Two-Step Incremental Evolution of a Prosthetic Hand Controller Based on Digital Logic Gates

Jim Torresen

Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
jimtoer@ifi.uio.no
http://www.ifi.uio.no/~jimtoer

Abstract. Evolvable Hardware (EHW) has been proposed as a new method for designing systems for real-world applications. In this paper it is applied for evolving a prosthetic hand controller. It is shown that better generalization performance than neural networks can be obtained. The proposed architecture is based on digital logic gates and its configuration is determined by two separate steps of evolution.

1 Introduction

To enhance the lives of people who has 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 [1]. 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. Long time is often required for rehabilitation.

By using Evolvable Hardware (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 [2]. These require a floating point CPU or a neural network chip. However, by using gate level EHW, a much 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 [3]. 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 [4, 5]. A long string is normally required for representing a complex system. However, a larger number of generations is required by genetic algorithms (GA) as the string increases. This often makes the search space too large. Thus, work has been undertaken to try to diminish this limitation. Various experiments on speeding up the GA computation have been undertaken [6]. The schemes involve fitness computation in parallel or a partitioned population evolved in parallel – by parallel computation. Other approaches to the problem have been by using variable length chromosome [7] and reduced genotype representation [8]. Another option, called function level evolution, is to evolve at a higher level than gate level [9]. Most work is based on fixed functions. However, there has been work in Genetic Programming for *evolving* the functions [10]. The method is called Automatically Defined Functions (ADF) and is used in software evolution.

Another improvement to artificial evolution – called co-evolution, has been proposed [11]. In co-evolution, a part of the data, which defines the problem, co-evolves simultaneously with a population of individuals solving the problem. This could lead to a solution with a better generalization than a solution evolved based on the initial data. Further overview of related works can be found in [12].

Incremental evolution for EHW was first introduced in [13] 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*. The goal is to develop a scheme that could evolve systems for complex real-world applications. In this paper, it is applied to the application of a prosthetic hand controller circuit. Several improvements in the EHW architecture as well as how incremental evolution is applied are to be introduced. These 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. Results are given in Section 4 with conclusions in Section 5.

2 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 [3], 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 you may observe the hand motions, a good training set can be generated. For the disabled person this is not possible since there is no hand 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. No other publication is yet available – to make a comparison of the results, where this data set is used.

2.1 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 *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.

3 A Gate Architecture for Incremental Evolution

The evolution scheme – introduced as *increased complexity evolution*, has been proposed to overcome the problem of a long chromosome string. The idea is to evolve a system gradually. Evolution is first undertaken individually on a set of basic units. Each of these could contain gates or higher level functions as building blocks. The evolved functions are the basic blocks used in further evolution (or assembly) of a larger and more complex system. This may continue until a final system is at a sufficient level of complexity. In this paper, a novel method applying two separate and succeeding evolution steps are proposed.

3.1 Approaches to Increased Complexity Evolution

The main advantage of the method is that evolution is not performed in one operation on the complete evolvable hardware unit but rather in a bottom-up way. It may be looked at as a division of the *problem* domain. The challenge of the approach would be how to define the fitness functions for the lower level subsystems. Two alternatives seem possible:

– **Partitioned training vectors.** A first approach to incremental evolution is by partitioning the training vectors. For evolving a truth table - i.e. like those used in digital design, each separate output could be evolved separately. In this method, the fitness function is given explicitly as a subset of the complete fitness function.
– **Partitioned training set.** A second approach is to divide the training set into several subsets. This corresponds to the way humans learns: Learning to walk and learning to talk are two different learning tasks. The fitness function would have to be designed for each task individually and used together with a global fitness function, when the tasks are to be assembled. This may not be a trivial problem.

The benefits of applying the *increased complexity evolution* are several:

- Making the search space *simpler* by having the complexity of the problem to be evolved reduced for each subsystem.
- Making the search space *smaller* by having a shorter chromosome string. This is because the circuit is smaller.

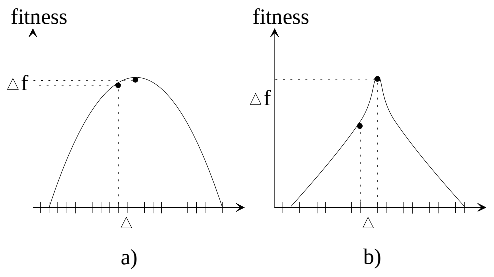

Fig. 1. Illustration of evolutionary search spaces.

Both of these two items concern the evolutionary search space as illustrated in Fig. 1. A simple search space is shown in Fig. 1a), while a more complex one is given in Fig. 1b). In the former figure, several instances would give a near maximum fitness while it becomes more important hitting the exact maximum in the latter. Thus, when both are having the same deviation △ to reach maxiumum fitness, there is a large difference in their corresponding fitness deviation △f. In the proposed method, the complexity of the problem is reduced by reducing the *amount* of information to be represented in a digital circuit.

The indices along the x-axis indicate the step size in the evolutionary search space. This is given by the chromosome length. A short string would imply a small number of indices compared to a longer string. The former would be more beneficial as this would reduce the search space. Thus, the goal of the *increased complexity evolution* approach is to *both* reduce the number of indices (by having a short chromosome string) as well as making the search space more smooth (by reducing the complexity of the problem to be evolved).

3.2 The Architecture of the Prosthetic Hand Controller

In this section, the proposed architecture for the controller is described. This includes the algorithm for undertaking the incremental evolution.

The architecture is illustrated in Fig. 2. 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

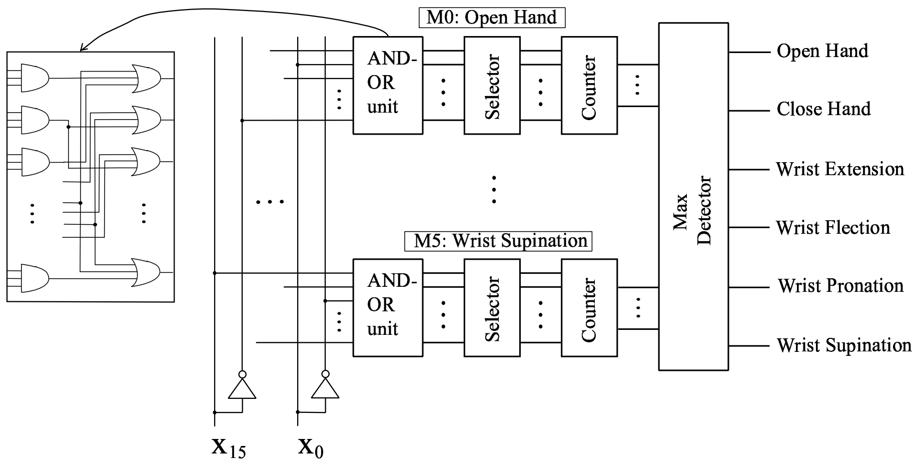

Fig. 2. The digital gate based architecture of the prosthetic hand controller.

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.

A scheme, based on using multi-input AND gates together with counters, has been proposed earlier [14]. However, the architecture proposed 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. A largest possible number of OR gates should be "1" for the 50 patterns corresponding to the motion the subsystem is set to respond to. For all other patterns, the number of gates outputting "1' should be as small as possible. That is, each subsystem should ideally respond only to the patterns for one specific prosthesis motion.
2. **Step 2 evolution.** Assemble the six AND-OR units into one system as seen in Fig. 2. The AND-OR units are now fixed and the *Selectors* are to be evolved in the assembled system. Which outputs, from each of the AND-OR units, to select for making the *total* performance highest possible are now determined. Thus, the fitness is measured using the same training set as in step 1 but the evaluation is now on the output of the Max Selector.
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 [12, 13]. In this paper, a less flexible AND-OR unit is used rather than the more general multi-layer gate array used in the earlier works. This is due to the initial experiments indicated that this restriction of flexibility was beneficial to make GA easier find better performing circuits. This is reasonable since only the connections are represented in the chromosome. In the architecture used earlier, the *function* of each gate was included as well.

This first step of evolution corresponds to the partitioned training vector approach presented in Section 3.1. In the following experiments, evolving subsystems, as described above, are compared to evolving a system directly. In the latter case, the system is evolved in one operation to classify all the six motions in the training set. The system consists of only one AND-OR unit with one gate output for each motion.

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 for GA. A larger number would have been beneficial for expressing more complex Boolean functions. However, the search space for GA 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:

Input 1 (5 bit)	Input 2 (5 bit)	(Input 3 (5 bit))	(Input 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 will be based on gate level. 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. 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.

3.3 Fitness Function

The fitness function is important for the performance of GA in 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:

$$F_1(m) = \frac{1}{s}\sum_{j=0}^{50m-1}\sum_{i=1}^{O} x + \sum_{j=50m}^{50m+49}\sum_{i=1}^{O} x + \frac{1}{s}\sum_{j=50m+50}^{P-1}\sum_{i=1}^{O} x \qquad \text{where } x = \begin{cases} 0 \text{ if } y_{i,j} \neq d_{m,j} \\ 1 \text{ if } y_{i,j} = d_{m,j} \end{cases}$$

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. This includes outputting “0” for input vectors for other motions.

The s is a scaling factor to implicit 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 result 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*[1] being “1” *equals* the *id* of the counter having the maximum output.

3.4 The GA Simulation

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 simple GA style – given by Goldberg [15], 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.

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 to n, the real performance should equal the simulation. Any spikes could be removed using registers in the circuit.

For each experiment presented in the next section, 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.

4 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.

[1] As mentioned earlier only *one* output bit is “1” for each training vector.

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: Fitness measure 16 (train)	2	51.3	60.7	54.8	64.3	71.3	67.5
C: Fitness measure 16 (test)	2	46	51.7	49	44.3	54.7	50
D: Fitness measure 16 (train)	4	59.3	71.3	65.5	70	76	73.4
D: Fitness measure 16 (test)	4	52.7	59.7	55.3	48.3	56.3	52.7
E: Direct evolution (train)	4	56.7	63.3	59.3	-	-	-
E: Direct evolution (test)	4	32.7	43.7	36.6	-	-	-

Table 1. The results of evolving the prosthetic hand controller in several different ways.

Table 1 shows the main results – in percentage correct classification. Several different ways of evolving the controller are included. The training set and test set performances are listed on separate lines in the table. The *fitness measure* – introduced in Section 3.3, will be discussed later in this section. The "# inp/gate" column includes the number of inputs for each gate in the AND-OR unit. 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). In average, there is an improvement in the performance for the latter. Thus, the proposed *increased complexity evolution* give rise to improved performances.

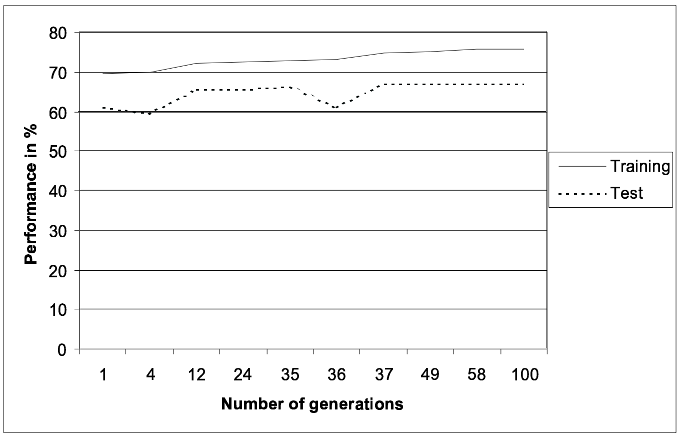

Fig. 3. Plot of the step 2 evolution of the best performing circuit.

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. Fig. 3 shows the step 2 evolution of this circuit. The training set performance is monotone increasing which is demanded by the fitness function. The test set performance is increasing with a couple of valleys. The circuit had a 60.7% test set performance after step 1 evolution[2]. 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. The results presented in this paper are from the first experiments undertaken. A lot more should be conducted to optimize parameters as well determining how to measure the fitness during evolution to secure a high test set performance in general.

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, see Fig. 4. The 16 outputs not used are included in the chromosome and have random values. That is, their value do not affect the fitness of the circuit. What is amazing about this is that 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!

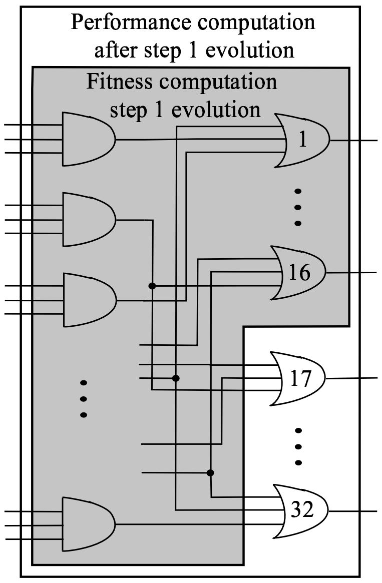
Performance computation after step 1 evolution
Fitness computation step 1 evolution
1
16
17
32

Fig. 4. A "fitness measure" equal to 16.

[2] Evaluated with all 32 outputs of the subsystems.

What is also interesting is that if the performance of the circuits in A is measured (after step 1 evolution) with only 16 outputs, the performance is not very impressive. Thus, "floating" outputs in the evolution substantially improve the performance – including the test set performance.

This may seem strange but has a reasonable explanation. 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. To explain why not B shows better performance, the description in Section 3.1 is appropriate. In this experiment, the chromosome is longer since in A the bits assigned to the 16 "floating" OR-gates are not used. Other numbers of "floating" OR gates (8 and 24) were tested but the results were best for 16.

A further improvement of A could be by introducing one *more* step of evolution. After finishing step 1 evolution, the AND-gates and the OR-gates covered by the fitness function F_1 could be fixed, and a new step of evolution could evolve the 16 "floating" OR gates. This could increase the performance, but not necessarily, since the step 2 evolution already removes the bad performing OR gates.

The C and D rows in the table contain the results when the gates in the AND-OR units each consists of two and four inputs, respectively. The lowest figures are for two input gates indicating that the architecture is too small to represent the given problem. Four inputs, on the other hand, could be too complex since having *three* input gates give a slightly better results.

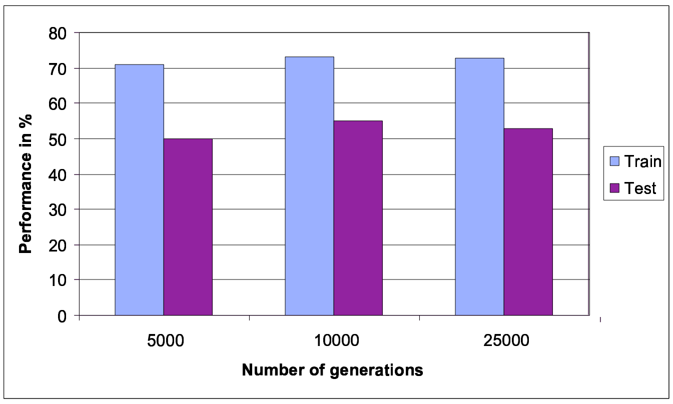

Fig. 5. Performance of three different numbers of generations in step 1 evolution (average of four runs).

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. One comparison of the step 1 evolution (each gate having three inputs) is included in Figure 5 and shows that the best average performance is achieved when evolving for 10,000 generations.

The circuits evolved with direct evolution (E) were undertaken for 100,000 generations[3]. 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. This is explained by having an architecture that is too small to provide good generalization. A larger one, on the other hand, would make the chromosome string longer with the problems introduced in Section 3.1. This once again emphasizes the importance of applying the *increased complexity evolution* scheme.

An interesting observation in the experiments is that the number of inputs to a gate is not always equal to the preselected value. Several inputs to *one* gate can be connected to the same source and thus, reducing the number of active inputs. Further, both inverted and non-inverted version of the same signal can be input to an AND gate. This makes the output becoming fixed to "0" and the OR gates connecting to it would then "reduce" its number of inputs. Such behavior illustrates the good adaptivity features within the AND-OR units.

According to these results there are several interesting topics for future work. The step 2 evolution could include a selection of *subsystem* for each motion *in addition* to the selector parameters. That is, for each motion a *set* of separately evolved subsystems could be available for the step 2 evolution to select among. In this way you would be able to assemble the best combination of many of the subsystems evolved in the step 1 evolution, rather than only six. Further, the architecture is a general one and should be applied to other problems within pattern recognition/classification.

5 Conclusions

In this paper, a new EHW architecture for pattern classification including incremental evolution has been proposed. The best circuit evolved shows a better generalization performance that what was obtained by artificial neural networks. The results illustrate that this is a promising approach to 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, Electrotechnical Laboratory, 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.

[3] This is more than six times 10,000 which were used in the other experiments.

References

1. R.N. Scott and P.A. Parker. Myoelectric prostheses: State of the art. *J. Med. Eng. Technol.*, 12:143–151, 1988.
2. S. Fuji. Development of prosthetic hand using adaptable control method for human characteristics. In *Proc. of Fifth International Conference on Intelligent Autonomous Systems.*, pages 360–367, 1998.
3. I. Kajitani and other. 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)*, 1999.
4. W-P. Lee et al. Learning complex robot behaviours by evolutionary computing with task decomposition. In Andreas Brink and John Demiris, editors, *Learning Robots: Proc. of 6th European Workshop, EWLR-6 Brighton.* Springer, 1997.
5. X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. In T. Higuchi et al., editors, *Evolvable Systems: From Biology to Hardware. First Int. Conf., ICES 96.* Springer-Verlag, 1997. Lecture Notes in Computer Science, vol. 1259.
6. E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallels*, 10(2), 1998. Paris: Hermes.
7. M. Iwata et al. A pattern recognition system using evolvable hardware. In *Proc. of Parallel Problem Solving from Nature IV (PPSN IV).* Springer Verlag, September 1996. Lecture Notes in Computer Science, vol. 1141.
8. P. Haddow and G. Tufte. An evolvable hardware FPGA for adaptive hardware. In *Proc. of Congress on Evolutionary Computation*, 2000.
9. M. Murakawa et al. Hardware evolution at function level. In *Proc. of Parallel Problem Solving from Nature IV (PPSNIV).* Springer Verlag, September 1996. Lecture Notes in Computer Science, vol. 1141.
10. J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* The MIT Press, 1994.
11. W.D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In *Physica D*, volume 42, pages 228–234. 1990.
12. J. Torresen. Scalable evolvable hardware applied to road image recognition. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware.* Silicon Valley, USA, July 2000.
13. J. Torresen. A divide-and-conquer approach to evolvable hardware. In M. Sipper et al., editors, *Evolvable Systems: From Biology to Hardware. Second Int. Conf., ICES 98*, pages 57–65. Springer-Verlag, 1998. Lecture Notes in Computer Science, vol. 1478.
14. M. Yasunaga et al. Genetic algorithm-based design methodology for pattern recognition hardware. In J. Miller et al., editors, *Evolvable Systems: From Biology to Hardware. Third Int. Conf., ICES 98.* Springer-Verlag, 2000. Lecture Notes in Computer Science, vol. 1801.
15. D. Goldberg. *Genetic Algorithms in search, optimization, and machine learning.* Addison Wesley, 1989.