Review

Neuromolecularware and its application to pattern recognition

Yo-Hsien Lin, Jong-Chen Chen *

Department of Information Management, National Yunlin University of Science and Technology, 123 University Road, Section 3, Douliou, Yunlin 64002, Taiwan, ROC

Abstract

Unlike computer systems, organisms have high adaptability in dealing with environmental changes or noise. The ability to evolve, self-organizing dynamics, and a closed structure–function relationship are the three principle features embedded in biological structures that provide great malleability to environmental change. Computer systems have fast processing speed for performing heavy computational tasks. One of the objectives in this research is to capture these three biological features and implement them onto a digital circuit. The proposed hardware (called neuromolecular hardware), is the integration of inter- and intraneuronal information processing applied to the pattern recognition problem domain. This approach was tested on the Quartus II system, a simulation tool for digital circuits. The experimental result showed good self-organizing capability in selecting significant bits for differentiating patterns and insignificant bits for tolerating noise. The proposed digital circuit also exhibited a closed structure–function relationship. This implied that this hardware embraced an adaptive fitness landscape that facilitated processing spatiotemporal information.

© 2008 Elsevier Ltd. All rights reserved.

Keywords: Evolution-friendliness; Self-organizing learning; Evolvable hardware; Pattern recognition

1. Introduction

Computer system, including software and hardware, provide effective programmability that allows us to explore various problem domains. However, as indicated by Conrad (1979), computer structures are brittle in that a slight program modification could inadvertently change the system functions or cause it to malfunction. Biological systems demonstrate better adaptability than computer systems. The ability to evolve, self-organizing dynamics, and a closed structure–function relationship provide organisms with great malleability (gradual transformation) in coping with environmental changes (i.e., noisy environments) and the ability to learn new survival strategies for uncertain environments (i.e., novel environments).

Evolvable hardware (EVH) is a new approach that attempts to apply evolutionary learning techniques in computer hardware (Sipper & Ronald, 2000; Yao, 1999). The idea is to combine the merits of biological systems and computer systems together, and hopefully create hardware with better adaptability. As pointed out by Gordon and Bentley (2002), EVH tends to have the following benefits: low hardware design costs, automatic designs, innovative designs, better adaptability in dealing with environmental changes, and good fault tolerant capability.

Yao indicated that there are two major approaches in the field of EVH: simulated evolution and reconfigurable hardware. The former applies evolutionary techniques to circuit synthesis while the latter dynamically and autonomously reconfigures hardware architectures for specific tasks. de Garis (1994, 1995, 1999) divided EVH into two categories: extrinsic and intrinsic. The former simulates evolution directly in the software while the latter accomplishes this with hardware. An early EVH study included evolutionary technique applications to designing primitive arithmetic and logic digital gates (Hemmi & Mizoguchi, 2000).
assembled groups of cellular automata-based neural net

de Garis (1994, 1999) developed an artificial brain that
satisfactory results in path finding (Chen & Conrad, 1994a, 1995, 1999; Miller, Kalaganova, Liptitskaya, & Job, 1999), a digital string generator (Zebulum, Pacheco, & Velhasco, 1996), and a digital filter (Miller, 1999). Besides digital circuit, the EVH studies on analogue circuits were discussed in (Iba & Ando, 2000; Koza, Bennet, Andre, & Keane, 1996; Lohn, 1999; Murakawa et al., 1999, 2003; Nam, Seo, Park, Park, & Kim, 2001).

In the application domains, Sipper et al. (1997) looked into the global behaviors of a swarm of fireflies and simulated their behaviors on an FPGA circuit, called the Firefly. Mange, Sipper, Stauffer, and Tempesti (2000) developed a timer (stopwatch) and a full watch (biowatch) on a digital circuit with a multilevel evolutionary architecture. Higuchi and his colleagues (Higuchi & Kajihara, 1999; Higuchi et al., 1999) worked on the development of evolvable hardware chips for various applications; an analog chip for cellular phones, a clock-timing chip for Giga hertz systems, a chip for autonomous reconfiguration control, a data compression chip, and a chip for controlling robotic hands and navigation. Murakawa et al. (1999) presented evolvable neural hardware for specific problem domains by reconfiguring network topology and changing the node functions. de Garis (1994, 1999) developed an artificial brain that assembled groups of cellular automata-based neural net modules for controlling the behavior of a kitten “robot”. Torresen (2001) designed an evolutionary digital circuit to control prosthetic hands. The majority of the EVH systems described above used computer software to implement the evolutionary process. Tufte and Haddow (1999) and Haddow and Tufte (2000) used a FPGA circuit to develop a hardware pipeline to implement the evolution process. A number of other researchers (de Garis, 1994, 1995, 1999; Mange et al., 2000; Sipper, 1997) used a group of cellular automata-type information processors to implement the evolution process in hardware.

There are two major problems encountered in the EVH field (Hemmi & Mizoguchi, 1996; Higuchi et al., 1993; Yao & Higuchi, 1999): scalability and noise tolerance. (Note that these two problems also exist in other domains). By scalability, we mean that the circuit design usually becomes extremely complicated as the number of gates involved increases. The proposed neuromolecular hardware is a central architecture that has rich dynamics that can be molded into the global behaviors of a swarm of fireflies and simulating their behaviors on an FPGA circuit, called the Firefly.

Moving on a fitness landscape can be compared to climbing on a mountainous region. Saddle points are critical as they provide mountain-climbers with intermediate points at which one can temporarily sustain and, in the meantime, look for better intermediate points. Without these intermediate points, one must move between two points at a time. The saddle point phenomenon is also critical for biological evolution. The chance for organisms to survive environmental changes is slim if survival requires multiple genetic mutations to occur simultaneously. With saddle points, several insignificant structural changes are allowed to occur at different times that may combined later problem in EVH. As indicated earlier, one of the objectives in this hardware is to capture some biological features and implement them on a digital circuit. Embracing these features provides great malleability for tolerating noise.

Section 2 discusses the biological features that are crucial for facilitating evolution. Section 3 introduces the neuromolecular hardware implementation onto digital circuits, including the evolutionary learning mechanisms. Section 4 shows the proposed digital circuit experimental results and the application domain that we used in this research and the model input–output links. The final section presents our concluding remarks.

2. Biological features friendly to evolution

We mentioned earlier that organisms have higher adaptability than computers. The Darwinian mechanism of variation and selection acting on a phylogenetic time scale is the essence of biological information processing. Variation allows an organism to be generated from its parent(s) giving it the opportunity to search for a better solution. With low variation rates, explorative search tends to concentrate on comparatively small areas and is thus subject to be caught indefinitely within a local optimum. High mutation rates (Cox & Gibson, 1974) or recombination rates (Strobeck, Maynard Smith, & Charlesworth, 1976) may increase the chance of jumping out of a local optimum, however, this increases the difficulty of converging to a global optimum. It is absolutely not easy to find the appropriate rate for each search stage. However, if a system possesses a structure with fitness landscapes holding some degrees of smoothness, finding feasible solutions within a reasonable time may become much easier (Conrad, 1979; Conrad, Ebeling, & Volkenstein, 1992; Feistel & Ebeling, 1982; Rechenberg, 1973).

Generally speaking, the fitness landscape of a computer program consists of isolated peaks that are separated by wide, deep valleys. In such a landscape, the chance of moving from one peak to another is comparatively small (Conrad, 1990). In biological systems, the smoothness (gradualism) property is naturally represented in the close structure–function relationship. A slight change in an organism’s gene structure generally will not significantly alter its functions (or behaviors).
to trigger some major functional changes in the organism or generate new traits. Conrad called this the bootstrapping effect (Conrad, 1979, 1990). Adding more saddle points will in turn increase the number of climbable pathways to make an ascent. The chance of jumping out of local optima is thus increased.

However, it should be noted that in a fixed landscape the number of saddle points is constant and the pathways between any two points are unique. The only possibility for adding more pathways and saddle points is by increasing the dimensionality of the landscape (Conrad, 1979). Eigen, McCaskill, and Schuster (1988) indicated that a high dimension search space generally provided more ridges (i.e., more saddle points). That is, it provides more pathways that facilitate outliers to climb up and down among different hills. In the study of Lotka–Volterra systems, May (1973) mentioned that the instability of a system may increase as its complexity increases. That is, it has better chance of escaping from a local optimum. Conrad (1979, 1990) pointed out that in a dynamic system, the number of saddle points increases as the system complexity increases. He called this the extra dimensional bypass. Harvey (1992) remarked that augmenting the length of genotype might increase the dimensionality of the search space.

Redundancy, weak interactions, and compartmentation are three important features in biological systems that facilitate evolutionary learning (Conrad, 1979). By redundancy, we mean that some constituted components are superfluous and can be added or removed without altering the outputs (or functions). However, redundancy conceals certain costs for organisms. Redundancy does serve as a buffer against the effects of genetic variations on phenotypic traits and hence on the fitness values (Conrad, 1982, 1993). The buffering effects allow an organism to absorb genetic changes and yet wait for other mutations to join together to make a significant change in its phenotypic traits. By virtue of redundancy, several genetic mutations do not have to occur simultaneously. The relaxation of genetic changes in a strict manner provides more pathways to achieve tasks, which in turn might speed up the search time. In a nutshell, redundancy is a costly feature for an organism; but it may pay off in terms of facilitating evolutionary adaptation.

Weak interaction is the second crucial feature for evolution-friendliness. By weak interaction, we mean that the interactions among the constituted components of a system are slight and adding a component into (or removing it from) a system will not significantly alter its outputs (or functions). This allows a system to stabilize its current state (or to change slightly) in responding to structural changes. In addition to these two features, compartmentation is another important feature that facilitates evolution. Compartmentation serves to block off disturbances (ramification of perturbation) and reduces the effects of instability within certain boundaries (Conrad, 1990; Conrad et al., 1992; Wagner & Günter, 1995).

As indicated above, redundancy, compartmentation, and weak interactions are three indispensable features for evolution-friendliness. In some senses, the differences between them may not be that distinct. For example, multiple weak interactions may be considered as a form of redundancy (Conrad, 1990). Likewise, redundancy can also be regarded as a form of compartmentalization.

3. Architecture

Neuromolecular hardware is a multilevel neural architecture motivated from brain-like information processing. It includes two types of neurons. The first type of neuron possesses complex intraneuronal dynamics that capture the molecular mechanisms inside real neurons. Through evolutionary learning, they can be molded to be specific input–output transducers. The second type of neurons has hierarchical control capability that can affect the firing of other neurons directly or indirectly. Through evolutionary learning, these neurons can be used to manipulate a group of neurons in a complex spatiotemporal manner for specific tasks. The integration of these two types of neurons constitutes an open evolutionary architecture with rich dynamics. An earlier version of this hardware and its preliminary experimental results were reported (Chen & Chen, 2002). In this paper, we perform a series of experiments emphasizing on the study of its spatiotemporal processing capability. We note that the neuromolecular hardware has some analogy to the Embryonics, leaded by Sipper, Mange, and their colleagues (Chen et al., 2000, Sipper and Ruppin, 1997).

In this section, we explain the intraneuronal architecture that plays the role of integrating signals in space and time. We then introduce the interneuronal architecture that orchestrates groups of neurons for coherent tasks. We then discuss the hardware features that are friendly to evolution.

3.1. Intraneuronal architecture

A number of researchers believed that signal processing occurring on the cytoskeleton might directly or indirectly control neuronal activity (Hameroff & Watt, 1982, Liberman, Minina, & Golubtsov, 1975; Liberman, Minina, Shklovsky-Kordy, & Conrad, 1982a; Liberman, Minina, Shklovsky-Kordy, & Conrad, 1982b; Hameroff, 1987; Matsumoto, Tsukita, & Arai, 1989; Werbos, 1992). The cytoskeleton has three major types of components: microtubules, microfilaments, and intermediate filaments. These components might interact with each other through proteins that link them together (Griffith & Pollard, 1978, 1982; Pollard, Selden, & Maupin, 1984; Selden & Pollard, 1983). However, the real signal interaction mechanism among them is not presently well understood. The objective of the neuromolecular hardware is not to identify the precise nature of these signal interactions, but rather to capture the feature that the cytoskeleton serves as a signal integrating system. More importantly, it covers the three...
biological features indicated earlier that facilitate evolutionary learning.

The cytoskeleton is simulated with $8 \times 8$ processing units (PUs). Each PU has eight neighboring PUs. The neighbors of an edge PU are determined in a wraparound fashion. For example, as shown in Fig. 1, the neighbors of the PU at location $(8,8)$ are the ones at $(7,7), (7,8), (7,1), (1,1), (1,8), (1,7),$ and $(8,7)$. Each PU has the same architecture, but may function differently by altering the memory bits that control its input/output (referred to as readin and readout enzymes, respectively), component type (to be referred to as C1-, C2-, or C3-type component), and connections with other PUs (referred to as a microtubule associated protein, MAP). Evolutionary learning occurs through the modification of these bit values (to be discussed later in Section 4). Fig. 2 shows the conceptual architecture of a PU, including four control parts and four signal processing departments. Fig. 3 shows the implementation of a PU on digital circuits. The following explains how to implement signal initiation, transmission, integration, and degradation on the cytoskeleton of a neuron (currently, the $8 \times 8$ PUs).

### 3.1.1. Signal initiation

Each PU has two possible input sources. The first is the external stimuli linked with its application domains. The second source is the signals sending from its eight neighboring PUs. Our assumption is that a PU will be directly activated by an external stimulus. For each signal sending from its neighboring PUs (through MAP), it will change the current state of a PU. We note that some specific combinations of these signals in space and time may turn a PU into a highly activated state. A new cytoskeletal signal is thus initiated when a PU is activated (to be explained in the next section).

In the present implementation, there are four possible PU types: C1, C2, C3, and none. The first three represent different types of cytoskeletal components for transmitting signals (i.e., different signal flows) whereas the last one represents that there is not any component. In Fig. 4, a signal from a C1-, C3-, and C2-type neighboring PU is labeled as S, W, and I respectively.

### 3.1.2. Signal transmission

The following explains how to implement signal transmission the $8 \times 8$ PUs. Our somewhat arbitrary assumption is that signal transmission is only allowed to occur on the PUs belonging to the same type. An activated PU will activate its neighboring PU of the same type at its next time step, which in turn activates its neighboring PU at following next time step. This process continues as long as there is a neighboring PU belongs to the same type. To assure unidirectional signal transmission, an activated PU will enter a refractory state. We note that a PU in the refractory state will ignore any stimuli till the refractory period is over. This would prevent a signal bouncing between two neighboring PUs. A refractory PU will then go back to the quiescent state. Fig. 5 gives two examples of signal transmission.

The hardware design is that the connection input department plays the role of regulating the signal flow from its neighboring PUs to the integration department. A switch controlled by the interrupt department (label in Fig. 2) is used to turn the signal flow on or off. In each
clock cycle, the interrupt department (Fig. 6) sends a high voltage signal to the switch, representing that the integration department is ready to take any neighboring signals. However, the switch will be turned off if the integration department is in the state of processing a neighboring signal or in the refractory state. The delay output department controls the timing for turning on/off the signal flow. Fig. 7 shows the timing control for each PU-type.

3.1.3. Signal integration

Signal integration is one of the most important features used in this model for processing spatiotemporal signals. As mentioned earlier, there are three types of PUs for transmitting signals. Our assumption is that, to fire a neuron, at least two signals must rendezvous at some specific PUs (the ones with readout enzymes sitting at the same place) within a short period of time. A PU serves as a signal

Fig. 3. Detailed architecture of a PU (A: bridger; B: comparator; C: adder 3; D: regulator 5; E: multiplier; F: downcounter 4; G: divider, H: delay; I: interrupter; J: decoder 2–4).
integrator to combine different signals in space and time. To implement this feature, two hypotheses are used. The first is that each PU-type has different transmission speed. The second hypothesis is that the degrees of influence between two different PU types are different (Table 1). We note that an activated PU can affect the state of its neighboring PU through MAP (microtubule associated protein) linking them together. We also assume that the interactions between two neighboring PUs are asymmetrical. That is, the influence of one PU on the other is not the same as that of the latter on the former.

As indicated earlier, C1, C2, and C3 represent three different types of cytoskeletal components for transmitting signals. In the present implementation, a signal traveling along C1-type PUs has the slowest speed, but has the greatest (“strong”) degree of influence on the other two PU types. In contrast, a signal traveling along C3-type PUs has the fastest speed, but has the least (“weak”) degree of influence on the other two PU types. The speed and the degree of influence for a C2-type signal are between that of C1- and C3-type signals. (Note that the ratio of transmission speed for C1-, C2-, and C3-type PUs is 1:2:3.) The signal with the greatest degree of influence serves as the major signal flow in a neuron while the other two types of signals play the role of providing modulating effects on it.

To implement the above feature, we assume that a PU has six possible states: quiescent (q0), active with increasing levels of activity (q1, q2, q3, and q4), and refractory (q5). Certainly, the complexity of intraneuronal dynamics would be greater when more PU states are allowed. Correspondingly, it will significantly increase the hardware size. We note that six states are sufficient for present use. The following describes the transition rules for each PU.

A PU in the highly active state (q3 or q4) will return to the refractory state (q5) at its next update time, and then
go into the quiescent state \((q_0)\) at the following update time. We note that each PU-type has its own update time, complying with its transmission speed. To ensure unidirectional signal transmission, a PU in the refractory state is not affected by its neighboring PUs until its refractory period is over. As shown earlier in Fig. 6, the hardware implementation for the refractory control turns off the switch that regulates the signal flow from the input department to the integration department. The next state for a less active PU \((q_0, q_1, \text{ or } q_2)\) depends on the sum of all stimuli received from its active neighboring PUs (Fig. 8). If a PU receives no stimulus before its next update time, it will go into state \(q_1\) if it was in state \(q_2\) and enter state \(q_0\) if it was in \(q_1\).

Fig. 6. Interrupt department.

Fig. 7. Delay department.

Table 1

<table>
<thead>
<tr>
<th>Neighboring PU</th>
<th>Current PU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
</tr>
<tr>
<td>C1</td>
<td>S</td>
</tr>
<tr>
<td>C2</td>
<td>I</td>
</tr>
<tr>
<td>C3</td>
<td>W</td>
</tr>
</tbody>
</table>

S: strong; I: intermediate; W: weak.

Fig. 8. Transition rules of a PU. \(S_1, S_2, \text{ and } S_3\) indicate a signal from a highly activated C1-, C2-, and C3-type PU, respectively. For example, if C1-type PU in the state \(q_0\) receives an \(S_2\) signal it will enter the moderately activated state \(q_2\). If it then receives an \(S_3\) signal it will enter the more activated state \(q_3\).

(a) C1-type PU

(b) C2-type PU

(c) C3-type PU

Fig. 9 shows how to implement the integration department on digital circuits. For each signal sent to the integration department, the comparator is responsible for determining its degree of influence on a PU. The adder combines the signals from the comparator and then sends its outputs to the regulator. The latter serves as a recorder (0 represents state \(q_0\), 1 state \(q_1\), 2 state \(q_2\), 3 state \(q_3\), and 4 state \(q_4\)) that each signal from the former will cause its state change (When we turn off the regulator, it represents that a PU is in the refractory state, \(q_r\)). For a PU in the most highly activated state, \(q_4\), a neuron will fire if there is a readout enzyme sitting at the same site. If a neuron is not firing, an activated PU will continue signal transmission to its neighboring PUs (through MAP connections). For a PU in less active state \((q_1 \text{ or } q_2)\), it will go to the state \(q_1\) if it was in the state \(q_2\) at its next update time and enter the state \(q_0\) if it was in \(q_1\). Different PU types have different update times. Our hardware implementation comprises a multiplier, a countdown counter, and a divider. First, the multiplier magnifies the signals from the regulator, depending on its transmission speed, and sends its output to the countdown counter. For each clock, the countdown counter decreases by one. Finally, the divider restores the
3.2. Interneuronal architecture – orchestration learning

Orchestration is an adaptive process mediated by varying neurons in the assembly that selects appropriate combinations of neurons to complete specific tasks. Currently, we use 128 cytoskeletal neurons, divided into four comparable subnets. Using comparable subnets, we mean that the neurons in these subnets are similar to each other in terms of their interneuronal connections and intraneuronal structures. Neurons in different subnets having similar interneuronal connections and intraneuronal structures grouped into a bundle. Thus, these 128 cytoskeletal neurons can also be grouped into 32 bundles.

Two levels of reference neurons are used to manipulate these 32 bundles. The two levels form hierarchical control architecture, as shown in Fig. 10. The first is referred to as the low-level reference neurons that directly control the bundles of cytoskeletal neurons. There are 32 low-level reference neurons, each of them controlling a bundle. (We note that only the bundles activated by reference neurons are allowed to perform information processing.) The second level is referred to as the high-level reference neurons that play the role of grouping the low-level reference neurons. The activation of a high-level reference neuron will fire all of the low-level reference neurons that it controls, which in turn will activate some of the 32 bundles of cytoskeletal neurons (i.e., neurons in different subnets have similar intraneuronal structures). The connections among low-level reference neurons and cytoskeletal neurons are fixed. However, the connections between high-level reference neuron and low-level reference neuron layers are subjected to change during evolutionary learning. The above process is called orchestral learning.

3.3. Evolutionary learning

Five levels of evolutionary learning are allowed in this system. They are at the initiating signal-flow level (controlled by readin enzymes), responding to signal-flow level (controlled by readout enzymes), transmitting signal-flow level (controlled by cytoskeletal PUs), controlling PU-
interaction level (controlled by MAPs), and determining output behavior level (controlled by effectors). The first four levels are intraneuronal and occur inside cytoskeletal neurons, whereas the last level is interneuronal.

Four competitive subnets and one reproduction subnet are used. In each competitive subnet, the initial pattern for readin, readout, MAP, PU-type, and effector is randomly decided. Evolutionary learning at the intraneuronal level has three major steps. First, we evaluate the performance of each competitive subnet. Then we select the best-performing subnet as the reproduction subnet. Finally, the readout enzyme, readin enzyme, MAP, PU-type, and effector pattern in the reproduction subnet is copied (with variation) to each of these competitive subnets if the former shows better performance than the latter.

Evolutionary learning at the reference neuron level is also comprised of three steps. First, cytoskeletal neurons controlled by each high-level reference neuron (through low-level reference neurons) are activated in sequence for evaluating their performance. Secondly, the patterns of neural activities controlled by the best-performing reference neurons are copied to the lesser-performing reference neurons. Finally, the lesser-performing reference neurons control slight variations in the neural groups controlled by the best-performing reference neurons, assuming that some errors occur during the copy process.

In the current implementation, only one level is opened for learning at a time while the other levels are turned off. Each level is opened for 16 learning cycles. Our approach is to turn on each level in an alternating manner until the simulation is terminated. We note that the segregation in time described above does not mean that the fitness assigned to the reference neurons is independent of the properties of the cytoskeletal neurons. Evolutionary learning at the cytoskeletal neuron level alters the performance characteristics of the collection of neurons (or combination of bundles) that the reference neurons control. This alters the fitness of the collection and therefore the fitness of the reference neurons is independent of the properties of the cytoskeletal neurons. Evolutionary learning at the cytoskeletal neuron level alters the performance characteristics of the collection of neurons (or combination of bundles) that the reference neurons control. This alters the fitness of the collection and therefore the fitness of the reference neurons is independent of the properties of the cytoskeletal neurons. Evolutionary learning at the cytoskeletal neuron level alters the performance characteristics of the collection of neurons (or combination of bundles) that the reference neurons control. This alters the fitness of the collection and therefore the fitness of the reference neurons is independent of the properties of the cytoskeletal neurons.

3.4. Evolution-friendliness features in the ANM system

As indicated earlier, weak interaction, redundancy, and compartmentation are three important features for facilitating evolution. We have introduced the ANM system and its implementation on digital circuits. In this section, we like to go over the above three features in this system.

The ANM system has 128 cytoskeletal neurons, divided into either four comparable subnets or thirty-two bundles. (Note that a cytoskeletal neuron can be grouped into a subnet or a bundle.) At the orchestral learning level, only one of these subnets is required to find appropriate cytoskeletal neuron bundles to perform specific tasks. Even though the other three subnets seem to be redundant; however, they serve as scratch pads that allow variations to work on them. A variation might help in a later search at the intraneuronal learning level. Similarly, at the intraneuronal learning level only the cytoskeletal neuron bundles selected by the reference neurons are allowed to engage in information processing. The non-selected bundles serve as scratch pads that allow variations to work on them. A variation might help in a later search at the orchestral learning level. As we allow variations to act on one group to look for better results, the other groups vary continually to explore the search space. Our early study (Chen & Conrad, 1994b) showed that opening evolutionary learning at the intraneuronal and orchestral levels in an alternate manner generates synergistic effects. Miller and his colleagues (Miller et al., 2000) presented a similar idea applying redundancy features on circuit design. This feature in some sense can also be regarded as a specific kind of compartmentation. As noted above, in the course of learning, neurons are divided into two groups: active and inactive. This separation blocks any possible interference between these two groups. None of the variations on one group can generate disturbances in the other group. Thus, learning can proceed in a stepwise manner.

The features friendly to evolution in the ANM system are more obvious at the intraneuronal levels. Readin and readout enzymes play the role of controlling neuron inputs and outputs. Most of the enzymes can be added or removed without changing neuron input/output behaviors as they might be located where no signal arrives or no signal integration takes place. In such a case, these enzymes are redundant. Even though they seem to be useless at times, there is an advantage in waiting a newly initiated signal to activate them. In most cases, relocating an enzyme to a neighboring location could hold the set of patterns recognized by a neuron constant, but in general it might slightly advance or delay its firing times. Moreover, readin enzymes can be set at any PUs whereas readout enzymes are confined to C1-type PUs. In other words, signals can be initiated anywhere in the cytoskeleton; however, only selective combinations of signals can fire a neuron. This keeps a neuron from being oversensitive to noise (or turbulence), a sort of compartmentation.

As indicated earlier, there are three types of cytoskeletal signals. One of them is the major signal flow in a neuron while the other two interact weakly with it. Our early study (Chen & Conrad, 1994b) showed that weak interaction plays an important role in facilitating learning. MAP plays the role of turning on or off the interactions between them. Removing a MAP that links two PUs together will shut down their interactions, compartmentalizing them into two isolated regions. In addition, a MAP is redundant if there is no signal integration occurring or a readout enzyme does not exist. However, like readin and readout enzymes, these MAPs are indispensable as they retain the possibility of waiting for other future changes to achieve better results.
4. Experimental results

We verified our PU hardware implementation on digital circuits to ensure that they worked correctly. Several checkpoints were set inside a PU. They were placed at the outputs of the adder, the regulator, the down-counter, and the divider (see Fig. 9). This hardware was tested using the Quartus II system, a digital circuit simulation tool developed by the Altera Corporation (San Jose, CA). The simulation waveforms (spatiotemporal outputs) showed that they worked as expected.

After verifying the digital PU circuits, we then performed an experiment on a cytoskeletal neuron to determine the approximate output signal timing. As indicated earlier, in the present implementation a cytoskeletal neuron was comprised of 64 PUs, whose types and connections were randomly decided. Thirty-two neurons were generated, each with a different random seed. The input pattern for each neuron was also randomly assigned. Note that the processing times for C1-, C2-, and C3-type PUs were 120 ns, 140 ns, and 160 ns, respectively. We mentioned earlier that the 64 PUs for a neuron were set up in a wrap-around fashion. Here, we assumed that it might form a cycle if an intraneuronal signal transmitted more than 32 PUs (i.e., one of the 64 PUs). If a neuron did not fire within 5000 ns, there were two possible cases. The first was that all intraneuronal signals die out before firing a neuron. The second case was that the intraneuronal signals were cycling. The experimental results (Table 2) showed the firing times of these 32 neurons. Only six out of the thirty-two neurons did not fire within 5000 ns. For those firing neurons, their firing times started from 540 ns to 2140 ns, suggesting that the cytoskeletal neurons in this model possessed sufficient dynamics to support more complicated experiments.

The neuromocular hardware was applied to a pattern recognition domain. Our training set was comprised of 40 patterns, each consisting of 25 bits. Twenty-six out of these forty patterns were the capital letters of the English alphabet (A, B, C, ..., Z), ten are the numeric digits (0, 1, 2, ..., 9), and four were arrow signs (←, ↑, →, ↓). Even though these patterns are meaningful to humans, they are partially random to the system. These 40 patterns were equally divided into four groups: 1000 ns, 2000 ns, 3000 ns, and 4000 ns, representing four different types of the system’s response times. For each of the patterns in the 1000 ns group, we expected the ANM system to generate its first output at 1000 ns immediately after it starts processing the pattern. Similarly, we expected it to generate its first output at 2000 ns, 3000 ns, and 4000 ns after it starts processing each of the patterns in the 2000 ns, 3000 ns, and 4000 ns group, respectively. Note that the above setup in some senses can be compared to different types of robotic motion controls, such as Higuchi’s welding robot (Higuchi et al., 1996) and the myoelectric prosthetic hand (Higuchi et al., 1999). The objective was to train the system in a manner that for each pattern the system could generate an output at a time as close to its predetermined time as possible. The smaller the total clock difference (TCD) value, the greater the system fitness (see Eq. (1)). The difference was set at 250-clock cycles (5000 nanoseconds) if the system did not generate an output within 5000 nanoseconds. Fig. 11 shows the input–output interface of the system with neurons in the same bundle (i.e., comparable neurons in different subnets) receiving the same pattern of inputs. The cytoskeletal neurons were divided into four groups (1000 ns, 2000 ns, 3000 ns, 4000 ns), corresponding to the four different pattern groups in the training set (Table 3).

Table 2
Signal elapsed time

<table>
<thead>
<tr>
<th>Run</th>
<th>Elapsed time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>800</td>
</tr>
<tr>
<td>2</td>
<td>800</td>
</tr>
<tr>
<td>3</td>
<td>1880</td>
</tr>
<tr>
<td>4</td>
<td>1740</td>
</tr>
<tr>
<td>5</td>
<td>1080</td>
</tr>
<tr>
<td>6</td>
<td>2120</td>
</tr>
<tr>
<td>7</td>
<td>&gt;5000</td>
</tr>
<tr>
<td>8</td>
<td>940</td>
</tr>
<tr>
<td>9</td>
<td>2000</td>
</tr>
<tr>
<td>10</td>
<td>1640</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1360</td>
</tr>
<tr>
<td>13</td>
<td>1880</td>
</tr>
<tr>
<td>14</td>
<td>940</td>
</tr>
<tr>
<td>15</td>
<td>1780</td>
</tr>
<tr>
<td>16</td>
<td>&gt;5000</td>
</tr>
<tr>
<td>17</td>
<td>1480</td>
</tr>
<tr>
<td>18</td>
<td>1300</td>
</tr>
<tr>
<td>19</td>
<td>&gt;5000</td>
</tr>
<tr>
<td>20</td>
<td>1420</td>
</tr>
<tr>
<td>21</td>
<td>1580</td>
</tr>
<tr>
<td>22</td>
<td>1780</td>
</tr>
<tr>
<td>23</td>
<td>&gt;5000</td>
</tr>
<tr>
<td>24</td>
<td>2140</td>
</tr>
<tr>
<td>25</td>
<td>&gt;5000</td>
</tr>
<tr>
<td>26</td>
<td>1580</td>
</tr>
<tr>
<td>27</td>
<td>1080</td>
</tr>
<tr>
<td>28</td>
<td>940</td>
</tr>
<tr>
<td>29</td>
<td>1260</td>
</tr>
<tr>
<td>30</td>
<td>1860</td>
</tr>
<tr>
<td>31</td>
<td>540</td>
</tr>
<tr>
<td>32</td>
<td>&gt;5000</td>
</tr>
</tbody>
</table>

"..." indicated a neuron did not fire within 5000 ns.

Possible function relationship. The fourth experiment examined the noise tolerance capability. The third experiment looked into the hardware structure–function relationship. The second experiment examined the noise tolerance capability. The first investigated the information processing capability (i.e., pattern differentiation capability). The second experiment examined the noise tolerance capability. The third experiment looked into the hardware structure–function relationship. The fourth experiment examined the hardware self-organizing learning capability. Perform-
ing these experiments was very time-consuming. Generally, this is the major drawback of testing digital circuits using computer simulations. Our current implementation required more computational time, as insufficient I/O pins were available to simulate more than one neuron at a time. In this implementation the total number of I/O pins required for a cytoskeletal neuron is 1026 whereas the maximum number that the Quartus II system could offer was 1508 pins. We temporarily bypassed this problem by simulating one neuron at a time (i.e., importing digital circuits, performing computer simulation, and reporting results). The process was repeated until each of the 128 cytoskeletal neurons was simulated, constituting a learning cycle. For each input pattern, half an hour was required to process a cycle using a Pentium-IV PC. (Only 5 μs were required when implemented on real digital circuit.) About twenty hours was needed to process the 40 patterns in the training set.

4.1. Pattern differentiation capability

The experimental result showed that the TCD value decreased as learning proceeded (Fig. 12). It started from 6532 at cycle 1 and decreased to only 3 at cycle 378. That is, in the beginning, the clock difference was 163 (6532/40) clock difference per pattern over average. After 378 learning cycles, there were only three patterns (#12, #14, #17) in the training set whose firing times were different from their expected times by one clock (20 ns). For the remaining thirty-seven patterns, the system was able to perform input/output transformation correctly. We continued the simulation and terminated it at cycle 500 as the TCD value stayed the same. As indicated earlier, slightly more than half a day was needed to process the 40 training patterns in a learning cycle. The above experiment took about eight months of CPU time. We could run the system longer for better results (to reduce the TCD from three to two, one, or zero). However, another eight months, or even longer, would be needed to do this. At this point, we would prefer not to continue any further so that we could switch to other experiments such as noise tolerance capability, structural malleability, and self-organizing capability (to be described later).

4.2. Noise tolerance capability

The following experiment tested the system’s noise tolerance capability using the structure after 500 learning cycles. The system structure was unchanged with the input patterns gradually varied in space and time to observe the output changes. If the system performance degraded gracefully with the extent of the increase in input variations, this in part supported the system structure embracing some degrees of noise tolerance capability.

4.2.1. Noise in space

To generate noise in space, we made a copy of the training set but altered some bits during the copy process (changing a bit into “1” if it was “0” and into “0” if it was “1”). Six levels of variation (noise) were imposed during the copy process: 2.5%, 5%, 7.5%, 10%, 20%, and 30%. For example, at the 5% level of variation, we mean that each bit had a 5% possibility of being altered (one or two bits changed per pattern on average). For each level of variation, five test sets were generated (each with a different random seed).

<table>
<thead>
<tr>
<th>Pattern #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td>C</td>
<td>G</td>
<td>K</td>
<td>O</td>
<td>S</td>
<td>W</td>
<td>D</td>
<td>H</td>
<td>L</td>
<td>P</td>
<td>T</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Errors (clocks)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 3

| Pattern # | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| Symbol    | A | E | I | M | Q | U | Y |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Errors (clocks) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fig. 11. Input/output interface of comparable subnets.

Fig. 12. Learning curve.
The experimental result showed that the TCD value increased gradually as we augmented the noise levels. The system’s performance degraded gracefully as the degree of noise increased. As shown in Fig. 13, the TCD changed slightly as we imposed a 5% variation level on the patterns (one or two bit changes per pattern on average). Even when we increased the variation level to 20% (i.e., five bits changed per pattern on average), the system still demonstrated acceptable results. This implied that the system had good noise tolerance capability in dealing with noises in space.

4.2.2. Noise in time
In the previous experiments, for each input pattern, all bit signals were sent to cytoskeletal neurons at the same time. In this experiment these signals might be sent at different times. Six levels of delay were performed: 2.5%, 5%, 7.5%, 10%, 20%, and 30%. For each level of delay, we ran five times (each with a different random seed). At the 2.5% level of delay, each signal had 2.5% possibility of being sent with delay. The delay time was randomly chosen from 1 to 5 clocks.

As expected, the TCD increased when we imposed temporal noise on the training patterns. An interesting result was that the TCD value did not necessarily increase accordingly as we increased the levels of delay. For example, the TCD increased when we imposed 2.5% level of delay. However, it decreased by nearly zero as we increased the delay level to 5%. Similarly, the TCD increased when the level of delay increased from 5% to 20%, and then decreased when we imposed 30% level of delay. Roughly speaking, the TCD trend exhibited a “wave”-shaped curve (Fig. 14). Fig. 15 shows the spatial and temporal noise effects on the system performance. In this implementation, to fire a neuron, at least two signals must be integrated within a short period of time. Postponing any one of these signals would make signal integration impossible. However, signal integration became possible if both signals were postponed simultaneously, as they might integrate at later time. This might explain why the TCD value did not change significantly when we greatly increased the levels of delay.

To look into this issue further, we performed other experiments by lengthening the maximum delay time. We increased the delay time from 5 clocks (100 ns) to 15 clocks (300 ns), to 25 clocks (500 ns), to 35 clocks (700 ns), and to 45 clocks (900 ns). As in the previous experiment, six levels of delay were performed. For each of these levels, we ran five times (each with a different random seed). As shown in Fig. 16, the results held basically the same. That is, the TCD increased when we imposed temporal noise on the training patterns, but did not necessarily increase accordingly as we increased the levels of delay. In most cases the TCD values in these four experiments were slightly smaller than those in the previous experiment. This might be because increasing the extent and degree of the postponed signals made it feasible to adjust the signal timing, providing better fit to the assigned tasks. The above result implied that the system had good generalization capability in dealing with input changes (noises) in space and time.

4.2.3. Structural malleability
As noted earlier, one of our objectives was to construct a malleable computer architecture that could be trained through evolutionary learning for specific tasks. A close structure–function relationship is an important feature captured in this system. The following experiment examined this neuromolecular hardware feature. We compared the system outputs as we gradually modified its structure. If the outputs did not change rapidly for each structural alteration, this implicitly suggested that this hardware held a certain degree of close structure–function. If the outputs changed dramatically, this indicated that this hardware structure was brittle.
As in the previous experiment, the system structure was used after a long period of learning. We copied and slightly varied the structure after 500 learning cycles. Four levels of structural changes were possible. They were at the PU-type, MAP, readin, and readout levels. In the following experiments, we took turns varying each level of structural change one at a time. This allowed us to understand the significance of each parameter. We varied the structure at the PU-type level first and maintained the structure unchanged at the other levels (MAP, readin, and readout). In this case the test structure was exactly the same as the trained structure except for the variation altered at the PU-type level. This procedure was repeated for the other three parameters: readin, readout, and MAP. At each level of structural change, we performed six degrees of variation: 2.5%, 5%, 7.5%, 10%, 20%, and 30%. At the 2.5% degree of variation, each copy operation was subject to 2.5% possibility of mutation. Three runs were performed for each degree of variation (each with a different random seed).

The experimental results (Fig. 17) showed that the TCD value changed accordingly as we increased the degree of variation in the structure. Among the four levels of structural change, the system outputs (the TCD value) changed most significantly in response to structural changes at the PU-type level. The result was reasonable as the PUs served as the major medium for transmitting signals. Altering a PU-type would change the signal type, which in turn directly affected the neuron signal integration activity. In contrast, the system output was less sensitive to structural changes at the readin and readout levels. There were two possible reasons for this result. First, most of these enzymes were redundant in that only a very small number of them served as receiving inputs (controlled by readin enzymes) and sending outputs (controlled by readout enzymes) for a neuron. The other reason was that the training set was very small compared to the set of all possible 25-bit patterns (i.e., 40 compared to 225 patterns). In such a small set, it was not necessary to utilize every bit to differentiate patterns, implying that most of these 25 bits were redundant (see the experimental results in the next section). Between the above two extreme cases, structural changes at the MAP level showed a moderate degree of influence on the system outputs.

The above result was important for facilitating evolutionary learning as it demonstrated that the system possessed different levels of sensitivity to parameter changes (structural changes). Some parameter changes significantly altered the system outputs. This provided the system a better chance of jumping out of local optima during an evolutionary search. Some parameter changes did not have a direct influence on the system outputs. This allowed the system to stay at its current stage and wait for other future changes to look for better solutions. Evolutionary learning turned on at different levels, allowing the system to perform different search modes.

4.2.4. Self-organizing learning

Previous experimental results (Chen, 1998; Chen & Conrad, 1997) indicated that certain bit positions were more significant than others. Significant bits were those utilized by the system to differentiate patterns. Altering the values of these bits would dramatically affect the system outputs. In contrast, some bits were comparatively less significant as the system outputs showed no changes when they were altered. From the viewpoint of pattern differentiation, insignificant bits seemed to be superfluous. However, from the viewpoint of pattern generalization, these bits were indispensable because they allowed the system to tolerate noise. In a nutshell, significant bits were useful for pattern differentiation whereas insignificant bits were useful for pattern generalization. Different data sets possessed different significant and insignificant bits. The following experi-
ment investigated the system’s capability in determining significant and insignificant bits.

As above, the system was used after substantial learning. The structure remained constant, while the training set was varied in a systematical manner to test the significance of each bit. Training was implementing by copying the training set and then altering bit information at a specific position in each pattern to generate a test set. Each test set was exactly the same as the training set, apart from the altered bit at the selected position. For example, we uniformly “flipped” the first bit in each pattern to generate the first test set, holding the other bits unchanged (“flip” means that a bit is changed into “0” if it was “1” and into “1” if it was “0”). This process was repeated for the second, third bits, and so on. This yielded 25 test sets, each comprising 40 test patterns.

As shown in Fig. 18, the TCD value changed significantly when we altered the bits at positions 11, 18, 21, 23, and 24. These bits were significant as the system used them to differentiate patterns. In contrast, the remaining bits were less significant as their outputs did not show significant changes when altered. As shown in Table 4, the significant bits were at locations 11 and 24 in the 1000 ns group, at locations 1, 2, 25 in the 2000 ns group, at location 18 in the 3000 ns group, and at locations 21 and 23 in the 4000 ns group. None of these significant bits was repeated in two different groups. Each pattern group had its own significant bit position. This result supported the notion that the system was capable of determining significant bits for each pattern group, based on its pattern structure, in a self-organizing manner.

5. Conclusions

Organisms have high adaptability. Evolutionary learning is one of the mechanisms used by organisms to cope with environmental changes. Evolution-friendly and self-organizing learning are two indispensable features. The former provides organisms a better chance of jumping out of local optima, while the latter allows organisms to function in an autonomous manner. With these two features, perpetual evolutionary learning is thus possible.

As indicated earlier, redundancy, compartmentation, and weak interactions are three indispensable features that constitute a structure amenable to evolution (Conrad, 1979). To some readers, the ANM system’s structure may be too complicated to comprehend at first glance. It is absolutely not our intention to bring every possible piece together and then try to convince people that it works. Instead, we constructed an open system with rich dynamics that covers the above three features, an elaborate architecture that allows individual elements to cooperate with each other in a coherent manner. More importantly, this is a central architecture that can be linked to different problem domains by altering its input–output interfaces.

In this paper, we introduced the ANM system implementation on digital circuits, the neuromolecular hardware, and then discussed the inter- and intraneuronal structures that were friendly to evolution. The evolution friendly structures included the constituted components and their interactions. In addition to these features, vertical information processing (an interleaved learning between its intra- and interneuronal structures) plays an important role. The power of a vertical information processing system lies in the sum of the contributions made by each constituent component. More importantly, synergistic effects are generated that cannot be achieved by each individual element. Our previous experimental results (Chen, 1998; Chen & Conrad, 1994a) demonstrated synergy occurring among different vertical information processing levels.

To look into the evolution friendly features in this system, we compared the system input–output behaviors by gradually modifying the structure after substantial learning on the input set. Our experimental results showed that the system outputs changed gradually as the structure was altered. Keeping the system’s structure unchanged, we examined the outputs by gradually imposing noise into the input patterns in space and time. Our results showed that the system demonstrates good noise tolerance capability. The outputs degraded gracefully as the degree of noise was increased. An interesting finding is that the system outputs did not change accordingly as the degree of variation in the output timing was increased. Note that delaying a signal may alter the neuron firing activity. However, this may not be true when several signals are delayed simultaneously as these signals may integrate at a later time (Undoubtedly, this will delay its firing timing). The above results demonstrated that this system has good noise tolerance capability in dealing with spatiotemporal changes in its inputs, implying that it possesses an adaptive surface that facilitates evolutionary learning.

Self-organizing dynamics was the other important feature that organisms use to adapt to different environments or environmental changes. In the present domain (pattern

<table>
<thead>
<tr>
<th>Pattern group</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 ns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000 ns</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000 ns</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000 ns</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4 Significant bits for each pattern group
recognition), the ANM system’s self-organizing capability was tested in selecting significant bits for differentiating patterns (pattern categorization) and insignificant bits for tolerating noise (pattern generalization). As mentioned earlier, different data sets possess different significant and insignificant bits. For successful adaptation, it is important to determine these bits in a self-organizing manner. Our experimental result demonstrated that this system has good self-organizing capability.

Evolvable hardware is a novel, but exciting, field. It provides a bridge between biological systems and computer systems. The important difference between these two systems is that they possess different information processing styles (Conrad, 1979). A computer system is a stably rigid system and a biological system has great flexibility. The neuromolecular hardware is a float between these two extremes. Undoubtedly, there is still a lot of future work left to do to fully implement the proposed method on digital circuits. As expected, several technical problems are involved. We currently proceed along this line (to be reported later). The presentation of this paper might attract more people’s attention to this subject. Together, we will uncover more of the nature of evolutionary processing.

Acknowledgement

This research was supported in part by the Taiwan National Science Council (Grant NSC 95-3113-P-224-003).

References


