EVOLVING BOTH HARDWARE SUBSYSTEMS AND THE SELECTION OF VARIANTS OF SUCH INTO AN ASSEMBLED SYSTEM

Jim Torresen
Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
E-mail: jimtoer@ifi.uio.no

KEYWORDS
Electronics, Parallel methods, Performance analysis, Signal processing, Signal processors.

ABSTRACT

Evolvable Hardware (EHW) has been proposed as a new method for designing electronic circuits. However, there are several problems to solve for making high performance systems. One is the limited scalability of the ordinary approach. To reduce this problem, a novel digital signal processing architecture has been developed that allows for incremental evolution. This is based on initially evolving subcircuits. In this paper, it is extended by evolving the best possible combination of subcircuits. Each such circuit is selected among a set of alternative circuits. The architecture is applied as a prosthetic hand controller. By applying the proposed method, the average performance is improved compared to more ordinary approaches.

INTRODUCTION

Recently, several automated procedures suited for design of pattern recognition and control problems have been developed. One of these is called evolvable hardware (EHW). It has been applied to a large range of real-world applications. One - implied in this paper, is prosthetic hand control.

To enhance the lives of people who have lost a hand, prosthetic hands have existed for a long time. These are operated by the signals generated by contracting muscles – named electromyography (EMG) signals, in the remaining part of the arm (Scott and Parker, 1988). Presently available systems normally provide only two motions: Open and close hand grip. The systems are based on the user adapting *himself* to a fixed controller. That is, he must train himself to issue muscular motions trigging the wanted action in the prosthetic hand. A long time is often required for rehabilitation.

By using EHW it is possible to make the *controller* itself adapt to each disabled person. The controller is constructed as a pattern classification hardware which maps input patterns to desired actions of the prosthetic hand. Adaptable controllers have been proposed based on neural networks (Fuji, 1998). These require a floating point CPU or a neural network chip. EHW based controllers, on the other hand, use a few layers of digital logic gates for the processing. Thus, a more compact implementation can be provided making it more feasible to be installed inside a prosthetic hand.

Experiments based the EHW approach have already been undertaken by Kajitani et al (Kajitani et al., 1999). The research on adaptable controllers is based on designing a controller providing six different motions in three different degrees of freedom. Such a complex controller could probably only be designed by *adapting* the controller to each dedicated user. It consists of AND gates succeeded by OR gates (Programmable Logic Array). The latter gates are the outputs of the controller, and the controller is evolved as one complete circuit. The simulation indicates a similar performance as artificial neural network but since the EHW controller requires a much smaller hardware it is to be preferred.

One of the main problems in evolving hardware systems seems to be the limitation in the chromosome string length (Lee et al., 1997; Yao and Higuchi, 1997). A long string is normally required for solving a complex problem as seen in Figure 1.

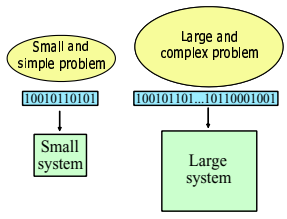

Figure 1: The Chromosome String Length and Representation Ability

However, a larger number of generations is required by the evolutionary algorithm as the string increases. This often makes the search space becoming too large. Thus, work has been undertaken to try to diminish this limitation. Various experiments on speeding up the evolution have been undertaken – see (Torresen, 2000).

Incremental evolution for EHW was first introduced in (Torresen, 1998) for a character recognition system. The approach is a divide-and-conquer on the evolution of the EHW system, and thus, named *increased complexity evolution*. It consists of a division of the *problem* domain together with incremental evolution of the hardware system. Evolution is first undertaken individually on a set of basic units. The evolved units are the building blocks used in further evolution of a larger and more complex system. The benefits of applying this scheme is both a *simpler* and *smaller* search

space compared to conducting evolution in one single run. The goal is to develop a scheme that could evolve systems for complex real-world applications.

In this paper, it is applied to evolve a prosthetic hand controller circuit. A new EHW architecture as well as how incremental evolution is applied are described. Further, it is extended by evolving the best possible combination of subcircuits. Each such circuit is selected among a set of alternative circuits. This should improve the generalization performance of gate level EHW and make it a strong alternative to artificial neural networks.

The next two sections introduce the concepts of the evolvable hardware based prosthetic hand controller. Then results are given in one section with conclusions in the last section.

PROSTHETIC HAND CONTROL

The research on adaptable controllers presented in this paper is based on designing controllers providing six different motions in three different degrees of freedom: Open and Close hand, Extension and Flection of wrist, Pronation and Supination of wrist. The data set consists of the same motions as used in earlier work (Kajitani et al., 1999), and it is collected by Dr. Kajitani at Electrotechnical Laboratory in Japan.

The published results on adaptive controllers are usually based on data for non-disabled persons. Since it is possible to observe the hand motions, a good training set can be generated. For the disabled person this is not possible since there is no hand to observe. The person would have to by himself distinguish the different motions. Thus, it would be a harder task to get a high performance for such a training set but it will indicate the expected response to be obtainable by the prosthesis user. This kind of training set is applied in this paper. Some of the initial results using this data set can be found in (Torresen, 2001).

Data Set

The absolute value of the EMG signal is integrated for 1 s and the resulting value is coded by *four* bits. To improve the performance of the controller it is beneficial to be using several channels. In these experiments *four* channels were used in total, giving an input vector of 4 x 4 = 16 bits. The controller interfaces are illustrated in Figure 2.

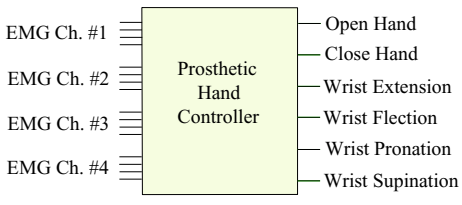

Figure 2: Illustration of the Controller Interfaces

The *output* vector consists of one binary output for each hand motion, and therefore, the output vector is coded by *six* bits. For each vector only *one* bit is "1". Thus, the data set is collected from a disabled person by considering one motion at a time. For each of the six possible motions, a total of 50 data vectors are collected, resulting in a total of: 6 x 50 = 300 vectors. Further, *two* such sets were made, one to be used for evolution (training) and the others to be used as a separate test set for evaluating the best circuit *after* evolution is finished.

AN ARCHITECTURE FOR INCREMENTAL EVOLUTION

In this section, the proposed architecture for the controller is described. This includes the algorithm for undertaking the incremental evolution. This is all based on the principle of *increased complexity evolution*.

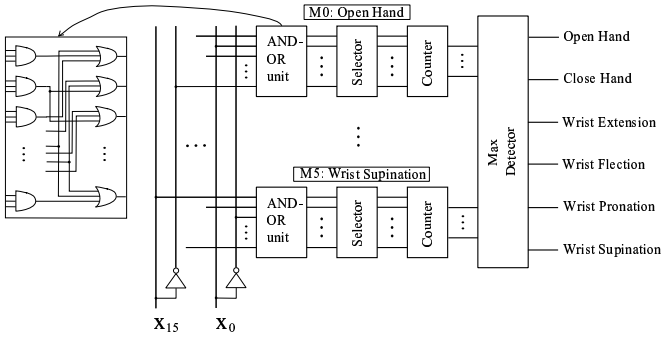

Figure 3: The Digital Gate Based Architecture of the Prosthetic Hand Controller

The architecture is illustrated in Figure 3. It consists of one subsystem for *each* of the six prosthetic motions. In each subsystem, the binary inputs $x_0 \ldots x_{15}$ are processed by a number of different units, starting by the AND-OR unit. This is a layer of AND gates followed by a layer of OR gates. Each gate has the same number of inputs, and the number can be selected to be two, three or four. The outputs of the OR gates are routed to the Selector. This unit selects *which* of these outputs that are to be counted by the succeeding counter. That is, for each new input, the Counter is counting the number of *selected* outputs being "1" from the corresponding AND-OR unit. Finally, the Max Detector outputs which counter – corresponding to *one* specific motion, is having the largest value. Each output from the Max Detector is connected to the corresponding motor in the prosthesis. If the Counter having the *largest* value corresponds to the correct hand motion, the input has been correctly classified. One of the motivations for introducing the selectors is to be able to adjust the *number* of outputs from each AND-OR unit in a flexible way. A scheme, based on using multi-input AND gates together with counters, has been proposed earlier (Yasunaga et al., 2000). However, the architecture used in this paper is distinguished by including OR-gates, together with the selector units involving incremental evolution. The incremental evolution of this system can be described by the following steps:

1. **Step 1 evolution.** Evolve the AND-OR unit for each subsystem *separately* one at a time. Apply *all* vectors in the training set for the evolution of each subsystem. There are no interaction among the subsystems at this step, and the fitness is measured on the output of the AND-OR units.

2. **Step 2 evolution.** Assemble the six AND-OR units into one system as seen in Figure 3. The AND-OR units are now fixed and the *Selectors* are to be evolved in the assembled system – in one common run. The fitness is measured using the same training set as in step 1 but the evaluation is now on the output of the Max Detector.

3. The system is now ready to be applied in the prosthesis.

In the first step, subsystems are evolved separately, while in the second step these are evolved together. The motivation for evolving separate subsystems – instead of a single system in one operation, is that earlier work has shown that the evolution time can be substantially reduced by this approach (Torresen, 2000; Torresen, 1998).

The layers of AND and OR gates in one AND-OR unit consist of 32 gates each. This number has been selected to give a chromosome string of about 1000 bits which has been shown earlier to be appropriate. A larger number would have been beneficial for expressing more complex Boolean functions. However, the search space for evolution could easily become too large. For the step 1 evolution, each gate's *inputs* are determined by evolution. The encoding of each gate in the binary chromosome string is as follows:

Inp.1 (5 bit)	Inp.2 (5 bit)	(Inp.3 (5 bit))	(Inp.4 (5 bit))

As described in the previous section, the EMG signal input consists of 16 bits. Inverted versions of these are made available on the inputs as well, making up a total of 32 input lines to the gate array. The evolution is based on gate level building blocks. However, since several output bits are used to represent one motion, the signal resolution becomes increased from the two binary levels.

For the step 2 evolution, each line in each selector is represented by *one* bit in the chromosome. This makes a chromosome of 32 x 6 bits= 192 bits. If a bit is "0", the corresponding line should *not* be input to the counter, whereas if the bit "1", the line *should* be input.

Fitness Measure.

In step 1 evolution, the fitness is measured on all the 32 outputs of each AND-OR unit. As an alternative experiment, we would like to measure the fitness on a *limited* number (16 is here used as an example) of the outputs. That is, each AND-OR unit still has 32 outputs but – as seen in Figure 4, only 16 are included in the computation of the fitness function:

$$\text{Fitness} = \sum_{i=1}^{16} \text{Output OR gate } i \qquad (1)$$

The 16 outputs not used are included in the chromosome and have *random* values. That is, their values do not affect the fitness of the circuit. After evolution, all 32 outputs are applied for computing the performance:

$$\text{Performance} = \sum_{i=1}^{32} \text{Output OR gate } i \qquad (2)$$

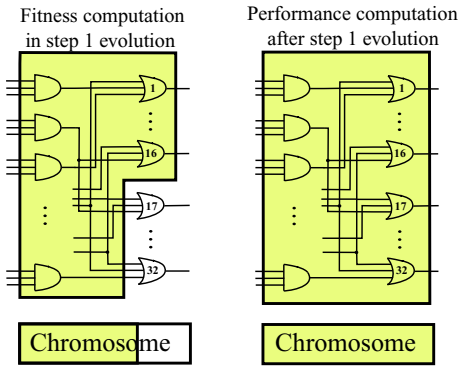

Figure 4: A "Fitness Measure" Equal to 16

Since 16 OR gates are used for fitness computation, the "fitness measure" equals 16. In the figure, gate 1 to 16 are used for the fitness function. However, in principle any 16 gates out of the 32 can be used. Other numbers than 16 were tested in experiments but 16 showed to give the best performance results and was used in the following reported experiments.

This could be an interesting approach to improve the generalisation of the circuit. Only the OR gates in the AND-OR unit are "floating" during the evolution since all AND gates may be inputs to the 16 OR gates used by the fitness function. The 16 "floating" OR-gates then provide additional combination of these *trained* AND gates.

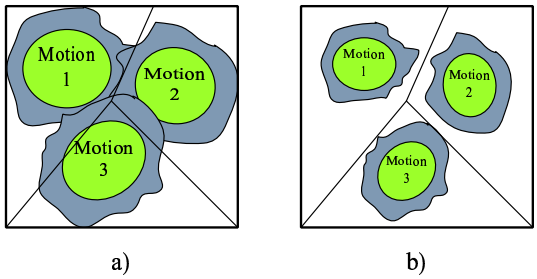

Figure 5: Illustration of Noise Added to a) A Plain Signal and b) A Pre-Processed Signal

Another way to look at this is that the "floating" gates provide "noise". However, the noise is not added to the plain input but to a preprocessed and *improved* signal (output from the AND gates) as illustrated in Figure 5. The inner circle for each motion indicates the *training set* domain, with the outer circle indicating the added generalisation obtained by adding "noise".

In a) the signal is not pre-processed and adding noise makes the interference among classes worse while in b) it improves the generalisation rather than introducing interference. The step 2 evolution will be evolving the ratio of noise in the final system by adjusting the number of selector bits set for gates 1 to 16 compared to for gates 17 to 32.

Fitness Function

The fitness function is important for the performance when evolving circuits. For the step 1 evolution, the fitness function – applied for each AND-OR unit separately, is as follows

for the motion m ($m \in [0, 5]$) unit:

$$F_1(m) = \frac{1}{s} \sum_{j=0}^{50m-1} \sum_{i=1}^{O} x + \sum_{j=50m}^{50m+49} \sum_{i=1}^{O} x + \frac{1}{s} \sum_{j=50m+50}^{P-1} \sum_{i=1}^{O} x$$

$$\text{where } x = \begin{cases} 0 & \text{if } y_{i,j} \neq d_{m,j} \\ 1 & \text{if } y_{i,j} = d_{m,j} \end{cases}$$

where $y_{i,j}$ in the computed output of OR gate i and $d_{m,j}$ is the corresponding target value of the training vector j. P is the total number of vectors in the training set ($P = 300$). As mentioned earlier, each subsystem is trained for one motion (the middle expression of F_1). This includes outputting "0" for input vectors for other motions (the first and last expressions of F_1).

The s is a scaling factor to implicitly emphasize on the vectors for the motion the given subsystem is assigned to detect. An appropriate value ($s = 4$) was found after some initial experiments. The O is the number of outputs included in the fitness function and is either 16 or 32 in the following experiments (referred to as "fitness measure" in the previous section).

The fitness function for the step 2 evolution is applied on the complete system and is given as follows:

$$F_2 = \sum_{j=0}^{P-1} x \qquad \text{where}$$

$$x = \begin{cases} 1 & \text{if } d_{m,j} = 1 \text{ and } m = i \text{ for which } \max_{i=0}^{5}(Counter_i) \\ 0 & \text{else} \end{cases}$$

This fitness function counts the number of training vectors for which the target *output* being "1" *equals* the *id* of the counter having the maximum output (as mentioned earlier only *one* output bit is "1" for each training vector).

Subsystem Selection in Step 2 Evolution.

Several subsystems (e.g. four AND-OR units in the following experiments) are evolved in the step 1 evolution for *each* hand motion. Still, normally only *one* AND-OR unit is evolved at a time for each of the six hand motions. These six become assembled and their total performance is measured. This is followed by the step 2 evolution. This is repeated for four times. However, there could be high performing system combinations consisting of units from the four *different* runs.

It is here proposed to incorporate the selection of the unit for each motion into the step 2 evolution as illustrated in Figure 6. In addition to evolving the selectors, one of four units is selected for each motion. The goal of this approach is to evolve (in step 2 evolution) systems with higher performance than what is achieved for the best system evolved the original way (only selector evolution) (Torresen, 2001).

One possible obstacle of this approach is that the best selector settings for one AND-OR unit are probably different from another one. Thus, if a unit is substituted by mutation (crossover could lead to the same phenomen – depending on the crossover point), this system could easily get a low fitness value and be excluded from the population. This occurs even

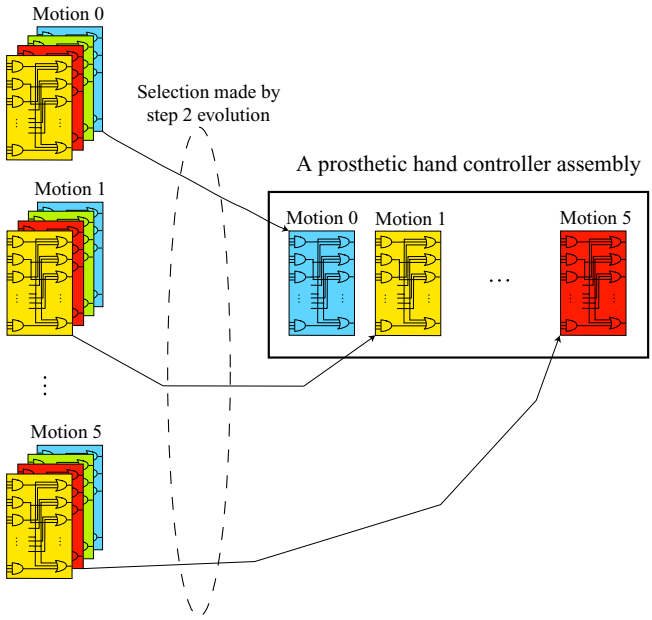

Figure 6: Subsystem Selection in Step 2 Evolution

though it could have achieved a high performance if the corresponding selector settings had been allowed to be adjusted.

Intelligent recombination operators could be applied in this scheme. If the unit applied for one motion is substituted by mutation, then it could be possible to substitute all the selector settings by those which may have been evolved earlier for this unit. Another choice is to run a mutation operator for some generations only on the selectors for the unit(s) that have been substituted. This approach could reduce the problem presented in the previous paragraph. This has not yet been implemented.

The Evolutionary Algorithm

The simple Genetic Algorithm (GA) – given by Goldberg (Goldberg, 1989), was applied for the evolution with a population size of 50. For each new generation an entirely new population of individuals is generated. Elitism is used, thus, the best individuals from each generation are carried over to the next generation. The (single point) crossover rate is 0.8, thus the cloning rate is 0.2. Roulette wheel selection scheme is applied. The mutation rate – the probability of bit inversion for each bit in the binary chromosome string, is 0.01. For some of the following experiments, other parameters have been used, but these are then mentioned in the text.

Various experiments were undertaken to find appropriate GA parameters. The ones that seemed to give the best results were selected and fixed for all the experiments. This was necessary due to the large number of experiments that would have been required if GA parameters should be able vary through all the experiments. The preliminary experiments indicated that the parameter setting was not a major critical issue.

The proposed architecture fits into most FPGAs. The evolution is undertaken off-line using software simulation. However, since no feed-back connections are used and the number of gates between the input and output is limited, the real per-

Type of system	# inp/gate	Step 1 evolution			Step 1+2 evolution		
		Min	Max	Avr	Min	Max	Avr
A: Fitness measure 16 (train)	3	63.7	69.7	65.5	71.33	76.33	73.1
A: Fitness measure 16 (test)	3	50.3	60.7	55.7	44	67	55.1
B: Fitness measure 32 (train)	3	51	57.7	53.4	70	76	72.9
B: Fitness measure 32 (test)	3	40	46.7	44.4	45	54.3	50.1
C: Direct evolution (train)	4	56.7	63.3	59.3	-	-	-
C: Direct evolution (test)	4	32.7	43.7	36.6	-	-	-

Table 1: The Results of Evolving the Prosthetic Hand Controller in Different Ways

formance should equal the simulation. Any spikes could be removed using registers in the circuit.

For each experiment presented, four different runs of GA were performed. Thus, *each* of the four resulting circuits from step 1 evolution is taken to step 2 evolution and evolved for four runs.

RESULTS

This section reports the experiments undertaken to search for an optimal configuration of the prosthetic hand controller. They will be targeted at obtaining the best possible performance for the *test* set.

Table 1 shows the main initial results – in percentage correct classification (Torresen, 2001). Several different ways of evolving the controller are included. The training set and test set performances are listed on separate lines in the table. Each gate in the AND-OR unit has three or four inputs. The columns beneath "Step 1 evolution" report the performance after only the *first* step of evolution. That is, each subsystem is evolved separately, and afterwards they become assembled to compute their total performance. The "Step 1+2 evolution" columns show the performance when the *selector units* have been evolved too (step 2 of evolution without subsystem selection). In average, there is an improvement in the performance for the latter. Thus, the proposed *increased complexity evolution* give rise to improved performances.

In total, the best way of evolving the controller is the one listed first in the table. The circuit evolved with the best *test set* performance obtained 67% correct classification. The circuit had a 60.7% test set performance after step 1 evolution (evaluated with all 32 outputs of the subsystems). Thus, the step 2 evolution provides a substantial increase up to 67%. Other circuits didn't perform that well, but the important issue is that it has been shown that the proposed architecture provides the *potential* for achieving high degree of generalization.

A feed-forward neural network was trained and tested with the same data sets. The network consisted of (two weight layers with) 16 inputs, 40 hidden units and 6 outputs. In the best case, a test set performance of 58.8% correct classification was obtained. The training set performance was 88%. Thus, a higher training set performance but a lower test set performance than for the best EHW circuit. This shows that the EHW architecture holds good generalisation properties.

The experiment B is the same as A except that in B all 32 outputs of each AND-OR unit are used to compute the fitness function in the step 1 evolution. In A, each AND-OR unit also has 32 outputs but only 16 are included in the computation of the fitness function as described in the "Fitness Measure" section. The performance of A in the table for the step 1 evolution is computed by using *all* the 32 outputs. Thus, over 10% better training set as well as the test set performance (in average) are obtained by having 16 outputs "floating" rather than measuring their fitness during the evolution.

Each subsystem is evolved for 10,000 generations each, whereas the step 2 evolution was applied for 100 generation. These numbers were selected after a number of experiments. The circuits evolved with direct evolution (E) were undertaken for 100,000 generations. This is more than six times 10,000 which were used in the other experiments. The training set performance is impressive when thinking of the simple circuit used. Each motion is controlled by a *single* four input OR gate. However, the test set performance is very much lower than what is achieved by the other approaches.

Subsystem Selection in Step 2 Evolution.

To test the feature of selecting units in addition to selector settings in step 2 evolution, experiments were undertaken by using the best circuits from step 1 evolution. These were the AND-OR units from the four runs named "A" earlier, and each having the test set performance shown in Table 2. This table also includes the average performance of the four step 2 runs for each circuit evolved in a step 1 run (no unit selection in step 2).

	Run number				Avr
	1	2	3	4	
Step 1 evolution	60.7	57	50.3	54.7	55.7
Step 1+2 evolution	61.9	54.2	52.6	51.8	55.1

Table 2: The Test Set Performance of the Four Runs of Experiment A

Type of system	Step 1+2 evolution		
	Min	Max	Avr
Selecting units from four runs	49.7	57.7	54.2
Sel. from two "best" runs (#1 and #2)	57.3	64.3	59.4
Sel. from two "worst" runs (#3 and #4)	49.3	57.3	54.9

Table 3: The Test Set Performance After Evolving both Selectors and the Selection of AND-OR Unit From Different Runs in Step 2 Evolution

	Run #	Motion						Perf
		1	2	3	4	5	6	
Selecting units from the two "best" runs	1	1	2	1	1	2	1	58
(Run #1 and #2 in step 1 evolution of A)	2	1	2	2	1	1	1	64.3
	3	1	2	2	1	1	1	57.3
	4	1	2	1	1	1	1	58
Selecting units from the two "worst" runs	1	3	4	4	3	3	3	57.3
(Run #3 and #4 in step 1 evolution of A)	2	3	4	4	4	3	3	57.3
	3	4	4	4	3	4	4	49.3
	4	3	4	4	4	3	3	55.67

Table 4: A List of Which Run a Unit is Selected From for Each Prosthetic Motion

Table 3 reveals the results when the AND-OR units are selected in step 2 evolution in addition to the selector settings. Each row in the table is the result of four runs. Each run is continued for 1000 generations. The crossover rate is 0.5 and the mutation rate is 0.005. By selecting units from all *four* runs, the average performance is less than the original step 2 evolution. This could be due to different optimal selector settings for each of the AND-OR units and lead to the problem described earlier (when a unit is substituted by mutation).

Better results are achieved, when selecting units only from *two* runs. The two "best" runs are those referred to as run #1 and #2 in Table 2, while the two "worst" are run #3 and #4. The best evolved circuit is not better than the best one evolved by the original step 2 evolution. However, the performances ("Avr") are *better* than the *averages* of the two corresponding runs by the original step 2 evolution scheme. These average to 58.1% (run #1 and #2) and 52.2% (run #3 and #4), respectively. The total average (of the two average figures in Table 3) for selecting units from two runs is higher than any other experiment undertaken (57.2%). Further, the *average* performance of 59.4% is better than that obtained for artificial neural networks in the *best* case.

A plot of the test set performance (measured each time the training set performance was improved) during the evolution is included in Figure 7. This is for selecting among run #1 and #2. The performance varies throughout the evolution. One important result is that the worst performing circuit has the best performance in the end of the evolution. If computation speed is a major issue, the evolution could have been stopped at around 100 generations with almost the same performance as for 1000 generations.

In Table 4, it is listed which AND-OR unit is selected for each motion. When selecting units from the two "best" – step 1 evolution runs, a majority of run #1 is selected. This is reasonable since it resulted in the best performance in the original step 2 evolution – see Table 2. For the two worst runs it is in average selected equally from the two runs, which could be explained by the less performance difference for the two runs (in both step 1 and step 2 evolution). Thus, the approach seems to be successful in selecting the best out of two systems. Moreover, the scheme is then promising for increasing the average test set performance.

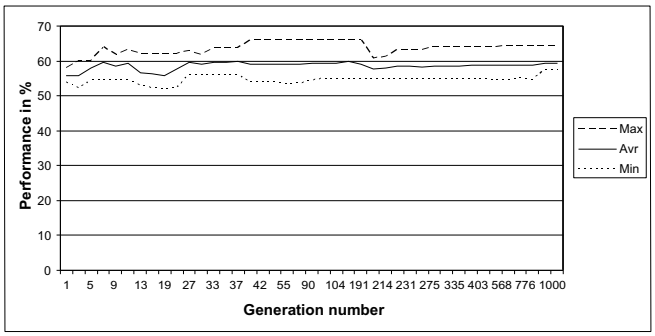

Figure 7: The Test Set Performance of the Four Different Runs When Selecting Among the Two Best Runs From Step 1 Evolution

CONCLUSIONS

In this paper, an EHW architecture for pattern classification including incremental evolution has been introduced. Experiments have been undertaken for selecting the best combination of circuits from different runs. The best circuit evolved shows a slightly better *average* generalization performance that what was obtained by artificial neural networks in the *best* case. The results illustrate that this is a promising approach for evolving systems for complex real-world applications.

ACKNOWLEDGMENTS

The author would like to thank the group leader Dr. Higuchi and the researchers in the Evolvable Systems Laboratory, National Institute of Advanced Industrial Science and Technology (AIST), Japan for inspiring discussions and fruitful comments on my work, during my visit there in January-April 2000. Further, I will express my gratefulness to the Japan Science and Technology Corporation (JST) for awarding me the STA fellowship making the visit possible.

REFERENCES

Fuji, S. (1998). Development of prosthetic hand using adaptable control method for human characteristics. In *Proc. of Fifth International Conference on Intelligent Autonomous Systems*, pages 360–367.

Goldberg, D. (1989). *Genetic Algorithms in search, optimization, and machine learning*. Addison–Wesley.

Kajitani, I., Hoshino, T., Kajihara, N., Iwata, M., and Higuchi, T. (1999). An evolvable hardware chip and its application as a multi-function prosthetic hand controller. In *Proc. of 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 182–187.

Lee, W.-P., Hallam, J., and Lund, H. (1997). Learning complex robot behaviours by evolutionary computing with task decomposition. In Birk, A. and Demiris, J., editors, *Learning Robots: Proc. of 6th European Workshop, EWLR-6 Brighton*, volume 1545 of *Lecture Notes in Artificial Intelligence*, pages 155–172. Springer-Verlag.

Scott, R. and Parker, P. (1988). Myoelectric prostheses: State of the art. *Journal of Medical Engineering and Technology*, 12(4):143–151.

Torresen, J. (1998). A divide-and-conquer approach to evolvable hardware. In Sipper, M. et al., editors, *Evolvable Systems: From Biology to Hardware. Second International Conference, ICES 98*, volume 1478 of *Lecture Notes in Computer Science*, pages 57–65. Springer-Verlag.

Torresen, J. (2000). Scalable evolvable hardware applied to road image recognition. In et al., J. L., editor, *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 245–252. IEEE Computer Society, Silicon Valley, USA.

Torresen, J. (2001). Two-step incremental evolution of a digital logic gate based prosthetic hand controller. In *Evolvable Systems: From Biology to Hardware. Fourth International Conference, (ICES'01)*, volume 2210 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag.

Yao, X. and Higuchi, T. (1997). Promises and challenges of evolvable hardware. In Higuchi, T. et al., editors, *Evolvable Systems: From Biology to Hardware. First International Conference, ICES 96*, volume 1259 of *Lecture Notes in Computer Science*, pages 55–78. Springer-Verlag.

Yasunaga, M., Nakamura, T., Yoshihara, I., and Kim, J. (2000). Genetic algorithm-based design methodology for pattern recognition hardware. In Miller, J. et al., editors, *Evolvable Systems: From Biology to Hardware. Third International Conference, ICES 2000*, volume 1801 of *Lecture Notes in Computer Science*, pages 264–273. Springer-Verlag.

BIOGRAPHY

JIM TORRESEN was born in Mandal, Norway. He received his M.Sc. and Ph.D. degrees in computer architecture and design from the Norwegian University of Science and Technology, University of Trondheim in 1991 and 1996, respectively. He was then employed as a senior embedded system designer in several different companies. Since 1999, he has been an associate professor at the Department of Informatics at the University of Oslo. Jim Torresen has been a visiting researcher at Kyoto University, Japan for one year (1993-1994) and four months at AIST (formerly Electrotechnical laboratory), Tsukuba, Japan (1997 and 2000). His main research interests are evolvable hardware and real-world applications.

E-mail: jimtoer@ifi.uio.no
Web: http://www.ifi.uio.no/∼jimtoer/