Chapter 10

Template design tools

During the first years after the introduction of the CNN paradigm, many templates were designed by cut-and-try techniques, playing with a few nonzero template elements, and using a simulator to calculate the CNN dynamics. After a while, some systematic design methodologies emerged. Today several methods are available for generating CNN templates or algorithms, even for complex tasks.

10.1. Various design techniques

The main classes of design techniques are as follows.

- systematic methods for binary I/O function via Boolean description and decomposition techniques using uncoupled CNN (see Chapters 5, 6, 7)
- systematic methods for binary I/O function using coupled CNN (see also Chapter 12)
- global optimization techniques as parameter optimization
- genetic algorithms for designing the template elements / synaptic weights[1]
- matching with the spatially discrete representations of partial differential equations (PDEs)
- matching with some neuromorphic models of a living organism, typically the nervous system, in particular the visual pathway of vertebrates (see Chapter 16)
- fuzzy design techniques[2]
- neural network techniques[3]
- matching with existing 2D or 3D algorithms, including techniques in signal processing, telecommunications, adaptive control, nonlinear spatiotemporal dynamical systems, etc.

We have to emphasize, however, that in spite of the many design techniques, new methods are emerging day by day based on the intuition and skill of the designers. A good example for this is a recent method[4] using active waves applied for a while and

[1] e.g. T. Kozek, T. Roska, and L.O. Chua, Genetic algorithm for CNN template learning, IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications, (CAS-I), Vol.40, No. 6, pp. 392-402, 1993

[2] e.g. Cs. Rekeczky, A. Tahy, Z. Végh and T. Roska, CNN Based Spatio-temporal Nonlinear Filtering and Endocardial Boundary Detection in Echocardiography, Int. J. Circuit Theory and Applications - Special Issue: Theory, Design and Applications of Cellular Neural Networks: Part II: Design and Applications, (CTA Special Issue - II), Vol.27, No.1, pp. 171-207, 1999

[3] e.g. P. Földesy, L. Kék, Á. Zarándy, T. Roska and G. Bártfai, Fault Tolerant Design of Analogic CNN Templates and Algorithms – Part I: The Binary Output Case, IEEE Trans. on Circuits and Systems I: Special Issue on Bio-Inspired Processors and Cellular Neural Networks for Vision, Vol. 46, No. 2, pp. 312-322, 1999

[4] Cs. Rekeczky and L.O. Chua, Computing with Front Propagation: Active Contour and Skeleton Models in Continuous-Time CNN , Journal of VLSI Signal Processing, Special Issue: Spatiotemporal

combining / colliding with other waves, as well as a method in which a wave metric is used[5] for complex pattern recognition tasks.

In this chapter, referring to the results of Chapters 5, 6, and 7, we will demonstrate a systematic method for binary I/O functions. The outline of this design process is as follows (Figure 1). This process is supported by the template design and optimization program TEMPO (Appendix C).

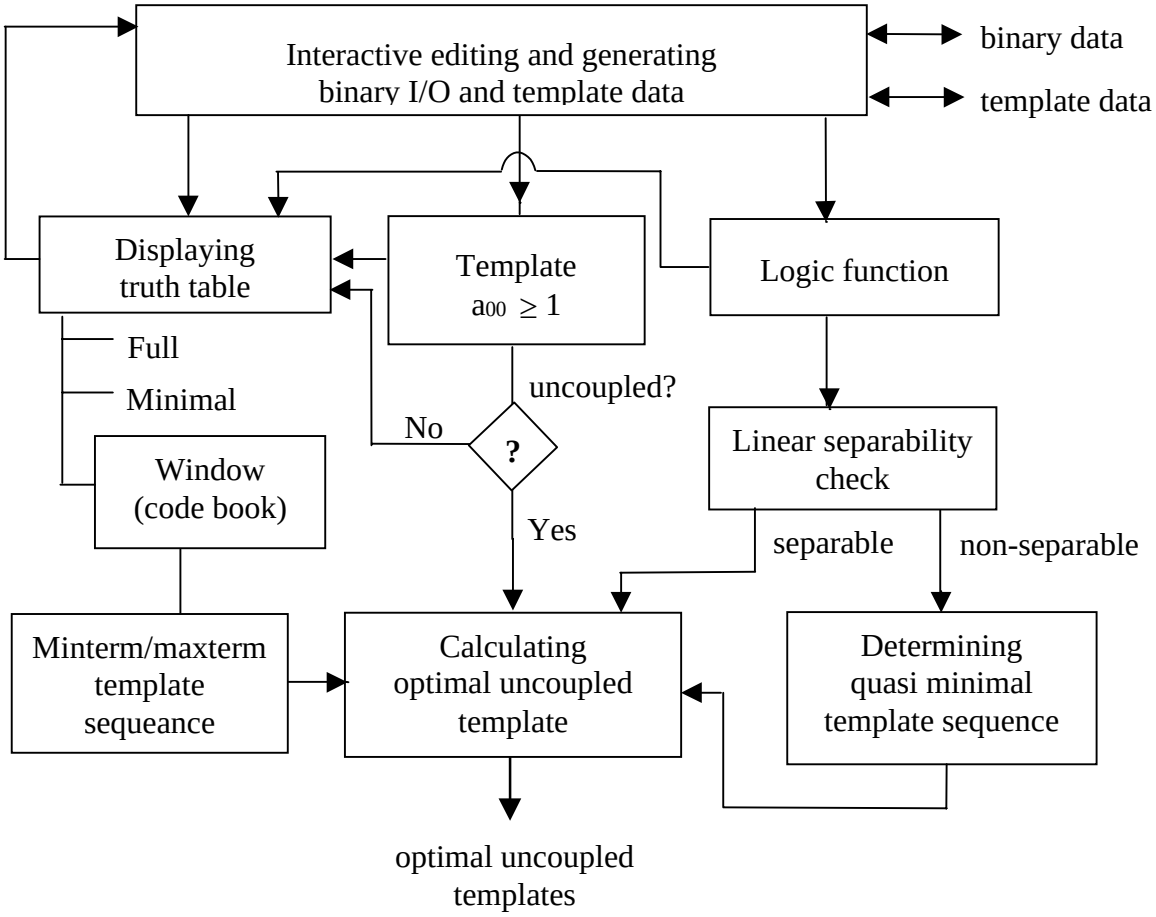


Fig.1. The outline of the binary I/O CNN template or template sequence design

Logic truth tables are given by a {0, 1} code (white and black), however, we can code binary data as TRUE(1), FALSE(-1) and DON'T CARE(0) as well.

When designing CNN templates to implement a given logic function F_k (.), we are typically using uncoupled templates with the following description and coding:

Signal Processing with Analogic CNN Visual Microprocessors, Vol.23. No.2/3. pp.373-402, Kluwer, 1999

[5] I. Szatmári, Cs. Rekeczky and T. Roska, A Nonlinear Wave Metric and its CNN Implementation for Object Classification, Journal of VLSI Signal Processing, Special Issue: Spatiotemporal Signal Processing with Analogic CNN Visual Microprocessors, Vol.23. No.2/3. pp.437-448, Kluwer, 1999.

$$A = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & a_{00}\geq 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \qquad B_k = \begin{array}{|c|c|c|}\hline w_{-1-1} & w_{-10} & w_{-11} \\ \hline w_{0-1} & w_{00} & w_{01} \\ \hline w_{1-1} & w_{10} & w_{11} \\ \hline \end{array} \qquad z \qquad (10.1)$$

In the design process we start with logic Boolean functions of 9 variables $F_k(u_1, u_2,\ldots, u_9)$, supposing a zero valued initial state, keeping in mind the convention

$$\begin{bmatrix} u_9 & u_8 & u_7 \\ u_6 & u_5 & u_4 \\ u_3 & u_2 & u_1 \end{bmatrix}$$

or with the cloning template (A,B,z) or the truth table, especially in its window (code book) form. The outcome of the design is an uncoupled cloning template with the parameters

$$a_{00}, b_1, b_2, b_3,\ldots b_9, z$$

keeping in mind the convention

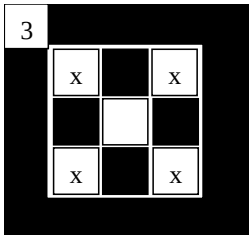
$$B_k = \begin{array}{|c|c|c|}\hline b_9 & b_8 & b_7 \\ \hline b_6 & b_5 & b_4 \\ \hline b_3 & b_2 & b_1 \\ \hline \end{array}$$

or the sequence of templates combined via some local logic functions implemented as a program on the CNN Universal Machine.

10.2. Binary representation, linear separability, and simple decomposition

The Boolean representation of a local logic function of 9 variables can be given in terms of the 9 Boolean input variables $F(u_1, u_2,\ldots, u_9)$.

Given this function as a sum of products, we can directly apply a check to determine whether this function is linearly separable or not. If not, we have to decompose it into a sequence of linearly separable templates (see Section 10.3). The simplest method to generate this sequence, though it does not lead generally to the shortest sequence of templates, via the window truth table. In this case, each window represents a minterm (or maxterm) related directly to an uncoupled template with the coding convention introduced in Chapter 5. For example, window #3



means a minterm $u_2\ u_4\ \bar{u}_5\ u_6\ u_8$ (x means DON'T CARE).

This term is implemented by a CNN with x(0) =0 and the template parameters are : a_{00}=1, z=-4 and b_1, b_2….b_9 are directly coded by window #3.

Hence, window #3

$$u_2 \quad u_4 \quad \bar{u}_5 \quad u_6 \quad u_8$$

generates the input values

$$b_1 = 0, b_2 = 1, b_3 = 0, b_4 = 1, b_5 = -1, b_6 = 1, b_7 = 0, b_8 = 1, b_9 = 0$$

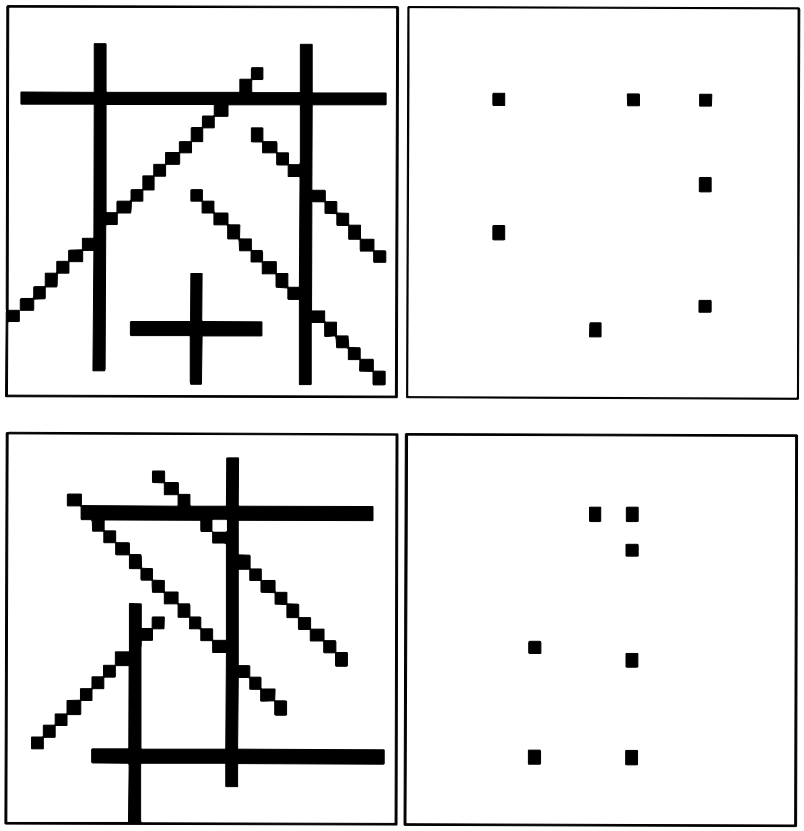
that is, the variables not appearing in the minterm (the DON'T CAREs) , will get a value of 0 at the corresponding places.

Cascading the minterm Boolean functions $F_k(.)$ represented by the appropriate templates by AND-ing the consecutive results, the Boolean function $F(\cdot)$ will be calculated.

Example

Suppose we have a binary image with one-pixel-wide lines. Detect those pixels where the line crossings are of 45 or 90 degrees. Two examples are shown in Figure 2 with few inputs and detected points. Indeed, we started with a blank Window Truth Table (all-white output) and "clicked" those windows black which contain the desired configurations to be detected. These are the following 6 places (simplest cases):

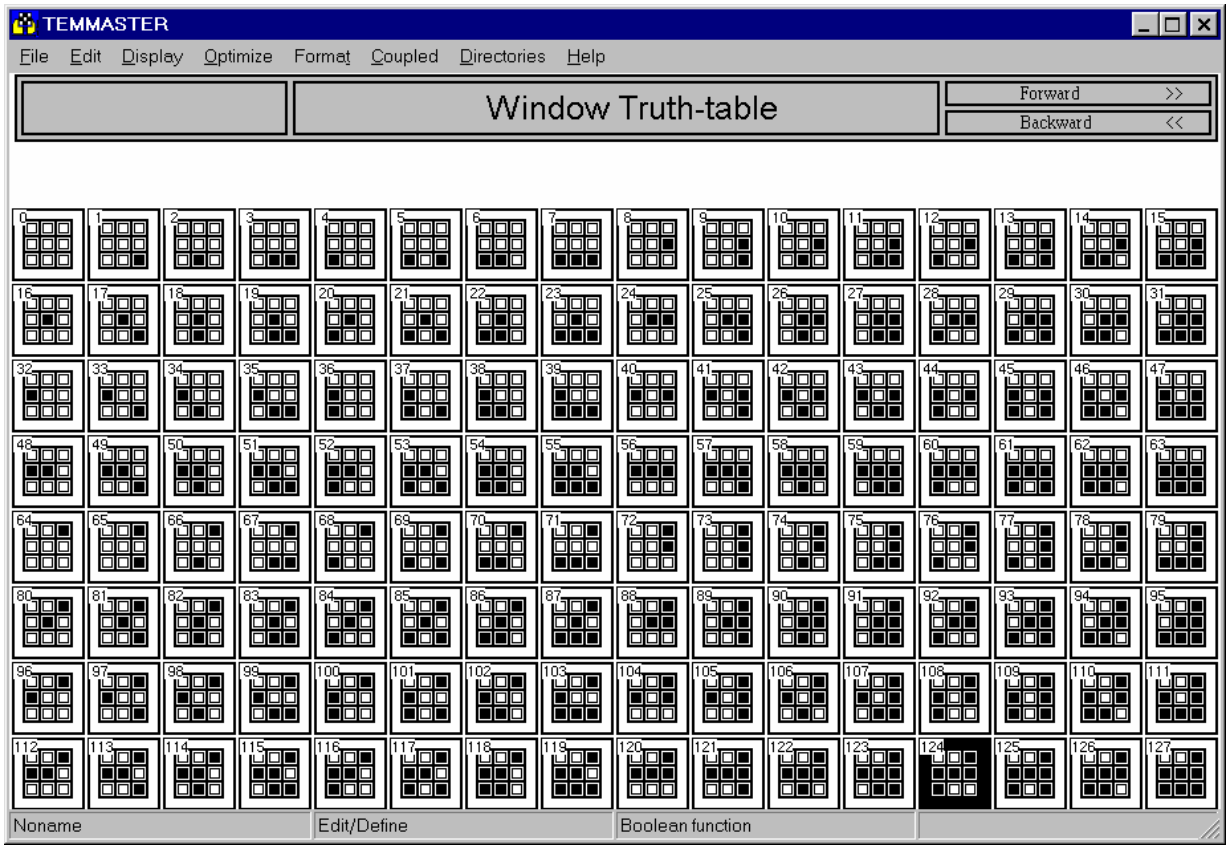
#: 124, 186, 214, 313, 341, 403

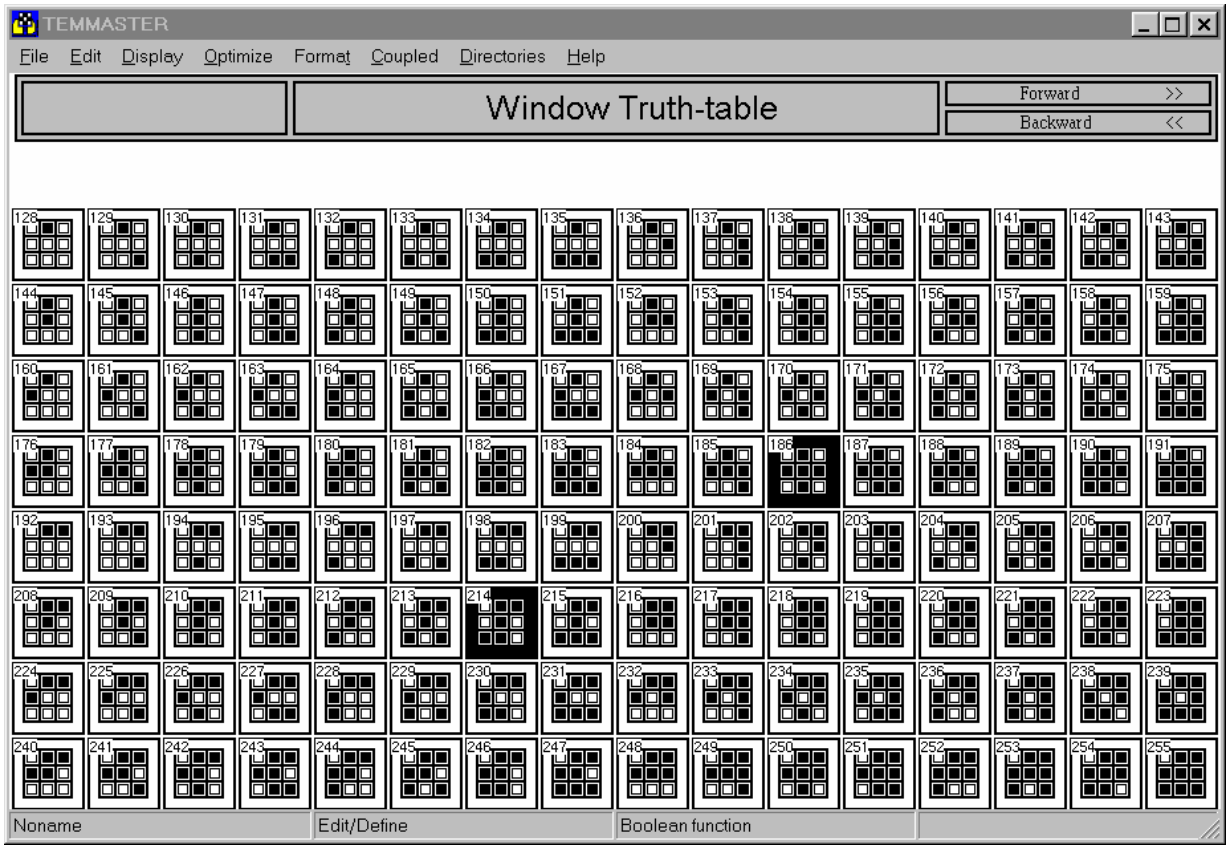


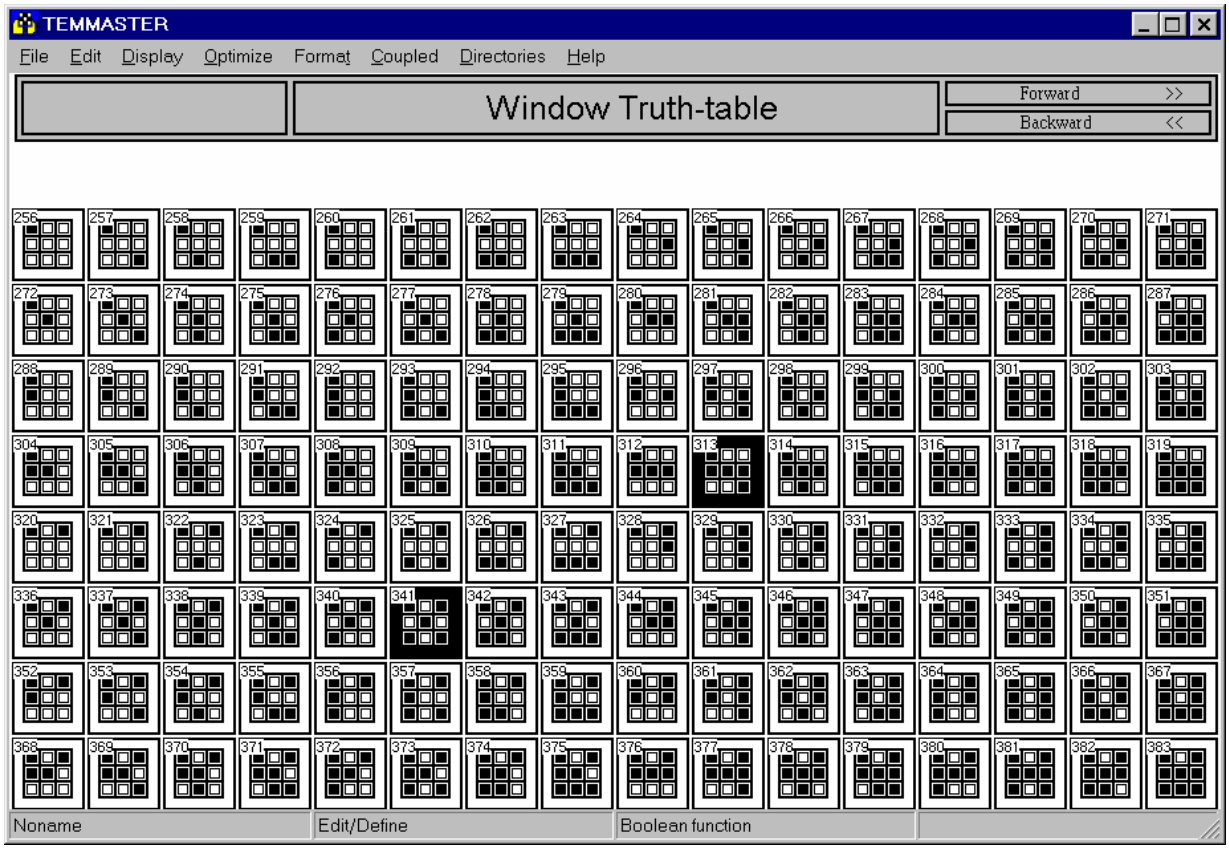
(a) (b)

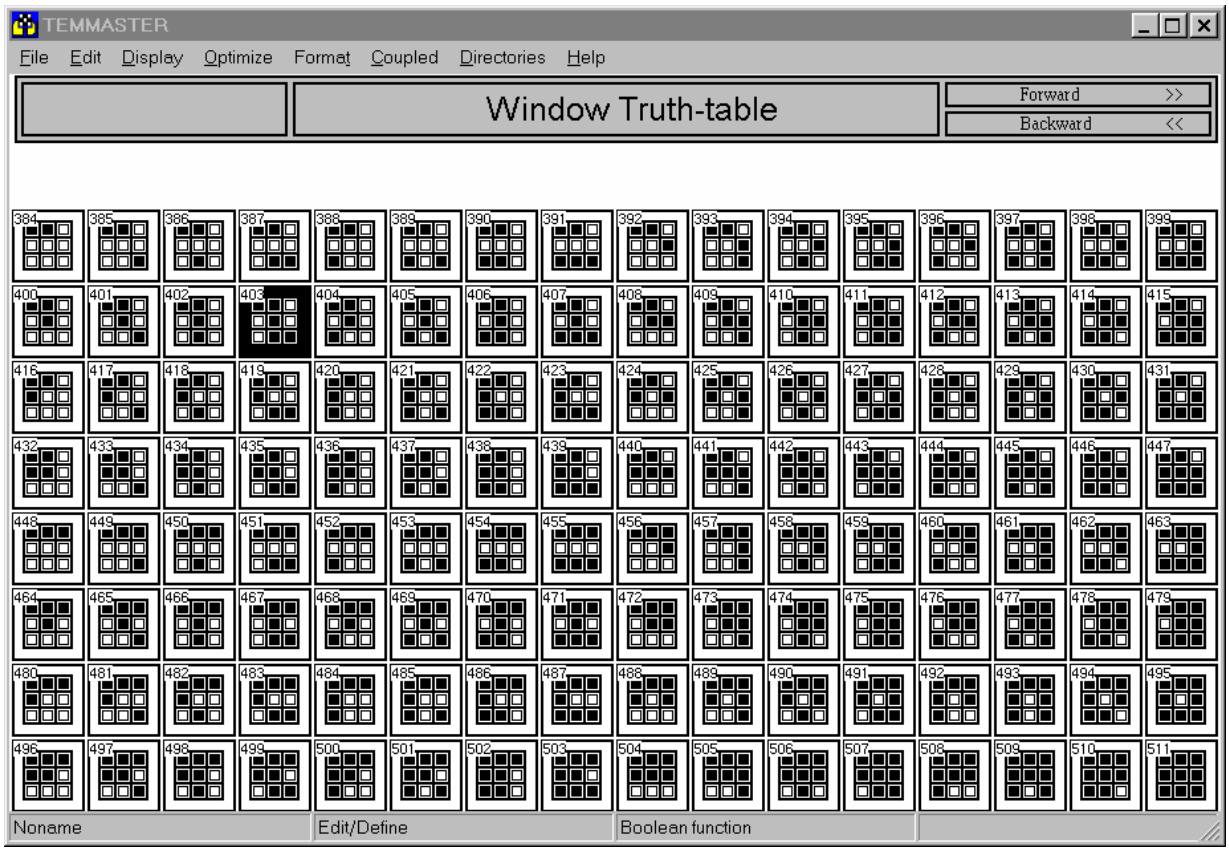
Fig.2. Input images (a) and the corresponding detected crossing places (b)

The selected windows are shown in the next tables


TEMMASTER
File Edit Display Optimize Format Coupled Directories Help
Window Truth-table
Forward >>
Backward <<
Noname
Edit/Define
Boolean function


TEMMASTER
File Edit Display Optimize Format Coupled Directories Help
Window Truth-table
Forward >>
Backward <<
Noname
Edit/Define
Boolean function


TEMMASTER
File Edit Display Optimize Format Coupled Directories Help
Window Truth-table
Forward >>
Backward <<
Noname
Edit/Define
Boolean function


TEMMASTER
File Edit Display Optimize Format Coupled Directories Help
Window Truth-table
Forward >>
Backward <<
Noname
Edit/Define
Boolean function

To each configuration, we code a cloning template. For example, for the last one (#403)

a_{00}=1 B= | 1 | 1 | -1 |
| --- | --- | --- |
| -1 | 1 | -1 |
| -1 | 1 | 1 |

z = -8

AND-ing the 6 templates all the desired crossings will be detected. Two examples are shown in Figure 2.

10.3. Template optimatization

Once we get a template like the one just determined we can optimize it for robustness. Using the method described in Section 6.7, we can optimize a separable binary template to get a separating hyperplane, which is distanced from the two values of output (black and white) equally. The template design and optimatization program TEMPO (Appendix C) contains this function as well.

In the next two cases, TEMPLATE1 and TEMPLATE2, we show the starting values and the optimized values. In the case of TEMPLATE1, which was designed by a cut-and-try method, indeed, it turned out that the robustness of the original template was zero (the hyperplane just hit one output vertex).

TEMPLATE 1: EdgeDetector

Initial template

a_{00}=1 B= | -0.25 | -0.25 | -0.25 |
| --- | --- | --- |
| -0.25 | 2 | -0.25 |
| -0.25 | -0.25 | -0.25 |

z = -1.5

Optimized template

a_{00}=1 B= | -1 | -1 | -1 |
| --- | --- | --- |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

z = -1

TEMPLATE 2: LocalConcavePlaceDetector

Initial template

a_{00}=1 B= | 0 | 0 | 0 |
| --- | --- | --- |
| 1 | 2 | 1 |
| 0.5 | -1 | 0.5 |

z = -5.5

Optimized template

a_{00}=1 B= | 0 | 0 | 0 |
| --- | --- | --- |
| 2 | 2 | 2 |
| 1 | -2 | 1 |

z = -7

This template optimization is perfect if the CNN implementation is ideal. In a real situation with a given VLSI implementation, more complex optimization procedures are to be applied.

10.4. Template decomposition techniques

If the local Boolean function is not linearly separable then we can apply different decomposition techniques. Many of these techniques are based on some assumptions on the template values and the logic functions used for combining the consecutive templates. A method described in Section 7.6 and another "compact" decomposition method[6] are used in the TEMPO program (Appendix C). The determination of the minimal number of templates for any given $F(\cdot)$ is a computationally hard problem. For the example given in Section 10.2, the 6 templates of the minterm decomposition could not be reduced. At the same time, for the game of life problem both methods yielded a decomposition of 2 templates only. The sequences of the 6 templates of our example in Section 10.2 are as follows.

TEMPLATE 1

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} -1.0 & -1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & -1.0 & -1.0 \end{bmatrix} \quad z = -8$$

XOR

TEMPLATE 2

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} -1.0 & 1.0 & -1.0 \\ 1.0 & 1.0 & 1.0 \\ -1.0 & 1.0 & -1.0 \end{bmatrix} \quad z = -8$$

XOR

TEMPLARE 3

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} -1.0 & 1.0 & 1.0 \\ -1.0 & 1.0 & -1.0 \\ 1.0 & 1.0 & -1.0 \end{bmatrix} \quad z = -8$$

XOR

TEMPLATE 4

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} 1.0 & -1.0 & -1.0 \\ 1.0 & 1.0 & 1.0 \\ -1.0 & -1.0 & 1.0 \end{bmatrix} \quad z = -8$$

XOR

[6] L. Nemes, L.O. Chua, and T. Roska, Implementation of Arbitrary Boolean Functions on a CNN Universal Machine, Int. J. CTA, Vol. 26, pp. 593-610, 1998

TEMPLATE 5

A =

0.0	0.0	0.0
0.0	1.0	0.0
0.0	0.0	0.0

B =

1.0	-1.0	1.0
-1.0	1.0	-1.0
1.0	-1.0	1.0

z = -8

XOR

TEMPLATE 6

A =

0.0	0.0	0.0
0.0	1.0	0.0
0.0	0.0	0.0

B =

1.0	1.0	-1.0
-1.0	1.0	-1.0
-1.0	1.0	1.0

z = -8