Genetic Algorithm for CNN Template Learning

Tibor Kozek, *Student Member, IEEE*, Tamás Roska, *Fellow, IEEE*, and Leon O. Chua, *Fellow, IEEE*

***Abstract*—A new learning algorithm for space invariant cellular neural networks (CNN's) is described. Learning is formulated as an optimization problem. Exploration of any specified domain of stable CNN's is possible by the current approach. Templates are derived using a genetic optimization algorithm. Details of the algorithm are discussed and several application results are shown. Using this algorithm propagation-type and gray-scale-output CNN's can also be designed.**

I. Introduction

Cellular neural networks [1]–[3] have found many applications [4], [5], among others, in image processing. CNN is now considered as the paradigm of cellular analog programmable multidimensional processing arrays with distributed logic and memory [6].

The question is, which arises every time CNN's are used to perform a given operation, what is the "program" of the network, i.e., in the simplest case, the template elements. A possible and perhaps the most general answer to this question is to design an algorithm which can derive the template for a given operation. In other neural network areas algorithms of this kind are referred to as learning.

This paper presents a learning algorithm which can be applied in a wide problem domain. Previous results [7]–[9] were restricted to binary output and the stability of the network was assumed. The basic method they were following was to set up a system of inequalities which provided the desired output to be a stable equilibrium point. By solving the system of inequalities a template was gained, which hopefully worked and was robust. In [9] templates were generated by means of a unimodal function which provided robustness in case of binary output transformation, and unique result in the parameter space. A necessary condition in deriving properly working templates is that the desired output is a stable equilibrium point. But the solution of the system of inequalities does not guarantee that there are no other local minima in the state space where the state transition might stop before reaching the desired output. These methods gave some useful templates if the initial condition was not too far from the desired output. Thus it was only possible to create templates with local dynamics.

Manuscript received July 7, 1992; revised manuscript received November 17, 1992. This work is supported by the joint grant INT 90-01336 of the National Science Foundation and the Hungarian Academy of Sciences. This paper was recommended by Associate Editor J. A. Nossek.

T. Kozek is with the Department of Electrical Engineering, University of California, Berkeley, CA 94720, and the Computer and Automation Institute of the Hungarian Academy of Sciences and the Technical University of Budapest, Hungary.

T. Roska is a visiting scholar at the Department of Electrical Engineering, University of California, Berkeley, CA 94720, permanently with the Computer and Automation Institute of the Hungarian Academy of Sciences.

L. O. Chua is with the Department of Electrical Engineering, University of California, Berkeley, CA 94720.

IEEE Log Number 9208947.

To derive reliably working templates, we also have to ensure that the transient finally reaches the desired output from the initial state. (To provide proper convergence the trajectory of the transient has to be considered.) The simplest way to accomplish this is to evaluate a template according to the difference of its settled output and the desired output. Minimizing this quantity by changing the template element values allows us to derive templates for a given operation.

This input–output approach offers a flexible description of CNN's and makes possible to learn propagating and gray-scale-output templates, but in return, the resulting cost function is difficult to minimize. If the stability of the network is not guaranteed, the network may oscillate or be chaotic. This means that the cost function based on the transient behavior of the CNN will be noisy. Another difficulty arises if the cost function is not differentiable. In addition, it may have multiple, separate, local, or even global minima. What still makes this unfriendly cost function useful is the genetic optimization algorithm which is able to cope with these types of functions. It can find global minima even in noisy and discontinuous search spaces without using differential information about the cost function. We have found that by using genetic algorithms adapted properly, some classes of problems can be solved reliably and with fast convergence.

In the next section a short introduction to genetic algorithms is given, then its application to template learning is described. Simulation results using our program with the genetic template learning algorithm are presented in Section III.

Application of genetic algorithms to template learning, design of their parameters, and the summary of the experiences (including propagating type templates) are our main results presented here.

II. Genetic Algorithms

For those not familiar with genetic algorithms a brief introduction is given in this section. For a more detailed description please consult [10].

2.1. *What is a Genetic Algorithm?*

Genetic algorithms (GA's) are stochastic optimization algorithms that were originally motivated by the mechanisms of natural selection and genetics, and have proven to be effective in a number of applications. A "classical" genetic algorithm has the following basic properties:

- it works with a binary encoding of the parameter set, (not the parameters themselves);

```
   parents 1:  0  0  1  0  1  1  0  1|1  0  0  0
           2:  1  0  1  1  0  0  1  0|0  1  0  1
                          crossing site ⇕
 offspring 1:  0  0  1  0  1  1  0  1|0  1  0  1
           2:  1  0  1  1  0  0  1  0|1  0  0  0
```

Fig. 1. The one-point crossover operator.

- it searches from a population of points, (not from a single point);
- it uses only the cost function values in the optimization, (no derivatives or other auxiliary knowledge); and
- it uses probabilistic transition rules.

What might make a genetic algorithm attractive is its simplicity and the fact that its applicability is not limited by restrictive assumptions about the search space (continuity, unimodality, existence of derivatives, etc.). Despite their relative simplicity, GA's outperform any random search because they can exploit information cumulated during the evolution of the search. Calculus based methods are inevitably superior in the problem domain where they can be used, but GA's provide a robust search in discontinuous and multimodal noisy search spaces.

2.2. Genetic Search Mechanism

Because genetic algorithms are rooted in both natural genetics and computer science, their terminology mixes natural and artificial expressions. The scope of GA's is global, as they use a population of binary strings—called chromosomes—to explore the search space. Each chromosome encode a point in the parameter space, i.e., a possible solution of the problem to be solved. These binary strings are *evaluated* through a "fitness" function (a kind of objective or cost function) which contains all the information about the problem. Evaluation means that the performance of each possible solution is determined and the fitness value of the corresponding chromosome is calculated accordingly. The better the solution encoded by a chromosome, the higher the fitness. The genetic algorithm then tries to improve the fitness of the population by combining information contained in high fitness chromosomes.

The search evolves through the subsequent generations of binary chromosomes. Each generation produces the next one by means of probabilistic operators. These operators ensure that the best members of the population will survive, and their information content is preserved and combined to generate even better offspring. In the simplest GA a new generation is created by *three basic operators*. First, chromosomes are *selected* for reproduction with a probability proportional to their fitness. This simple mechanism assures that the most successful ones will produce the next generation. Then the selected chromosomes are mated randomly and each couple produces two children by *crossover* and *mutation* reproduction operators. Crossover means exchange of substrings between two parent chromosomes combining valuable information of the parents. The simplest crossover operator is the one-point crossover where first a crossing site is selected with uniform probability over the chromosome length, then the corresponding strings are exchanged as shown in Fig. 1.

1. Initialize randomly a population of chromosomes.
2. Evaluate each chromosome in the population.
3. *Create new chromosomes by mating current chromosomes;* apply selection, crossover and mutation operators in reproduction.
4. Evaluate the new chromosomes and insert them into the population by *replacing old ones.*
5. If time is up or a good solution is achieved stop and return, as result, the best chromosome of the population, else go to 3.

Fig. 2. Abstract description of a genetic algorithm.

Population after random initialization		Generation #10		Generation #20	
chromosomes	$g(\cdot)$	chromosomes	$g(\cdot)$	chromosomes	$g(\cdot)$
00111110001110100111:	12.00	10111001010111010101:	11.98	00101001100110110100:	0.00
01011010000101110000:	17.02	00110001010011100100:	0.00	10010011111001100101:	11.90
11100011101110110011:	12.06	10010001111001010001:	11.99	00110011111011100100:	0.00
00111001100010011101:	11.96	10101101101110110011:	12.08	00010011110011110100:	0.00
11100101001011100111:	12.01	00110011111001000111:	12.12	00011111110101100100:	0.00
10101011110101010001:	12.07	11101111000010100001:	12.09	10110001101011100101:	12.08
00101111011010100010:	21.42	11101011100000010001:	12.09	00110011110010110101:	12.00
11101101011101101010:	20.60	10100010011010010001:	11.89	00011001100011010000:	0.00
00100100110001110000:	21.12	10101011000001010111:	12.04	00010011110011100100:	0.00
00000011110001010011:	12.15	10010000111110110001:	12.04	10010001110010110101:	11.98
10100000010010010101:	12.02	00011111111111010111:	12.08	00010011110010110100:	0.00
00111011101000110000:	*23.65*	*10110001010000110101:*	*11.98*	10010001110111000101:	11.96
10100011000000110101:	11.94	10010001011010010001:	11.93	00011001010011100101:	11.94
00011110110001001101:	12.00	10100011110010010001:	11.93	00011011111111000000:	17.54
01110101111001010111:	12.04	00111010011010000100:	0.00	00011111110011010001:	11.98
10001000101111110101:	11.98	10100011110011110001:	12.10	00110011100011010000:	0.00
00101001000010011100:	*11.82*	*11001000101011101111:*	*17.20*	*10110011111010110100:*	*0.00*
10101011100001111111:	17.66	00111001000010010100:	28.00	10110011010001100100:	0.00
11010001101010111110:	21.21	00111001101110111101:	21.09	00010011111111100100:	0.00
10010110011111011011:	12.00	10010001010011100011:	11.97	00010011111011100100:	0.00
Population average:	14.79		12.33		5.07

Fig. 3. Evolution of the genetic algorithm minimizing the cost function $g(\cdot)$. Starting from a population of arbitrary strings better and better chromosomes are generated. The function $g(\cdot)$ is multimodal, all the chromosomes having zero cost encode possible good solutions of the problem.

Mutation maintains diversity in the string population by flipping an arbitrary bit in the chromosomes with a given probability that is generally low. This operator results in a random walk in the parameter space and introduces new information into the evolution process which might have been lost with a premature convergence of the algorithm.

An abstract description of a genetic algorithm using these components is given in Fig. 2.

What links this algorithm to a particular problem is the *parameter encoding* and the *evaluation function.* As it is discussed later encoding of the parameters is crucial and varies from problem to problem. Unfortunately, there is no general method to find an optimal one.

If the algorithm works well, unexceptional chromosomes of the initial population are replaced by better and better ones as shown in Fig. 3. Through this process of simulated evolution finally some highly evolved solution to the problem will be produced. How it works is explained in the next section.

2.3. How Does It Work?

To understand the mechanism of genetic algorithms the notion of similarity templates or *schemata* has to be introduced. Since every bit in the binary encoding of a possible solution captures some information about the problem, high performance chromosomes are similar in the sense that they contain ones and zeros in particular locations which encode high performance features. Schemata describe these similarities in binary strings by means of the ternary alphabet $\{0, 1, *\}$.

string 1 :	0	0	1	0	1	1	0	1
string 2 :	1	0	1	1	0	1	1	0
schema :	*	0	1	*	*	1	*	*

Fig. 4. Example of a schema contained in both binary strings.

Numbers 0 or 1 at a specific location denote that there is a fixed binary value in the string. The $*$ symbol means "don't care" and matches either 0 or 1 at the position it stands. The schema in Fig. 4 is contained in both string 1 and string 2.

Two important properties of schemata are used to describe and classify string similarities. The *order* of a schema S, denoted by $o(S)$, is simply the number of fixed positions in the string. For the schema in the example above $o(*01{*}{*}1{*}{*}) = 3$. Another characteristic property of a schema is its *defining length*. For a schema S the defining length $\delta(S)$ is the distance between the first and last specified positions. In the previous example $\delta(*01**1**) = 6 - 2 = 4$.

Performance of a single chromosome is measured by its fitness. Similarly, a particular schema is characterized by the average fitness of chromosomes containing it. How the search evolves depends on the effect of reproduction operators on schemata contained in a population of chromosomes.

Using uniform random selection chromosomes are selected with a probability proportional to their fitness. This means that schemata whose fitness is greater than the population average will be found in an increasing number of chromosomes, while the number of below-average schemata representatives will decrease. More specifically, if the fitness of a particular schema remains above or below the population average with a certain quantity, the rate of survival or decay of the schema is exponential.

Effects of the one-point crossover operator on a particular schema depend on its defining length. A schema survives if the crossing site falls outside its defining length, but will not necessarily be disrupted by the crossover operator, since it is possible that both parents contain the schema. All possible crossing sites can be chosen with the same probability, therefore the shorter the defining length, the higher the survival probability of the schema.

Mutation alters a position of a chromosome with a given, usually small, probability. A schema survives if all of the specified positions remain unchanged. Since mutations are statistically independent, survival is more likely when the order of the schema is low.

Summarizing these effects we can state that short, low-order above-average schemata will have exponentially increasing number of representatives in the subsequent generations. This property has a special importance in designing GA applications. A coding should be chosen so that *short, low-order schemata are relevant to the underlying problem.* Since this is essential in the evolution process coding greatly affects performance of the resulting algorithm.

A binary string of length l contains 2^l schemata, since at every location of the schema a $*$ or the corresponding bit of the binary string can stand. The number of schemata contained in the whole population of size n is between 2^l and $n \cdot 2^l$ depending on the diversity of the population. Although the actual number can never reach the upper bound, considering short defining length schemata occur in many of the strings, this is a very large number. From the above conclusion we know, that not all of these schemata are processed effectively, since the crossover operator destroys those of relatively long defining length. It can be shown [10] that there is a lower bound on the number of schemata processed at the desirable exponential survival or decay rate. For a population of n chromosomes the number of effectively processed schemata is at least $O(n^3)$. It means that though in each generation we perform only computation proportional to the size of the population, we get effective processing of approximately n^3 schemata. This unique feature of genetic algorithms is called *implicit parallelism* and explains how this mechanism works.

2.4. Improved Techniques

Performance of the simplest GA can be enhanced by using advanced operators and some domain specific knowledge built into them. It is also possible to use real number representation and combine the algorithm with other optimization methods. In these cases the algorithm is said to be a hybrid GA. A hybrid algorithm can be more efficient since it is adapted to the specific problem, but the underlying theoretical background is still not firm and we can rely only on the power of similarities and successful examples.

III. GA Based Template Learning

3.1. The Problem of Template Learning

Operations performed by an asymptotically stable CNN can be described by a triplet of signal arrays, e.g., images: the input, initial state, and settled output of the network mapped into gray scale values of pixels. The problem of learning is to find the template to an operation given by the image triplet. The template to be found should define the dynamics such that the desired output is a stable equilibrium point in the state space and the initial state is in its basin of attractions.

We can fulfill both requirements by considering the trajectory of the transient. The simplest way to attain this is to create a cost function which compares the desired output to the result of the transient defined by a given template and the input and initial state from the image triplet. The following formula gives such a function:

$$g(p) = \sum_{i=1}^{k} (y_i^d - y_i(\infty))^2 \qquad (1)$$

where p denotes the parameter vector, i.e., the template, k is the size of the network (the number of cells), y_i^d is the value of the ith pixel of the desired output and $y_i(\infty)$ stands for the corresponding pixel of the settled output. $g(p) = 0$ if the result of template p is identical to the desired output and gives a quadratically increasing distance elsewhere. By using $g(\,.\,)$ as a cost function the problem of learning can be formulated as an optimization problem. Applying genetic algorithms $g(\,.\,)$ is minimized indirectly: its value is mapped into a fitness value $f(\,.\,)$ which is to be maximized.

3.2. Optimization by GA

To apply genetic algorithms, templates are coded as binary chromosomes and are evaluated by the above cost function. Although the cost function is a quadratic distance in the state space of the network, its form and properties vary from problem to problem in the parameter space.

The simplest way to consider stability is to check symmetry [1] or positive cell-linking [11] of the A template. Some equivalent transformations resulting positive cell-linking templates can also be checked [12]. But minimizing $g(p)$ with a genetic algorithm we do not have to apply constraints to assure stability because unstable trajectories will result low fitness values; therefore, any specified domain of stable CNN's can be explored. It implies that propagating mode templates can also be captured. Since the underlying network is nonlinear and can oscillate and be chaotic, the cost function can be discontinuous and noisy if the CNN is not stable. The shape of the function and the number of local and global minima depend on the problem also. The genetic algorithm used for minimization of $g(p)$ has to be effective in each cases. To develop robust genetic search a number of algorithms using different operators and parameter encodings were tested on a variety of template learning problems. Here we describe these operators and give a comparison on their performance later.

Coding methods: Coding here means how to represent a template as a binary string. Since the ultimate goal of template design is to build analog VLSI CNN chips, feasibility of parameters has to be considered. Constraints coming from technology restrict normalized parameter values to be in the approximate range of $[-5,\ 5]$. Using a fixed length binary representation of the template elements this constraint can be simply enforced. Relative accuracy of template elements is also limited by the technology and is around 10^{-2}. Therefore higher resolution is not required. For these reasons each real template value was coded with ten bits providing the range $[-5,\ 5]$ and the resolution 0.01.

As it was indicated above encoding of parameters into binary chromosomes is crucial to the performance of the algorithm. Three coding methods have been used.

- *Standard coding:* A widely used coding is to concatenate simply the binary strings representing each real parameter value as shown in Fig. 6. This method works well until the dimensionality of the parameter space is low. But in larger spaces the chromosomes are longer and the performance of the genetic operators decreases rapidly. This happens because high-performance schemata become longer and the algorithm cannot process them at the desirable exponential rate, since the crossover operator destroys them with high probability.
- *Enhanced coding:* A better encoding is to put the corresponding bits of the real parameters next to each other. Having n parameters, this results the following string:

$$(p_{1,\,1},\, p_{1,\,2}, \cdots, p_{1,\,n},\ p_{2,\,1},\, p_{2,\,2}, \cdots, p_{2,\,n}, \cdots, p_{10,\,1},\, p_{10,\,2}, \cdots, p_{10,\,n}) \quad (2)$$

  where $p_{i,\,j}$ is the ith bit of the jth parameter. This representation is much less sensitive to the number of

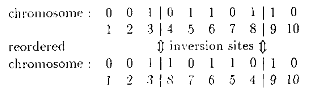

chromosome : 0 0 1 | 0 1 1 0 1 | 1 0
1 2 3 | 4 5 6 7 8 | 9 10
reordered ⇕ inversion sites ⇕
chromosome : 0 0 1 | 1 0 1 1 0 | 1 0
1 2 3 | 8 7 6 5 4 | 9 10

Fig. 5. Reordering by inversion operator.

  parameters, since the sign and ratio of the template values are more relevant to the CNN than the magnitude of the parameters. An example of this coding method is also shown in Fig. 6.
- *Reordering by inversion:* There is a way to use the genetic algorithm itself to improve the encoding in parallel with the evolution process. In the above two coding methods the meaning of a bit of a chromosome is determined by its location. Here the correspondence is given by an integer that refers to the location of a given bit in the standard encoding. After evaluating a generation, each chromosome is *reordered by inversion*. This means that between two arbitrary positions the substring is inverted with the corresponding position substring as shown in Fig. 5. Reordering changes the defining length of schemata while they themselves remain intact.

The mechanism provides that bit values retain their original meaning regardless of their position. This operator clearly has no effect on the fitness of the chromosomes, but enhances encoding as the search evolves. There is no rigorous explanation at the moment how this mechanism contributes to the performance of the genetic algorithm. Encoding of the same template with extended representation is shown in Fig. 6.

Reproduction strategies:

- *Nonoverlapping populations.* Each population of size n generates n new chromosomes and this offspring replaces entirely the old population. This method was also combined with elitist strategy which means that one or a few of the best chromosomes are preserved and inserted into the new generation.
- *Steady-state reproduction* means that only a few individuals are replaced in the population to produce the next generation. Using this strategy chromosomes having fitness value below the population average were replaced. In this case the above-average part of a generation overlaps the next one.

An example of the results of these reproduction strategies is shown in Fig. 7. In both cases new generations were constructed not to have duplicate chromosomes, since they do not contain additional information. This was carried out by the crossover and mutation operators.

Selection operator: For selection a modified version of the simple selection mechanism has been used. High variance of the random selection slows down the evolution process. To avoid this, above-average chromosomes are selected automatically and their fitness is reduced by the amount of the average. After having no more chromosomes with fitness higher than the population average, parents are selected randomly with a probability proportional to their fitness. An example of this mechanism is shown in Fig. 8.

$$\text{The template:}\qquad \mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathbf{B} = 0, \qquad \mathbf{I} = 0,$$

```
Standard coding:            0 0 0 0 0 1 1 0 0 1|0 0 0 0 1 1 0 0 1 0|1 1 1 1 1 0 0 1 1 1

Enhanced coding:            0 0 1|0 0 1|0 0 1|0 0 1|0 1 1|1 1 0|1 0 0|0 0 1|0 1 1|1 0 1

Extended representation
before inversion:           0 0 0 0 0 1 1 0 0 1|0 0 0 0 1 1 0 0 1 0|1 1 1 1 1 0 0 1 1 1
                            1 2 3 4 5 6 7 8 9 10|11 12 13 14 15 16 17 18 19 20|21 22 23 24 25 26 27 28 29 30
```

Fig. 6. Different binary encodings of the connected component detector template. Only the nonzero template elements are represented. Vertical lines in the strings show logical boundaries.

Current generation:		Next generation produced by Nonoverlapping strategy:		Steady-state reproduction:	
chromosome	fitness	chromosome	fitness	chromosome	fitness
10001000001110110101:	38.04	00001111100010001101:	31.76	10001000001110110101:	38.04
00111110101111100111:	21.98	00111000110001010001:	28.14	00111110101111100111:	21.98
10100011101000110001:	12.26	00111110101110110011:	21.03	00111011000000110100:	28.00
00111011000000110100:	28.00	00000011101111100111:	38.92	00000011101110110011:	26.06
11100011110001010011:	12.15	11110011101000110001:	17.33	01101010000101110000:	31.45
00000011101110110011:	26.06	10100101111001111111:	32.08	10000001010010010101:	28.31
00001110110001001101:	11.92	10000000010010010101:	11.75	00111000000010011100:	34.13
00111001000010011100:	10.62	10110110011111011011:	28.62	10111000111001111111:	15.04
01101010000101110000:	31.45	00001011100001010111:	35.14	00100101110001010001:	27.63
10011011110101010001:	12.09	10101000001110110101:	10.35	00000011101110010111:	35.17
Population average:	20.46				

Fig. 7. Reproduction by nonoverlapping and steady-state strategy. In case of steady-state reproduction above-average chromosomes are not replaced.

Old population:			Population after deterministic selection:			Selected parent pool:	
#	chromosome	fitness	#	chromosome	fitness	#	chromosome
(1)	10100000010010010101:	21.84	(1)	10100000010010010101:	21.84	(2)	00111011101000110000
(2)	00111011101000110000:	32.18	(2)	00111011101000110000:	5.06	(3)	10100011000000110101
(3)	10100011000000110101:	37.02	(3)	10100011000000110101:	9.90	(6)	10001000101111110101
(4)	00011110110001001101:	13.27	(4)	00011110110001001101:	13.27	(9)	11010001101010111110
(5)	01110101111001010111:	25.40	(5)	01110101111001010111:	25.40	(10)	10010110011111011011
(6)	10001000101111110101:	38.12	(6)	10001000101111110101:	11.00	(8)	10101011100001111111
(7)	00101001000010011100:	16.67	(7)	00101001000010011100:	16.67	(6)	10001000101111110101
(8)	10101011100001111111:	22.83	(8)	10101011100001111111:	22.83	(5)	01110101111001010111
(9)	11010001101010111110:	35.72	(9)	11010001101010111110:	8.60	(3)	10100011000000110101
(10)	10010110011111011011:	28.17	(10)	10010110011111011011:	1.05	(1)	10100000010010010101
	Population average:	27.12					

Fig. 8. Example of the selection mechanism. Chromosomes above the line in the right column were selected deterministically, those below the line with a probability proportional to their fitness in the middle column.

```
 parents 1:  0  0 | 1  0  1  1  0  1 | 1  0  0  0
         2:  1  0 | 1  1  0  0  1  0 | 0  1  0  1
                  ⇕   crossing sites  ⇕
offspring 1: 0  0 | 1  1  0  0  1  0 | 1  0  0  0
         2:  1  0 | 1  0  1  1  0  1 | 0  1  0  1
```

Fig. 9. The two-point crossover operator.

```
    parents 1:  0  0  1  0  1  1  0  1  1  0  0  0
            2:  1  0  1  1  0  0  1  0  0  1  0  1
               ____________________________________
random string:  0  1  1  0  0  1  1  1  0  1  0  1

  offspring 1:  0  0  1  0  1  0  1  0  1  1  0  1
            2:  1  0  1  1  0  1  0  1  0  0  0  0
```

Fig. 10. The random crossover operator.

Crossover operators:

- *One-point crossover* was applied in a number of cases exactly the same way as shown on Fig. 1.
- *Two-point crossover* shown in Fig. 9 is similar to the previous one, but in this case two crossing sites are selected and substrings between the crossing sites are exchanged. This method has the same properties as already described, but can combine certain schemata which the one-point version cannot.
- *Random crossover* is shown in Fig. 10. This operator combines two chromosomes according to a random binary string. At every location the corresponding bits of the parents are exchanged if the random string contains a 1 at that location. If the random bit is 0 no exchange takes place.

All crossover operators were realized in a way that they generate new chromosomes different from those already in the new generation. Difference from both parents, i.e., actual exchange of information is also required. If the new chromosomes created by the operator do not meet these conditions, different crossing sites are tried. If it still fails to create different chromosomes, new, random mating is selected. This method generates duplicate free offspring with a high probability, but if it still fails, mutation can alter duplicates.

Mutation operators: Mutation is performed in the standard way, namely a bit of a chromosome is altered with a specified, low probability. A modified version have also been used which deterministically modifies duplicate chromosomes in the population by flipping an arbitrary bit.

Evaluation: The evaluation function takes a chromosome (i.e., a template) and returns a fitness value associated with it. Each time it is invoked the transient of the CNN governed by the encoded template is calculated. Calculation is performed by integration of the state equation. Computation stops if the network reached an equilibrium point or after a prescribed number of iterations. Given the desired output and the result of the transient, the cost $g(p)$ is calculated. The value of the cost is then mapped into a fitness value so as to fit into the genetic algorithm. There are a number of methods to perform this mapping known as fitness techniques. (See Fig. 11.) We used the following ones.

- *Direct mapping:* This technique simply transforms $g(p)$ to be minimized into a fitness value f_i which is to be maximized by the genetic algorithm. The possible largest difference between two images is proportional to the image size, since the pixel values are in the range $[-1, +1]$. The fitness of chromosome p_i is calculated by $f_i = \text{maxdiff} - g(p_i)$, where maxdiff is the possible largest difference ($\text{maxdiff} = 4 \cdot \text{image\_size}$).
- *Windowing:* Zero or a constant minimum fitness value is assigned to the worst chromosome. Then each member of the population is credited with an increased fitness proportional to the amount its cost is less than the cost of the worst one.

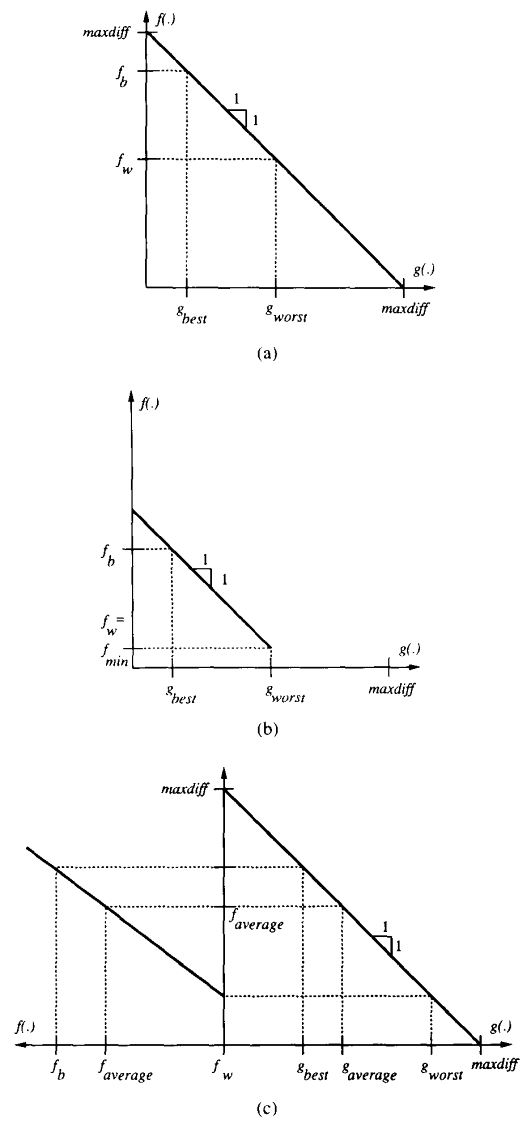


Fig. 11. Fitness techniques: (a) direct mapping, (b) windowing, (c) linear scaling.

- *Linear scaling:* First a raw fitness is calculated using direct mapping then a linear function maps the raw fitness into f_i such that the average cost of the population is mapped into the average fitness and zero fitness or a minimum amount is assigned to the chromosome with the maximal cost.

Direct mapping is the most straightforward method of transforming cost values into fitness values. But as the search evolves the population becomes more uniform and the fitness difference between good and bad chromosomes becomes smaller. Since this difference governs the survival or decay of the chromosomes the performance of the algorithm decreases. Windowing and linear scaling provide two alternative methods to overcome this difficulty.

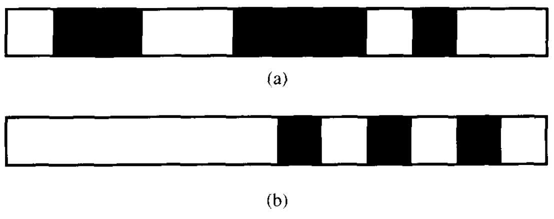


Fig. 12. Training set for horizontal connected component detection: (a) initial state, and (b) desired output.

Feedback template:			Feedforward template:			Current constant:
0.00	0.00	0.00	0.00	0.00	0.00	
1.95	3.04	-2.07	0.00	0.00	0.00	0.07
0.00	0.00	0.00	0.00	0.00	0.00	

Fig. 13. Horizontal CCD template produced by the simple genetic algorithm.

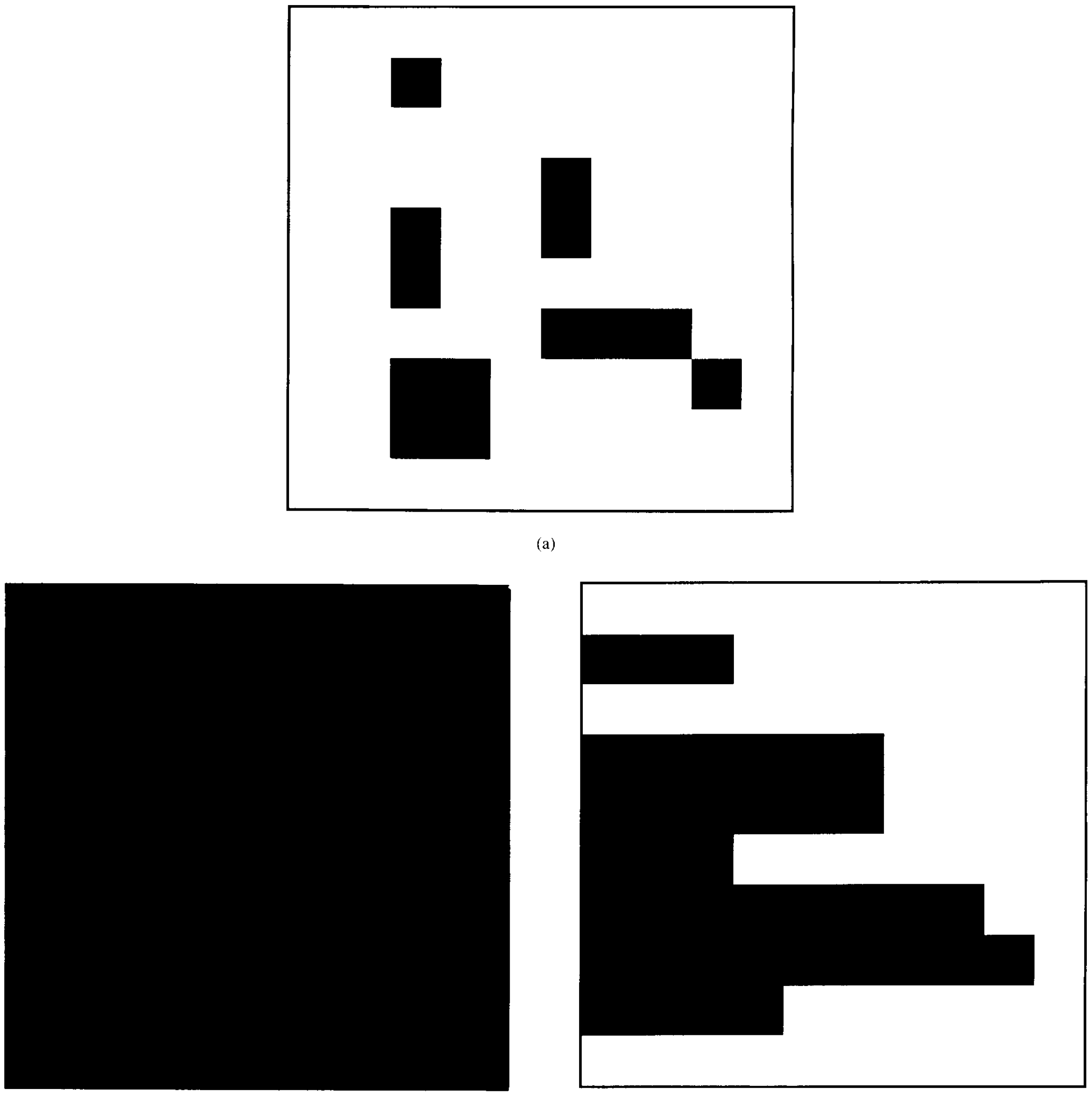
3.3. Simulation Results

The template learning program has been implemented in C code. The genetic algorithm evaluates every chromosome by computing the transient of the CNN defined by the chromosome. Since the computation starts always from the same initial state and with the same input values, in case of a given template the state equation is integrated every time along the same trajectory in the state space of the network. It is possible that an unstable trajectory is close to the desired output when the integration stops. This results high fitness value although the corresponding template is not stable. If a fixed integration length were used, it would be possible that the algorithm converges to a chaotic or periodic template which reaches the desired output right at the moment when the integration stops. To overcome this problem, random integration length is used. This results in a noisy cost function if the network is not stable what the genetic algorithm can cope with.

Many times there is some *a priori* knowledge about the template to be learned. Additional information can increase the performance of the algorithm on a given problem. To facilitate the use of knowledge of this kind an additional input is used for the learning program. This input is a *format template* which specifies which template elements are free parameters and which can be set to a fixed value (e.g., 0). There is also a possibility to specify a template element to be equal to another one or its inverse. Both, setting a template element to an explicit value and applying equality constraints, decrease the dimensionality of the search space, hence increase the speed of the search.

There are a number of parameters in a genetic algorithm which have to be specified. Unless otherwise noted the following parameters were used in the simulations: population size = 100, bit mutation rate = 0.005, nonoverlapping populations, two-point crossover, direct mapping as fitness technique and enhanced coding method.

Example 1: The first example shows that even the simplest genetic algorithm is very effective in solving relatively simple learning problems. In this case one-point crossover and standard coding were applied for a population of size 20. In Fig. 12 the training set for horizontal connected component detection (CCD) [13] is shown.



(a)

(b) (c)

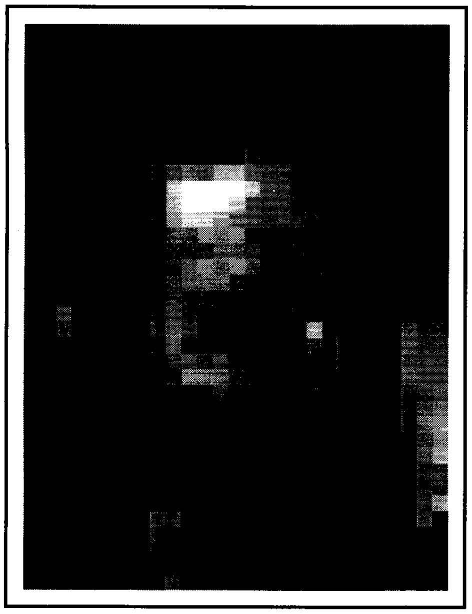
Fig. 14. Training set for shadow detector: (a) input, (b) initial space. (c) Desired output.

Feedback template:			Feedforward template:			Current constant:
0.00	0.00	0.00	0.00	0.00	0.00	
0.44	4.95	3.78	0.19	4.92	1.18	-0.05
0.00	0.00	0.00	0.00	0.00	0.00	

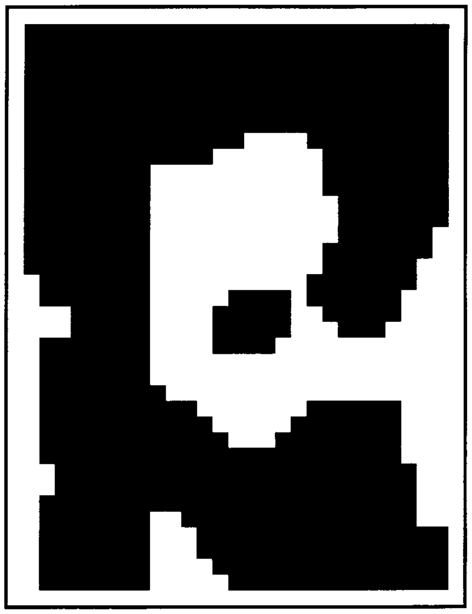
Fig. 15. Horizontal shadow detector.

The following reasoning gives an example how *a priori* knowledge can be used. The CCD is a propagating type template and its result is independent from the actual location of the connected components. Therefore, the location dependent input information has to be ignored by setting the feedforward template elements to zero and only the initial state of the network contains the input image. It is also clear that CCD operates in one direction, consecutively; in the case of horizontal CCD, feedback template elements connecting different rows can be set to zero as well. Altogether this means that there are four free parameters: three template elements in the central row of the feedback template and the constant current value, i.e., the format of the template is the following:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 \\ * & * & * \\ 0 & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{B} = 0, \qquad \boldsymbol{I} = *$$



(a)



(b)

Feedback template			Feedforward template			Current constant
0.19	1.54	0.19	0.00	0.00	0.00	
1.54	2.27	1.54	0.00	0.00	0.00	0.45
0.19	1.54	0.19	0.00	0.00	0.00	

Fig. 16. Averaging template: (a) initial state, (b) desired output and the template generated by the GA using linear scaling. (Since the feedforward template is zero, the input of the network is indifferent.)

where $*$ denotes free parameters coded in the chromosome and $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{I}$ denote the feedback template, feedforward template, and current constant, respectively. Using the simple GA after 10 generations the best chromosome of the population encodes the template which performs the desired operation (see Fig. 13).

Example 2: Another example for propagating type templates is the shadow detector [14]. Its training set is shown in Fig. 14.

Exploiting the horizontal propagating behavior of the operation the row-connecting template elements are set to zero. After 105 generations the template in Fig. 15 is produced.

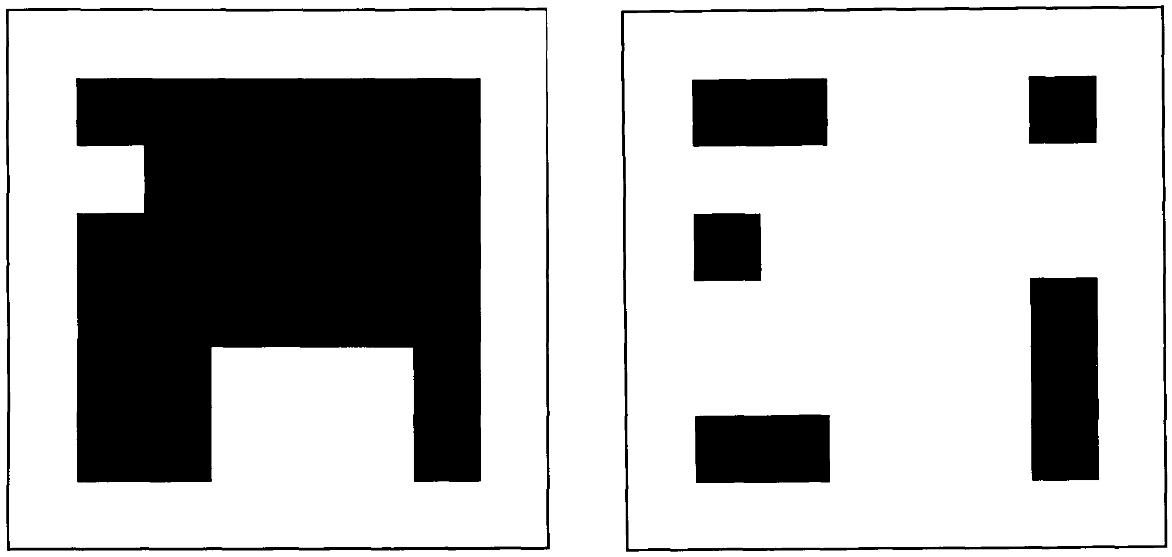
Example 3: As the search evolves the population becomes more uniform, the fitness values of the chromosomes are increasing and get closer to each other. Since reproduction is controlled by the fitness values, difference between higher and lower performance chromosomes in the population becomes smaller. Using direct mapping as fitness technique the speed of convergence decreases. This undesirable effect can be avoided by windowing or linear fitness scaling. In the case of the averaging template shown in Fig. 16 it took considerably shorter to reach a good solution using windowing or linear scaling.

In the previous examples, there were only a few free parameters in each template; that is, the dimensionality of the search space was low. It also means that the binary chromosomes were short and bits encoding important properties of the network (e.g., signs of template elements) were close to each other. In other words, with the standard encoding the length of schemata describing high-performance features of the design are proportional to the number of free parameters, i.e., dimensions of the search space. If the number of free parameters increases, these schemata will not survive with the desired exponential probability; consecutively, the convergence speed dramatically decreases and finally the algorithm may get stuck at a local minimum.

Example 4: The next example of a corner detector [2] template (Fig. 17) was not possible to learn with the standard coding method but gave the result shown in Fig. 17 using the enhanced coding.

It was also possible to learn the corner detection using reordering to find better and better coding in parallel with the evolution process, but this algorithm has proven to be inferior to the previous one.

Example 5: Different reproduction strategies were also tested. Elitism improves convergence in large populations (≥ 50) preserving the best chromosome in each iteration. In smaller populations it may cause premature convergence by dominance of a super individual. Steady-state reproduction usually does not work well in noisy search spaces. To



Feedback template			Feedforward template			Current constant
0.00	0.00	0.00	-0.57	-1.19	-0.57	
0.00	2.00	0.00	-1.19	3.30	-1.19	-3.45
0.00	0.00	0.00	-0.57	-1.19	-0.57	

Fig. 17. Convex corner detector: (a) input image and identical initial state, (b) desired output, and the template generated using enhanced coding.

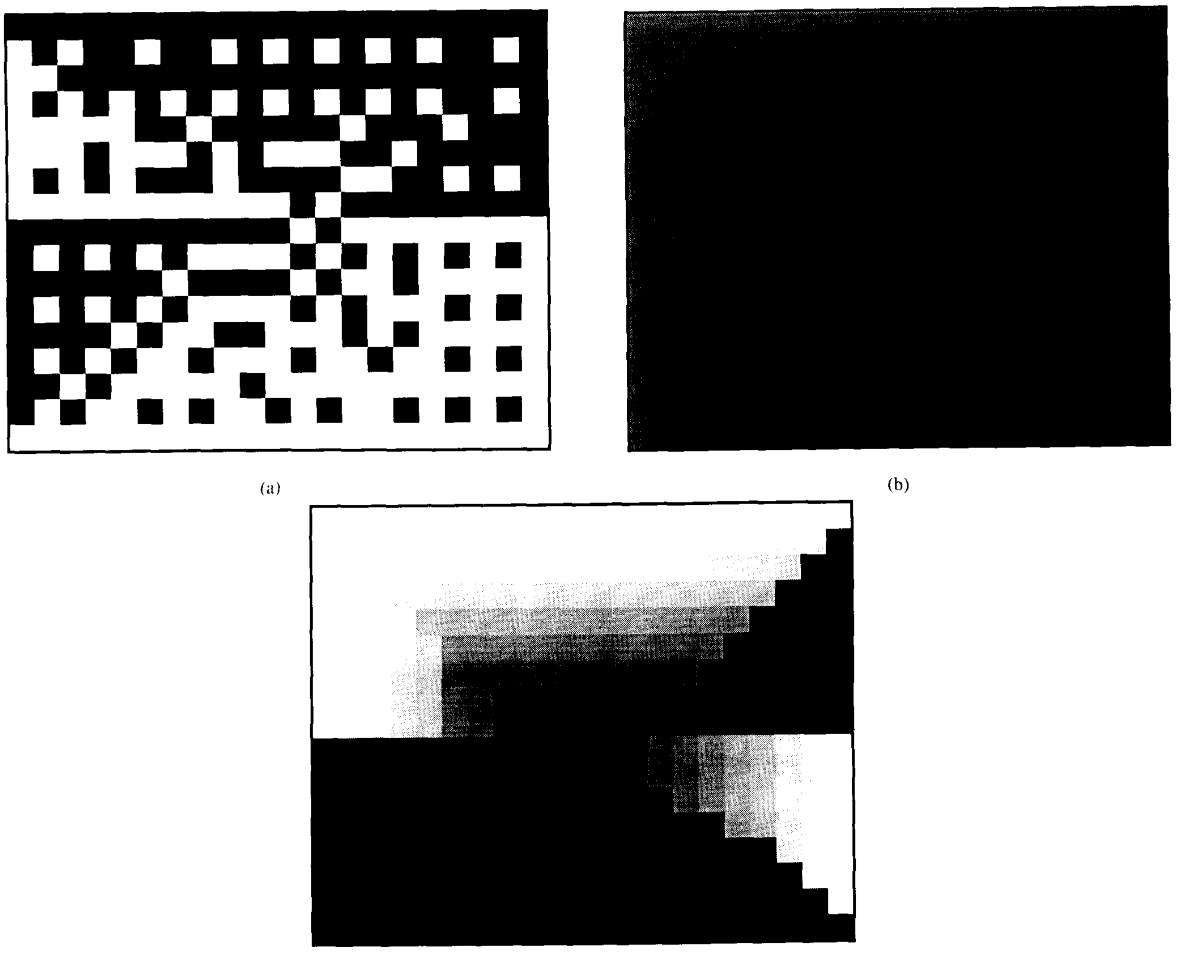
overcome this difficulty the whole population was evaluated in each iteration, not only the new chromosomes. This mechanism provided good performance in most cases but higher mutation rate was necessary to maintain the diversity of the population. The inverse halftoning [15] template in Fig. 18 was generated by this algorithm.

Another possibility to make the algorithm insensitive to the number of free parameters is to apply random crossover. This operator exchanges corresponding bits of the parent chromosomes at every location with the same probability ($P\{exchange\} = 0.5$). In case of one-point crossover a certain schema survives if the crossing site falls outside the schema. Hence short defining length schemata have higher survival probability. Since random crossover alters a schema uniformly at any location, its survival is independent from the defining length. This involves that the resulting algorithm will not be sensitive to the length of the chromosomes. Simulation results confirm that the genetic algorithm with random crossover has good overall performance. Although on simple problems it is inferior to most other algorithm variants because short, high performance schemata are destroyed more often, it performs better than any other when the number of free parameters is high.

IV. Conclusions

A new learning algorithm for CNN's based on genetic search was described. The whole domain of stable CNN's can be explored using this method. Templates were evaluated according to the transient behavior of the network they resulted. Performance of a template was determined by means of a quadratic difference between the desired output and the settled output of the CNN governed by the template. This difference was minimized using genetic optimization algorithms.

Even the simplest genetic algorithm could generate simple templates, but as the number of free parameters in the template increases its performance breaks down. Therefore, several genetic algorithm variants were tested on a wide range of template learning tasks to develop a more robust algorithm. Templates with symmetric feedback (e.g., average, convex corner detector) as well as propagating type (e.g., connected component detector, shadow detector) and gray-scale output templates (inverse halftoning) were generated successfully. The enhanced coding method with elitist strategy, two-point crossover, and windowing fitness technique was effective in simple and higher dimensional problems as well, but its performance was still sensitive to the size of the problem. Application of random crossover provided an algorithm whose reproduction properties are independent from the length of the binary chromosomes; therefore, its performance is not influenced by the number of free design parameters. Although on simple problems it is inferior to the algorithm mentioned above, as the dimensionality of the problem increases it out-performs any other algorithm variant.


(a)

(b)

(c)

Feedback template					Feedforward template					Current constant
0.00	0.00	0.00	0.00	0.00	0.02	0.06	0.11	0.06	0.02	
0.00	0.00	0.00	0.00	0.00	0.06	0.14	0.17	0.14	0.06	
0.00	0.00	0.00	0.00	0.00	0.11	0.17	0.21	0.17	0.11	0.00
0.00	0.00	0.00	0.00	0.00	0.06	0.14	0.17	0.14	0.06	
0.00	0.00	0.00	0.00	0.00	0.02	0.06	0.11	0.06	0.02	

Fig. 18. Inverse halftoning template and its training set: (a) input image. (b) initial state. (c) Desired output. In this example the generated template cannot perfectly reproduce the desired output (the corresponding cost value is not zero) since the halftoning process has already distorted the image.

Further performance enhancement can possibly be achieved by hybrid genetic algorithms, where using special operators more domain specific knowledge can be exploited. Moreover, generality of the approach allows learning of nonlinear and delay-type templates [3] with the same algorithm by parametrizing nonlinearities and delay factors.

References

[1] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1257–1272, 1988.

[2] ——, "Cellular neural networks: Application," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1273–1290, 1988.

[3] T. Roska and L. O. Chua, "Cellular neural networks with nonlinear and delay-type template elements," in *Proc. IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pp. 12–25, 1990.

[4] *Proc. 1990 IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, 1990.

[5] *Int. J. Circuit Theory and Applications*, July 1992.

[6] T. Roska and L. O. Chua, "CNN: Cellular analog programmable multidimensional processing array with distributed logic and memory," submitted for publication in *IEEE Trans. Circuits Syst.* (Preliminary version: Rep. DNS-2-1992, Computer and Automation Inst. of the Hung. Acad. Sci., Budapest).

[7] F. Zou, S. Schwarz, and J. A. Nossek, "Cellular neural network design using a learning algorithm," in *Proc. IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pp. 73–81, 1990.
[8] S. Schwarz and W. Mathis, "A design algorithm for cellular neural networks," in *Proc. 2nd Int. Conf. Microelectronics for Neural Networks*, pp. 53–59, 1991.
[9] P. Szolgay and T. Kozek, "Optical detection of layout errors of printed circuit boards using learned CNN templates," Report DNS-8-1991, Dual and Neural Computing Systems Res. Lab., Comp. Aut. Inst., Hung. Acad. Sci., Budapest, 1991.
[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.
[11] L. O. Chua and T. Roska, "Stability of a class of nonreciprocal cellular neural networks," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 1520–1527, 1990.
[12] L. O. Chua and C. W. Wu, "On the universe of stable cellular neural networks," ERL Memorandum UCB/ERL M91/31, University of California, Berkeley, 1991 (to appear in *Int. J. Circuit Theory and Applications*).
[13] T. Matsumoto, L. O. Chua, and H. Suzuki, "CNN cloning template: Connected component detector," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 633–635, 1990.
[14] ——, "CNN cloning template: Shadow detector," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 1070–1073, 1990.
[15] K. R. Crounse, T. Roska, and L. O. Chua, "Image halftoning with cellular neural networks," *Tech. Rep. UCB/ERL M91/106*, Univ. California, Berkeley, Electronics Research Laboratory, Nov. 1991.

Tibor Kozek graduated in electrical engineering from the Technical University of Budapest, Hungary, in 1992.

He is currently a graduate student at the Computer and Automation Institute of the Hungarian Academy of Sciences. His research interests include neural network learning algorithms, numerical simulation and application of cellular neural networks.

Tamás Roska (M'87–SM'90–F'92), for a photograph and biography, please see page 146 of the March issue of this TRANSACTIONS.

Leon O. Chua (S'60–M'62–SM'70–F'74), for a photograph and biography, please see page 156 of the March issue of this TRANSACTIONS.