88.8%	89.1%
CPG4	28	224	61.2%	66.3%	90.3%	91.9%
CPG5	40	320	71.0%	74.5%	92.9%	93.2%
CPG6	54	432	79.0%	65.4%	92.2%	93.0%
CPG7	70	560	84.5%	63.3%	92.6%	92.0%
CPG8	88	704	87.9%	64.7%	94.4%	89.9%
CPG9	108	864	91.3%	57.0%	93.2%	88.9%
CPG10	130	1040	87.3%	59.8%	93.8%	88.8%

Table 2: mCGA vs. Simple Genetic Algorithm for Varying Search Space Sizes
Yield and avg. performances are as defined in Table 1. mCGA refers to the CGA with mutation and elitism. sGA refers to a standard simple genetic algorithm. Mann-Whitney tests show no significant differences in either yield or average performance up through CPG5. After CPG5, both yields and average performances drop off sharply and significantly for the standard genetic algorithm, while both yield and average performance hold steady for mCGA at least up through CTRNNs of nine neurons.

more detailed picture of the efficacy of mCGA in searching CTRNN spaces.

Approximately 100 GA searches were run over each of eight architecture sets. The base architectures searched were 3, 4, 5, 6, 7, 8, 9, and 10 neuron fully-connected CTRNNs. The purpose of these tests as to examine how well mCGA scaled to more difficult searches. The results of these experiments, as well as the results of previously reported applications of the simple genetic algorithm to the same problems [Gallagher, 1998], are shown in Table 2. mCGA runs were limited to a maximum of 100,000 tournaments to ensure that approximately the same number of candidate evaluations were made in both mCGA and simple genetic algorithm experiments. In terms of either the quality of solutions or yield of successful solutions, there is no significant difference between the simple GA and mCGA for small CTRNNs. However, there is a significant difference in both yield and quality of solution for CTRNNs of six or more neurons. The simple genetic algorithm simply can not cope with longer bit strings, while the mCGA seems quite capable of searching these larger spaces.

The reason for the relatively poor yields of CPG3 networks has been identified and discussed extensively in other works [Beer, Chiel, and Gallagher; 1999][Chiel, Beer, and Gallagher; 1999]. In short, three-neuron CTRNNs lack sufficiently many degrees of freedom to properly solve the locomotion problem. The relative scarcity of three-neuron solutions increases the difficulty of the search. It also puts a cap on the maximum effectiveness of the solutions evolved. Our data shows that mCGA performs no worse than the simple GA in the face of these difficulties. Preliminary experiments with single elimination tournaments in the mCGA, however, show an increase in the yields of CPG3s to 68%. We are currently engaged in further experimentation to better characterize this surprising phenomenon.

6. mCGA and the De Jong Test Functions

mCGA was developed against a specific CTRNN benchmark problem. During that development, we saw that though both the addition of elitism and mutation were useful, it was the addition of mutation that seemed to provide the most benefit. Both as a means of evaluating the effect of differing mutation rates and as means of evaluating the mCGA against standard benchmarks, we tested it against the De Jong test functions [De Jong, 1975]. We ran 100 mCGA searches for each of the five De Jong test functions for bitwise mutation rates of 0.0, 0.5, 0.1, 0.15, and 0.20. Each objective function parameter was coded with same precision and range as in De Jong's thesis. Because mCGA as formulated above doesn't converge for higher mutation rates, we capped the number of generations at 100,000 to be consistent with the CTRNN benchmarks previously discussed. Simulated population size was set to 100 for similar consistency with the CTRNN benchmarks.

mCGA always succeeded in finding the global optimal for De Jong F1. This is perhaps not surprising, as the unimodal and symmetric F1 represents a very easy optimization problem. We did note, however, that for F1, small mutation rates allowed for faster searching. On average, mCGA found the F1 global optimal after about 672 generations with a mutation rate of 0.0. With a mutation rate of 0.05, this was halved to about 332 generations. Higher rates, resulted in delayed appearances of the optimal. At mutation rates of 0.1, 0.15, and 0.2 it took on average 537, 1589, and 13363 generations respectively to find the global optimal.

Figure 2 shows the average scores, based on 100 runs, of the best solutions found for De Jong F2, F3, F4, and F5 for the same bitwise mutation rates listed above. Note that for F2, F3, and F5, increased mutation leads to an increased chance of finding the optimal. Also note, however, that just like with F1, increasing the mutation also increases the number of generations, on average, that one needs to wait for the optimal to appear. F4 behaves differently. A little

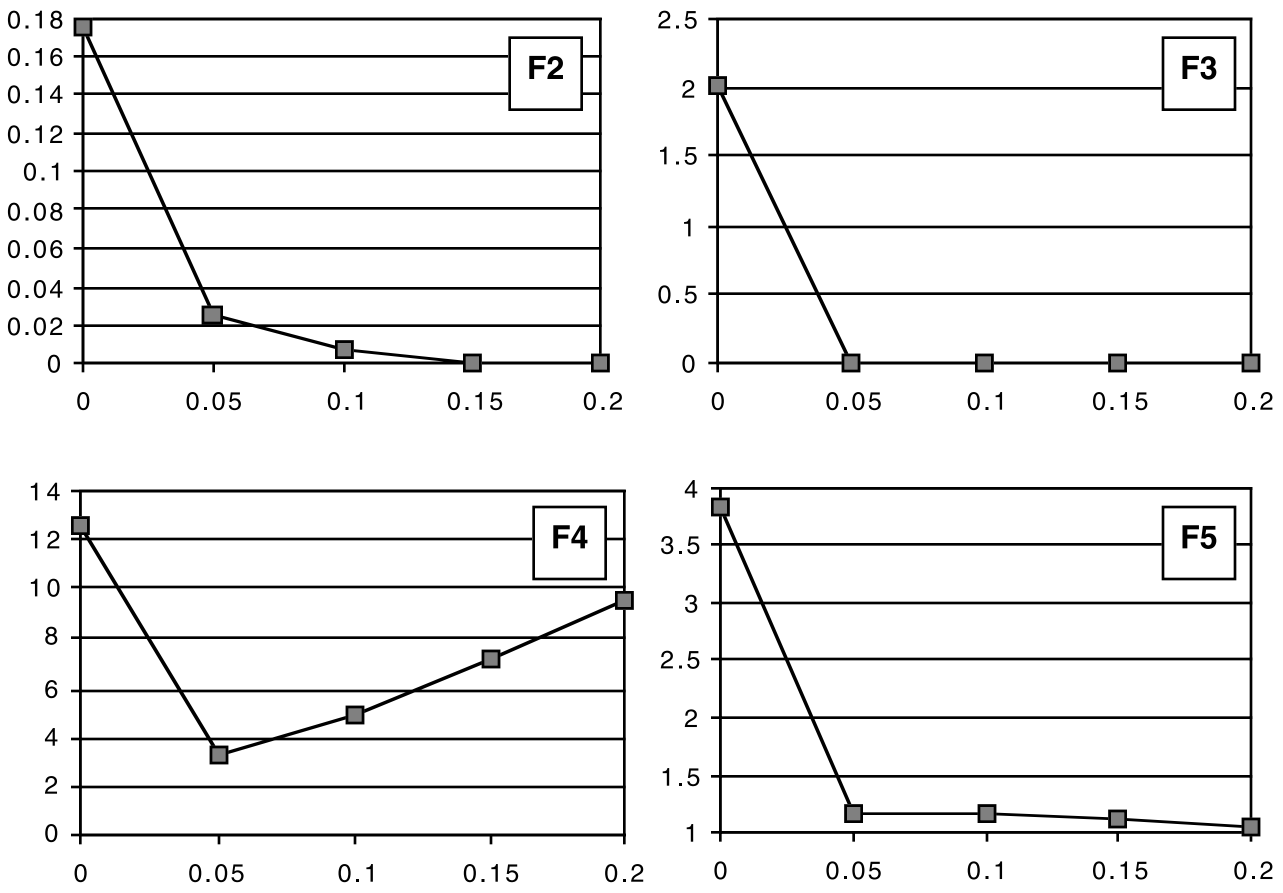

Figure 2: Average Errors Attained on De Jong F2 - F5
For each graph, the y-axis shows the average final error attained based on 100 runs. In all cases, the bottom tick on the y-axis represents the lowest attainable error score. The x-axis shows setting of the mCGA bitwise mutation rate.

mutation helps, but increasing it too much results in a degradation of the quality of solutions found. F4 is a simple unimodal function with gaussian noise and is meant to test how well an optimization algorithm deals with noisy objective functions. In the simple genetic algorithm, a mutation of an individual that received a deceptively good score by random chance would be likely to drop out of the population eventually. We would expect it would not receive deceptively good scores sufficiently often to allow it to spread through the population. mCGA, however, simulates a mutated individual having spread widely into the population instantaneously by modifying the probability vector that simulates the population. In a sense, the momentum effects provided by having a real population are not present in mCGA, and we could therefore expect it to be quite easily tricked by deceptive evaluations of individuals. This effect is magnified as we allow more mutation events to occur in a search. We will discuss possible fixes to this problem later in this paper.

7. A Proposed Hardware Implementation

A design for a hardware implementation of the compact genetic algorithm is in the literature [Aporntewan and Chongstitvatana, 2001]. A hardware implementation of our mCGA is not much more difficult to achieve. In this section, we will outline one possible hardware implementation of the mCGA. We will present a data path that supports all the operations needed to implement the mCGA as described in section three of this paper. We will also provide a qualitative description of the actions that an on-chip microcontroller needs to take to implement the mCGA.

A proposed data path is provided in Figure 3. The design is similar to that in [Aporntewan and Chongstitvatana, 2001], but contains additional machinery to implement elitism and mutation. The bit probability for a genome position as well as bits for that position for two candidates are held in a number of "bit modules". In figure 3, two bit modules are

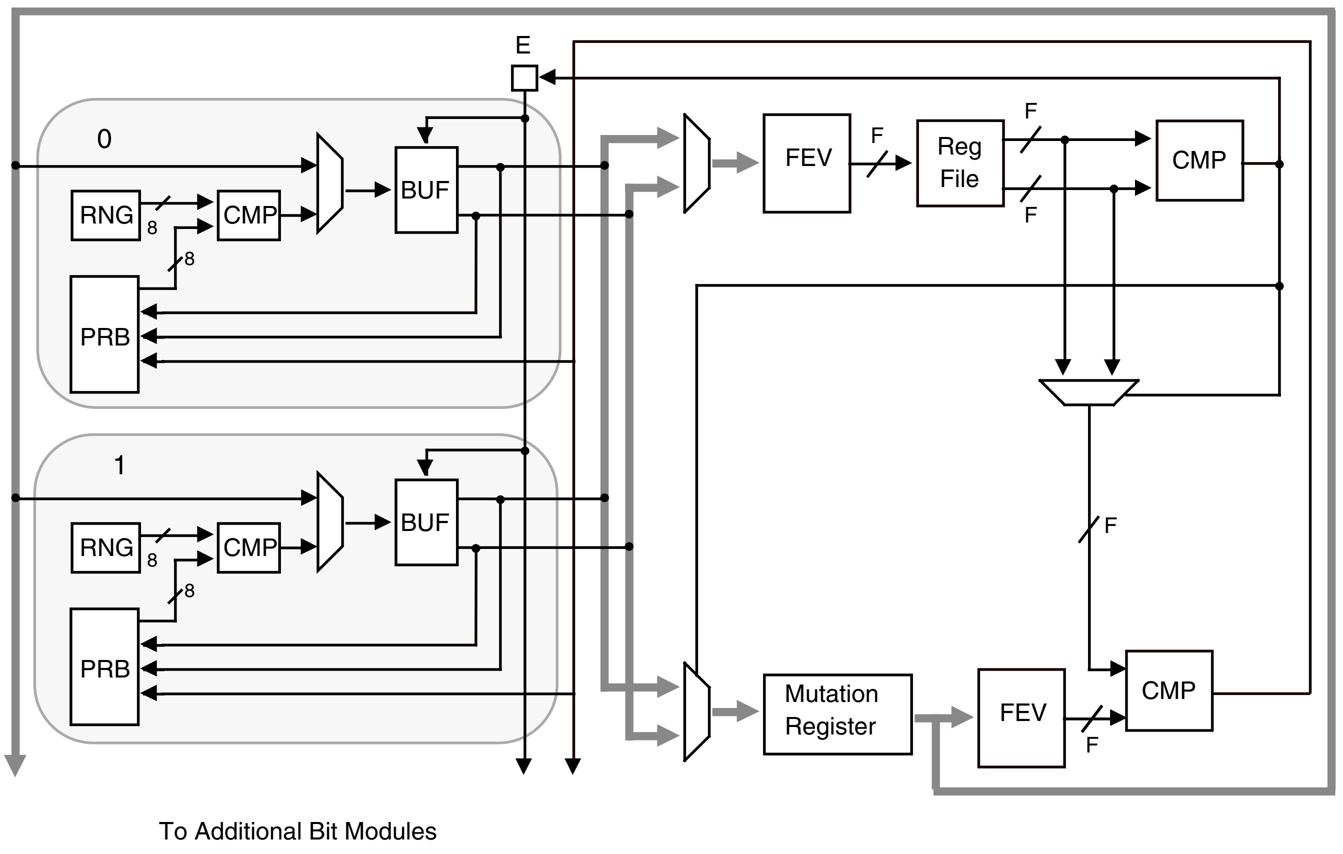

Figure 3: Sample Data Path for the mCGA
A sample data path capable of implementing the mCGA in hardware. Components are defined in the text. Bold gray lines are N bit busses where N is the number of bits in the genome. For large genomes, these wide busses can be replaced with serialized communication channels.

shown enclosed in gray boxes and labeled 0 and 1. Larger genomes can be implemented by adding additional modules as needed. The components in the data path are defined as follows:

RNG : A random number generator. When activated, this device will generate a random eight-bit number

CMP : A standard multibit comparator

PRB : The probability register. This register contains an eight bit value that encodes the probability of that bit position is a 1. The device also has control lines that allow incrementing, decrementing, or setting the probability to 0.5.

BUF : A 2x1 memory device. A select line allows one to load a bit into the "top" or "bottom" bit position. This device is used to hold bit values at a position for the two candidates under consideration.

E : E is a one bit register that encodes which of the two bits stored in each BUF (top or bottom) is part of the current "elite individual.

FEV : The Function Evaluator. This is hardware that provides an error score for the bit pattern supplied to it. This hardware could be on chip or it could obtain the error evaluation with off-chip circuitry. We encode the error score as an F bit number.

MR : Mutation Register. This device acts like a normal register except that it mutates bits according to a user specified mutation rate.

RF: Register File. A simple register file that stores the F bit errors for the "top" and "bottom" candidates stored in the bit modules.

MUX : Standard multibit multiplexers. Shown in the data path as unlabeled trapezoids.

The mCGA would be implemented by augmenting the data path with a microcontroller that would complete the following operations.

1. The Probability Registers (PRBs) are set to hold their initial probabilities of 0.5. All registers and buffers are zeroed.

2. Each RNG generates a random number. This value is compared with the contents of the corresponding PRB. Each CMP produces a 1 if the generated random number is less than the probability and a zero if it is greater. Each generated bit in each of the bit modules is written into the "top" slot of the buffer (BUF). The E bit, which was cleared in step one, designates the "top" slot as holding the current elite individual.

3. Step 2 is repeated, except this time the bits generated by each comparator are stored in the "bottom" slots of each BUF.

4. One at a time, all the bits in the top buffers are routed to the FEV module. Each string is evaluated and the F bit errors are stored in the "top" and "bottom" slots of the register file.

5. The errors sent from the register file to a comparator. The identity of the best score (top encoded as zero, bottom encoded as 1) is sent back to and stored into the E bit. We now know which of the two initially generated individuals is the better.

6. All the bits making up the current elite individual are routed into the mutation register, creating a mutated version of the current best individual.

7. The bits making up the mutated candidate string are routed to a FEV. The error resulting error score is compared with the previously computed error score of the best (taken from the register file). If the mutated individual is better, all the bits from the mutation register are copied into their corresponding bit modules into the currently elite buffer slots. The PRBs inside of bit modules corresponding to bits that changed are also instructed to reset to hold a probability of 0.5.

8. Each RNG generates a random number, which is compared with the corresponding PRB to generate a bit as described in step two. Each new bit is written into the NON ELITE slot in each bit buffer (BUF).

9. If our end condition is not met, go to step 4.

8. Conclusions and Discussion

We desire to combine reconfigurable analog neurons and a hardware-based genetic algorithm into a single device to be used to control and regulate physical processes. For our application, compact size is vital. The CGA is particularly amenable to implementation using standard VLSI techniques and would result in extremely compact circuitry. The CGA, however, is known to be a weak search method. We have shown that the unmodified CGA is not sufficiently powerful to evolve CTRNN controllers. However, by making simple modifications that do not require major increases in the number of gates necessary to implement the CGA in hardware, we can produce a modified CGA that competes well with the simple genetic algorithm. In fact, we have shown that our modified CGA, for this problem, is actually provides superior search performance for approximately the same computational cost. Additionally, we have shown mCGA, for the most part, performs well on the five De Jong test functions. Where it performs less well, specifically on randomized error functions, we have identified the problem and are in a position to fix it. These results alone are significant. At least as significant, however, are the interesting questions raised by these results

First, we expected that unmodified CGA would not be able to search CTRNN spaces effectively. We also expected that with appropriate modification, we would be able to make CGA work without incurring a great penalty in hardware cost. We did not expect that the modified CGA would outperform methods previously employed. We intend to carefully study our modified CGA to determine exactly why we observed this improved performance. We formulated our modifications based on our experience about what works for evolving CTRNNs. Did, however, we stumble on something that searches other spaces well too? We intend to answer this question by applying the mCGA to difficult search problems outside of our target problem domain. If more universally successful, we will more carefully and formally analyze the algorithm.

Second, the introduction of single elimination tournaments, unlike the other modifications we proposed, does somewhat increase the number of gates required for hardware implementation. Tournaments are, therefore, somewhat less attractive for our stated problem domain. However, that we have already observed a significant gain in yield for the difficult CPG3 problem is interesting. We intend to study the tournament modification more carefully in the future. It may be the case that increases in effective yield might be well worth the increased hardware costs.

Third, the single-leg CPG problem was not chosen arbitrarily. It happens to be the best studied, and because the controller may not make use of sensory feedback, one of the more challenging, locomotion control problems we have considered to date. This makes it a reasonable initial benchmark of mCGA efficacy. We need, however, to apply the search method to a wider variety of CTRNN control problems. Initial results are very encouraging. We have successfully used mCGAs to evolve hexapod locomotion controllers and controllers correcting arrhythmia in simulated human hearts. These results also represent important verifications against prior work. However, we desire to be careful and ensure that all our results possess a high degree of statistical significance. Therefore, we need to run more experiments against these other problems before we have enough instances to report more than anecdotally on wider success.

Fourth, we have extensively studied the neural dynamical principles underlying the operation of CTRNN locomotion controllers evolved using the simple genetic algorithm [Beer, Chiel, and Gallagher, 1999][Chiel, Gallagher, and Beer, 1999]. We have yet to rigorously study the principles underlying the operations of the mCGA produced controllers. One would expect them to operate using similar principles -- however, we have three mCGA produced CPG3's that seem to defy explanation using the dynamical systems techniques previously developed. Analysis of these devices is under way. The differences in the resulting products may reveal that the mCGA is searching a portion of CTRNN space difficult for previous methods to reach. The nature of the "difficult to reach space" may provide important clues on why our modifications work so well and perhaps, suggest additional modifications that might help us search CTRNN spaces more reliably.

In the future, we intend to expand and optimize the hardware mCGA. This should be a fairly straightforward task and we expect to have completely validated designs very shortly.

In conclusion, we have demonstrated that the marriage of the mCGA and reconfigurable CTRNN hardware is likely to produce compact, capable EH chips. In addition to providing an important feasibility result, we have also opened several interesting new lines of inquiry that will likely provide equally interesting results as they are pursued. Our mCGA is almost certainly well suited to CTRNN-EH, and may be adaptable to other EH efforts as well.

Acknowledgments

The authors would like to thank Steven Perretta and Wendy Peluso for their comments and suggestions on this manuscript.

References

Aporntewan, C. and Chongstitvatana, Prabhas. (2001). A hardware implementation of the compact genetic algorithm. In *The Proceedings of the 2001 IEEE Congress on Evolutionary Computation.* Seoul, Korea. IEEE Press

Beer, R.D. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior* **3**(4):469-509.

Beer, R.D., Chiel, H.J. & Gallagher, J.C. (1999). Evolution and analysis of model CPGs for walking II. General principles and individual variability. *J. Computational Neuroscience* 7(2):119-147. (Kluwer).

Chiel, H.J., Beer, R.D., & Gallagher, J.C. (1999). Evolution and analysis of model CPGs for walking I. Dynamical modules. *J. Computational Neuroscience* 7:(2):99-118.

De Jong, K.A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, University of Michigan, Ann Arbor. University Microfilms Num 76-9381.

Gallagher, J.C. (1998). *A dynamical systems analysis of the neural basis of behavior in an artificial autonomous agent.* Ph.D. Doctoral dissertation, Case Western Reserve University. Cleveland, OH. USA.

Gallagher, J.C. and Fiore, J.M. (2000). Continuous time recurrent neural networks: a paradigm for evolvable analog controller circuits. in *The Proceedings of the National Aerospace and Electronics Conference*. IEEE Press.

Harik, G.R., Lobo, F.G., and Goldberg, D.E. (1999). The compact genetic algorithm. In *IEEE Transactions on Evolutionary Computation* vol. 3 no. 4. pp 287-297

Kajitani, T., Hoshino, T. Nishikawa, D., et.al. (1998). A gate level EHW chip: implementing GA operations and reconfigurable hardware on a single LSI. In *Proceedings of the International Conference on Evolvable Hardware (ICES 1998).*

Scott, S. and Seth, A. (1995). HGA: a hardware-based genetic algorithm, In *Proceedings of the ACM/SIGDA Third Int. Symposium on Field-Programmable Gate Arrays.*

Yoshida, N. and Yasuoka, T. (1999). Multi-gap : parallel and distributed genetic algorithms in VLSI. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*