Training Cellular Automata for Image Processing

Paul L. Rosin

***Abstract*—Experiments were carried out to investigate the possibility of training cellular automata (CA) to perform several image processing tasks. Even if only binary images are considered, the space of all possible rule sets is still very large, and so the training process is the main bottleneck of such an approach. In this paper, the sequential floating forward search method for feature selection was used to select good rule sets for a range of tasks, namely noise filtering (also applied to grayscale images using threshold decomposition), thinning, and convex hulls. Various objective functions for driving the search were considered. Several modifications to the standard CA formulation were made (the B-rule and two-cycle CAs), which were found, in some cases, to improve performance.**

***Index Terms*—Cellular automata, image denoising, image processing, rule selection.**

I. Introduction

CELLULAR automata (CA) consist of a regular grid of cells, each of which can be in only one of a finite number of possible states. The state of a cell is determined by the previous states of a surrounding neighborhood of cells and is updated synchronously in discrete time steps. The identical rule contained in each cell is essentially a finite state machine, usually specified in the form of a rule table with an entry for every possible neighborhood configuration of states

CA are discrete dynamical systems, and they have been found useful for simulating and studying phenomena such as ordering, turbulence, chaos, symmetry-breaking, etc., and have had wide application in modelling systems in areas such as physics, biology, and sociology.

Over the last 50 years, a variety of researchers (including well-known names such as Ulam and von Neumann [1], Holland [2], Wolfram [3], and Conway [4]) have investigated the properties of CA. Particularly, in the 1960s and 1970s, considerable effort was expended in developing special purpose hardware (e.g., CLIP) alongside developing rules for the application of the CAs to image analysis tasks [5]. More recently, there has been a resurgence in interest in the properties of CAs without focusing on massively parallel hardware implementations, i.e., they are simulated on standard serial computers. By the 1990s, CAs could be applied to perform a range of computer vision tasks, such as

- calculating distances to features [6];
- calculating properties of binary regions such as area, perimeter, and convexity [7];
- performing medium level processing such as gap filling and template matching [8];
- performing image enhancement operations such as noise filtering and sharpening [9];
- performing simple object recognition [10].

A related development over the last decade is the introduction of cellular neural networks, an extension of CAs that includes weight matrices. Both continuous time [11] and discrete time [12] versions have been applied to a variety of image processing tasks.

One of the advantages of CAs is that, although each cell generally only contains a few simple rules, the combination of a matrix of cells with their local interaction leads to more sophisticated emergent global behavior. That is, although each cell has an extremely limited view of the system (just its immediate neighbors), localized information is propagated at each time step, enabling more global characteristics of the overall CA system. This can be seen in examples such as Conway's Game of Life as well as Reynolds' [13] Boids simulation of flocking.

A disadvantage with the CA systems described above is that the rules had to be carefully and laboriously generated by hand [14]. Not only is this tedious, but it does not scale well to larger problems. More recently, there has been a start to automating rule generation using evolutionary algorithms. For instance, Sipper [15] shows results of evolving rules to perform thinning, and gap filling in isothetic rectangles. Although the tasks were fairly simple, and the results were only mediocre, his work demonstrates that the approach is feasible (in addition, it should be noted that he used nonuniform CA in which cells can have different rules). Another example is given by Adorni, who generated CAs to perform pattern classification [16].

This paper concentrates on techniques for training CAs to perform several fairly standard image processing tasks to a high level of performance. Once this is achieved, the benefit of the approach is that it should be possible to easily retrain the system to work on other new image processing tasks.

II. Design and Training of the Cellular Automata

In the current experiments, all input images are binary, and cells have two states (i.e., they represent white or black). Each cell's eight-way connected immediate neighbors are considered (i.e., the Moore neighborhood). Fixed-value boundary conditions are applied in which transition rules are only applied to nonboundary cells. The input image is provided as the initial cell values.

A. Rule Set and Its Application

Working with binary images means that all combinations of neighbor values gives 2^8 possible patterns or rules. All the tasks

Manuscript received April 27, 2005; revised October 14, 2005. This work was supported by the IEEE. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Giovanni Ramponi.

The author is with Cardiff University, Cardiff CF24 3XF, U.K. (e-mail: paul.rosin@cs.cf.ac.uk).

Digital Object Identifier 10.1109/TIP.2006.877040

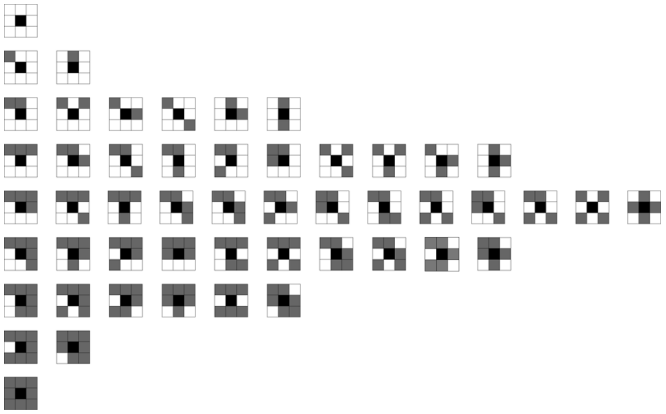


Fig. 1. Complete rule set containing 51 patterns after symmetries and reflections are eliminated. Note that there remains the symmetry of the top half of the set being equivalent to the lower half after reversal of black and white.

covered in this paper should be invariant to certain spatial transforms, and so equivalent rules are combined. Taking into account 45° rotational symmetry and bilateral reflection provides about a five-fold decrease in the number of rules, yielding 51 in total (see Fig. 1[1]). The problem now becomes how to choose a good subset of these rules (which we denote as the *rule set*) to obtain the desired effect.

The 51 neighborhood patterns are defined for a central black pixel, and the same patterns are inverted (i.e., black and white colors are swapped) for the equivalent rule corresponding to a central white pixel. According to the application, there are several possibilities:

- both of these two central black and white rule sets can be learned and applied separately;
- the two rule sets are considered equivalent, and each corresponding rule pair is either retained or rejected for use together, leading to a smaller search space of possible rule sets;
- just one of the black and white rule sets is appropriate, the other is ignored in training and application.

Examples of the latter two approaches will shown in Sections III–V.

Typically, the overall operation of the CA is such that at each pixel the rule set is tested to check if any of its rules match the pixel neighborhood pattern. If so, the central pixel color is inverted, otherwise it remains unaltered. The individual steps of the algorithm are given in Fig. 2. Before processing the image, in Step 1, a flag is set for each of the 51 rules which have been chosen to be in the rule set. In Step 2, the eight pixel values in the 3×3 neighborhood are extracted, and concatenated to form an 8-bit string. If the two rule sets for the central pixel being white and black are considered equivalent (see the paragraph above), then the instances of a central white pixel are inverted (using ones complement) to form the corresponding equivalent pattern for a central black pixel. The

[1]The rules are shown with a black central pixel—which is flipped after the application of the rule. The neighborhood pattern of eight white and/or black (displayed as gray) pixels which must be matched is shown. In the following examples, rule sets are shown (left to right) in the order that the rules were added to the set by the SFFS process.

Procedure *CA*
input: image $\mathcal{I}$
output: image $\mathcal{B}$
begin
  step 1: *initialisation*
  $i \leftarrow 0$
  $\mathcal{B} \leftarrow \mathcal{I}$
  set "active" flags on required rules

  repeat
    $i \leftarrow i + 1$
    $\mathcal{A} \leftarrow \mathcal{B}$

    for each pixel $\mathcal{A}[x][y]$ **begin**

      step 2: *encode neighbourhood pattern*
      make 8 bit string S from values in 3×3 neighbourhood
      if $\mathcal{I}[x][y] = white$ **then**
        $S \leftarrow$ one's complement of S

      step 3: *convert pattern to rule ID*
      $C = \mathrm{LUT}[S]$

      step 4: *conditional application of rule*
      if active$[C] = true$ **then**
        $\mathcal{B}[x][y] \leftarrow invert\,(\mathcal{A}[x][y])$
      else
        $\mathcal{B}[x][y] \leftarrow \mathcal{A}[x][y]$
    endfor

  until $\mathcal{A} = \mathcal{B}$ or $i = M$
end

Fig. 2. Basic algorithm for applying the CA. The process of applying the rules continues until the system has converged or the number of iterations has reached a preset maximum M.

extracted pattern (one of 256 possibilities) is converted into the rule ID (one of 51 possibilities), in which symmetries and reflections have been removed. This can easily and efficiently be performed using a look-up table (LUT)—Step 3. At each iteration, all the image pixels are notionally processed in parallel. However, since we use a sequential operation, the processed pixels are stored in a secondary image $\mathcal{B}$ instead, and then copied back to image $\mathcal{A}$ at the end of each iteration. In Step 4, the pixels are copied to image $\mathcal{B}$ and inverted if the rule corresponding to the 3×3 neighborhood pattern is in the rule set. This cycle is repeated until convergence or the maximum desired number of iterations is reached (fixed at 100 in all the examples shown in this paper).

B. Computational Complexity

Even without the specialized hardware implementations that are available [5], [17], [18], the running time of the CA is moderate. At each iteration, if there are P pixels, and a neighborhood size of N (where $N = 8$ in this paper), the computational complexity is $O(PN)$. Note that the complexity is independent of the number of rules available or active.

To give an indication of the running times, to denoise the binary 1536×1024 images in Section III-A on a 2.0-GHz Pentium 4 with programs coded in C took between 1–55 s using a 3×3 median filter, and between 5–15 s using the CA, depending on how many iterations were required. Of course, training the CA takes considerably longer, but this is not a problem since it can be carried out offline. Again, depending on how many iterations of the rules, and how many rule combinations were considered, this took between 30–60 min.

Convergence of the CA is not guaranteed. As a simple counter example, consider the rule

applied to the (small) image configuration

(where shaded squares indicate black pixels). Repeated application of this rule (in which the rule is applied to both black and white pixels as in the noise filtering application in Section III-A) results in the output alternating between

and

In other situations, convergence does occur, but is slow. For instance, the two rules

and

will erode the end of a single pixel width rectilinear spiral, which could be half the number of image pixels, and so the number of iterations equals the length of the spiral. Nevertheless, in the examples presented in this paper, convergence was achieved in all but one instance (of the B-rule CA) and required, at most, a few tens of iterations.

By more careful coding, several speedups could be achieved. For instance, the assignment $\mathcal{A} \leftarrow \mathcal{B}$ at the beginning of the repeat loop in Fig. 2 would be more efficiently implemented by swapping buffer references rather than copying all the pixel values as is currently implemented. Other, more sophisticated techniques involve avoiding processing some pixels by learning (or having prespecified) certain patterns (e.g., blocks of empty cells), simultaneously processing multiple neighborhoods, etc.

C. Training Strategy

Most of the literature on CA studies the effect of applying manually specified transition rules. The inverse problem of determining appropriate rules to produce a desired effect is hard [19]. In our case, if separate black and white rule sets are used, there are $2^{102} \approx 5 \times 10^{30}$ combinations of rules (possible rule sets) to be considered! Under certain conditions (when adding a feature to a subset does not decrease the criterion function), optimal feature selection is tractable and can be performed using branch and bound algorithms, for example [20]. However, in general, an optimal selection of rules cannot be guaranteed without an exhaustive enumeration of all combinations [21], and this is clearly generally impractical. A simple approach would be to rate the effectiveness of each rule when applied independently, and then use this criterion to direct construction of rule combinations. However, in practice, this does not work well, and methods are required that reveal at least some of the interrule relations.

In the literature on automatically learning rules for CAs, most of the papers focus on a single, somewhat artificial, example, which is a version of the density classification problem on a one-dimensional (1-D) grid. Given a binary input pattern, the task is to decide if there are a majority of 1s or not, i.e, a single binary outcome. For CAs with rules restricted to small neighborhoods, this is a nontrivial task, since the 1s can be distributed through the grid, and so it requires global coordination of distant cells that cannot communicate directly.

Evolutionary solutions appear to be preferred. Mitchell *et al.* [22] used a standard genetic algorithm (GA) to solve the density classification task. Some of the difficulties they encountered with the GA learning were 1) breaking of symmetries in early generations for short-term gains, and 2) the training data became too easy for the CAs in later generations of the GA. Julle and Pollack [23] tackled the latter problem using GAs with co-evolution. To encourage better learning, the training set was not fixed during evolution, but gradually increased in difficulty. Thus, once initial solutions for simple versions of the problem were learned, they would be extended and improved by evolving the data to become more challenging. Instead of GAs, Andre *et al.*[24] used a standard genetic programming framework. Since this was computationally expensive, it was run in parallel on 64 PCs. Extending the density classification task to two-dimensional grids, Jiménez Morales *et al.* [25] again applied standard GA to learn rules.

In comparison to such evolutionary methods, a deterministic feature selection method called the sequential floating forward search (SFFS) [26] is very widely used for building classifier systems. Several studies have compared the effectiveness of SFFS against alternative strategies for feature selection. For instance, Jain and Zongker [27] evaluated fifteen feature selection algorithms (including a genetic algorithm) and found that overall SFFS performed best. Other experiments on different data sets found that there was little difference in effectiveness between GAs and SFFS [28], [29]. Therefore, we have used SFFS rather than evolutionary methods since it has several

Procedure *SFFS*
begin
  step 1: *initialisation*
  $i \leftarrow 0$
  $\mathcal{R}_i \leftarrow \{\}$
  do
    step 2: *select and add the best rule*
    $r^+ \leftarrow \arg\min_{r \notin \mathcal{R}_i} J(\mathcal{R}_i + r)$
    $\mathcal{R}_{i+1} \leftarrow \mathcal{R}_i + r^+$
    $i \leftarrow i + 1$
    step 3: *select the worst rule*
    $r^- \leftarrow \arg\min_{r \notin \mathcal{R}_i} J(\mathcal{R}_i - r)$
    step 4: *conditional removal of worst rule*
    if $J\left(\mathcal{R}_i - r^-\right) < J(\mathcal{R}_i)$
    then
      $\mathcal{R}_{i+1} \leftarrow \mathcal{R}_i - r^-$
    else
      $\mathcal{R}_{i+1} \leftarrow \mathcal{R}_i$
    $i \leftarrow i + 1$
  while $J(\mathcal{R}_i) \leq J(\mathcal{R}_{i-1})$ and unassigned rules remain
end

Fig. 3. Sightly modified version of the sequential floating forward search algorithm used; individual rules are denoted by r, and the score is computed by applying the objective function J (which is to be minimized) to the subset of rules $\mathcal{R}_i$.

advantages: 1) it is extremely simple to implement, and 2) it is relatively fast, providing a good compromise between speed and effectiveness.

The SFFS algorithm can be described as follows. Let $\mathcal{R}_i$ denote the rule set at iteration i and its score be $J(\mathcal{R}_i)$. In our case, $J(\mathcal{R}_i)$ is computed by applying the CA with the rule set $\mathcal{R}_i$ to the input image as specified in Fig. 2, and returning the error computed by one of the objective functions described in Section II-D. The initial rule set $\mathcal{R}_0$ is empty. At each iteration i, all rules are considered for addition to the rule set $\mathcal{R}_{i-1}$. Only the rule giving the best score is retained, to make $\mathcal{R}_i$. This process is repeated until no improvements in score are gained by adding rules (an alternative termination rule is when a known desired number of rules has been found). This describes the sequential forward search, which is extended to the sequential floating forward search by interleaving between each iteration the following test. One at a time, each rule in $\mathcal{R}_i$ is removed to find the rule whose removal provides the candidate rule set $\mathcal{R}'_{i-1}$ with the best score. If this score is better than $J(\mathcal{R}_{i-1})$, then $\mathcal{R}_i$ is discarded, $\mathcal{R}_{i-1}$ is replaced by $\mathcal{R}'_{i-1}$, and the process continues with the addition of the ith rule. Otherwise, $\mathcal{R}'_{i-1}$ is discarded, and the process continues with the addition of the $i+1$th rule to $\mathcal{R}_i$. Whereas the standard SFFS algorithm continues to remove rules one after another while this improves the score, the version used here only removes a single rule between adding two rules and tends to speed up the training process. A procedural description of the SFFS algorithm is given in Fig. 3.

As an alternative to SFFS, Taguchi's orthogonal array method for factorial design [30] was also considered, but it consistently gave worse results than SFFS, and will not be described any further.

The power of training algorithms such as those described in this section is that all that is required is 1) a set of training images, 2) a set of corresponding target (i.e., ideal) output images, and 3) an objective function for evaluating the quality of the actual images produced by the CA, i.e., the error between the target output and the CA output. If this is available, then the training process should be able to select a good (but typically not optimal) set of rules to produce the functionality implicitly specified by the training input and target images, with the following caveats: 1) the image processing function needs to be computable using the available range of rules and 2) the objective function is appropriate for the problem, since the optimization process depends crucially on it.

D. Objective Functions

An objective function is required to direct the SFFS, which is essentially a hill climbing algorithm, and various error measures have been considered in this paper. The first is root mean-square (RMS) error between the input and target image.

In some applications, there will be many more black pixels than white (or vice versa) and it may be preferable to quantify the errors of the black pixels separately from the white. This is done by computing the proportion B of black target pixels incorrectly colored in the output image, and, likewise, W is computed for white target pixels. The combined error is taken as $B + W$.

The above measures do not consider the positions of pixels. In an attempt to incorporate spatial information, the distance at each incorrectly colored pixel in the output image to the closest correctly colored pixel in the target image is calculated. The final error is the summed distances. The distances can be determined efficiently using the distance transform of the target image.

A modification of the above is the Hausdorff distance. Rather than summing the distances, only the maximum distance (error) is returned.

E. Extensions

There are many possible extensions to the basic CA mechanism described above. In this paper, two modifications were implemented and tested. The first is based on Yu *et al.*'s [31] B-rule class of 1-D CA. Each rule tests the value of the central pixel of the *previous* iteration in addition to the usual pixel and its neighbor's values at the *current* iteration. The second variation is to split up the application of the rules into two interleaved cycles (denoted the two-cycle approach). In the even-numbered iterations one rule set is applied, and in the odd-numbered iterations, the other rule set is applied. The two rule sets are learned using SFFS as before, and are not restricted to be disjoint.

III. Noise Filtering

A. Binary Image Processing

The first experiment is on filtering to overcome salt and pepper noise. Two large binary images (1536×1024 pixels) were constructed, one each for training and testing, and consisted of a composite of several 256×256 subimages obtained by thresholding standard images. In the following figures demonstrating the results of processing, only small subparts of the test image are shown so that the fine detail is clearly visible. Varying amounts of noise were added, and for each level the CA rules were learned using the various strategies and evaluation criteria described above. In all instances, the rules were run for

TABLE I
RMS ERRORS OF FILTERED VERSIONS OF THE TEST IMAGE CORRUPTED BY SINGLE PIXEL SALT AND PEPPER NOISE. THE NUMBERS IN BRACKETS INDICATE THE NUMBER OF ITERATIONS OF THE MEDIAN FILTER OR THE STRUCTURING ELEMENT SIZE THAT GAVE THE BEST RESULTS ON THE *TEST* IMAGE

S & P prob.	orig.	median			MM	CA	CA B-rule	CA 2 cycle
		1 iteration	100 iterations	optimal iterations				
0.01	17.9	42.5	54.1	42.5 (1)	17.9 (1)	14.1	12.1	14.1
0.1	57.0	45.0	55.4	45.0 (1)	40.6 (2)	32.4	31.8	32.3
0.3	99.0	55.8	59.3	53.3 (2)	69.0 (2)	47.6	47.7	47.6

100 iterations. It was found that using the SFFS method with the RMS error criterion provided the best results, and, unless otherwise stated, all the results shown used this setup.

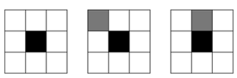
For comparison, results of filtering are providing using 1) a 3×3 median filter and 2) the mathematical morphology (MM) operation of an opening followed by closing using a square structuring element. While there are more sophisticated filters in the literature [32], these still provides a useful benchmark. Moreover, the optimal parameters (number of iterations of the median and width of the structuring element) were determined for the *test* image, giving them favorable bias.

At low noise levels $(p = 0.01)$, the CA learns to use a single rule to remove isolated pixels

As the RMS values show (Table I), this is considerably better than median filtering which in these conditions has its noise reduction overshadowed by the loss of detail. The B-rule CA produces even better results than the basic CA. Fifty rules were learned, although this is probably far from a minimal set since most of them have little effect on the evaluation function during training. As before, the first rule is

applied when the central pixel is a different color in the previous iteration. In contrast, most of the remaining rules are applied when the central pixel is the same color in the previous iteration. The difference in the outputs of the basic and B-rule CAs is most apparent on the portion of the test image containing the finely patterned background to Lincoln (Fig. 4), which has been preserved while the noise on the face has still been removed. The two-cycle CA produces identical results to the basic CA.

At greater noise levels, the CA continues to perform consistently better than the median and morphological filters (see Fig. 5 and Table I). At $p = 0.1$, the learned CA rule set is



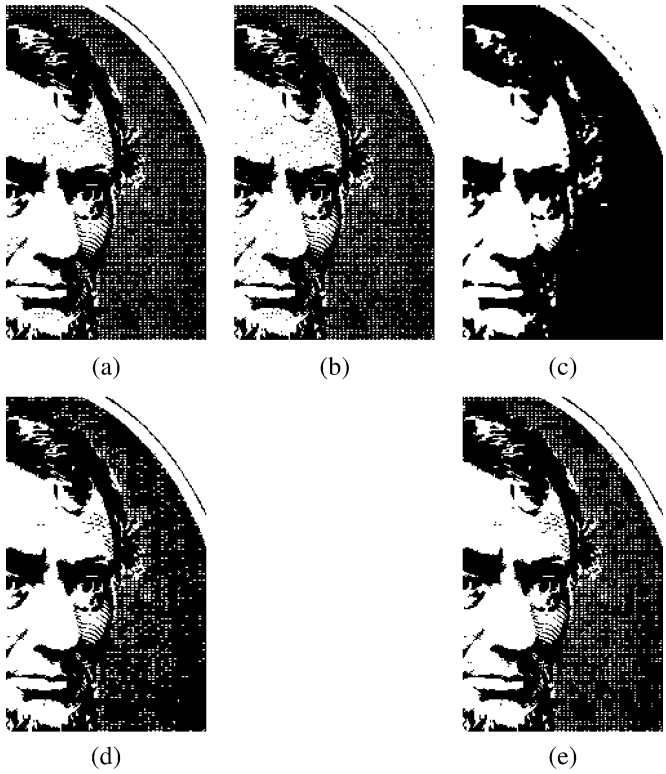


Fig. 4. Salt and pepper noise affecting single pixels occurring with a probability of 0.01; (a) original, (b) original with added noise, (c) 1 iteration of median, (d) filtered with CA, and (e) filtered with B-rule CA.

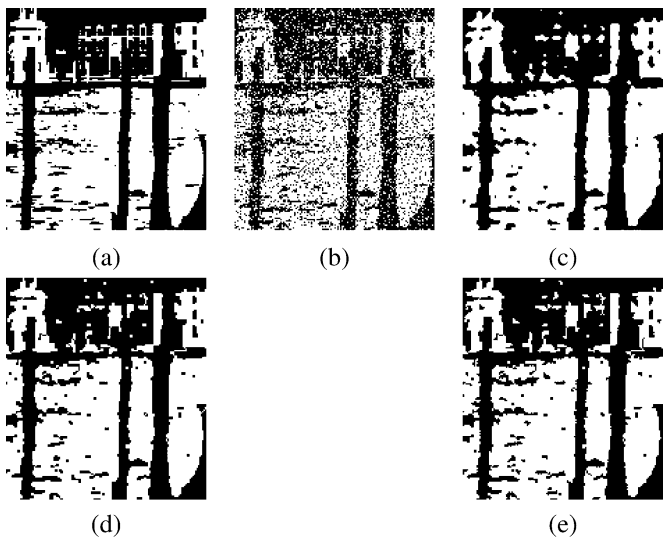
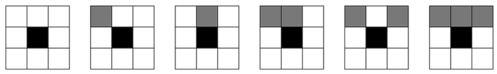


Fig. 5. Salt and pepper noise affecting single pixels occurring with a probability of 0.3; (a) original, (b) original with added noise, (c) two iterations of median, (d) filtered with CA, and (e) filtered with B-rule CA.

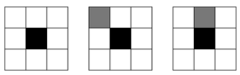
TABLE II
RMS ERRORS OF FILTERED VERSIONS OF 3×3 PIXEL SALT AND PEPPER NOISE

S & P prob.	orig.	3×3 median			5×5 median			MM	CA	CA B-rule	CA 2 cycle
		1 iter.	100 iter.	opt. iter.	1 iter.	100 iter.	opt. iter.				
0.01	53.1	60.9	58.9	56.1 (3)	62.6	79.3	62.6 (1)	53.1 (1)	44.8	37.7	44.6
0.1	141.0	135.0	116.5	116.5 (39)	125.0	95.1	94.5 (25)	131.4 (4)	92.4	89.9	92.9

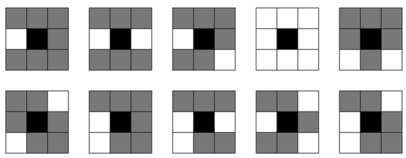
and required 31 iterations for convergence. At $p = 0.3$, the learned CA rule set is



and required 21 iterations for convergence. Again the two-cycle CA produced little improvement over the basic CA, while the B-rule CA does at $p = 0.1$, but not $p = 0.3$. The B-rule rule sets are reasonably compact, and the one for $p = 0.1$ is shown: The rule set applied when the central pixel is a different color in the previous iteration is



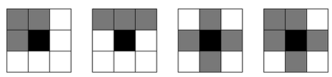
while, for the same colored central pixel at the previous iteration, the rule set is



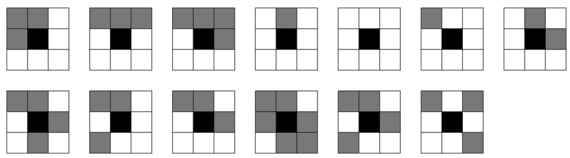
Increasing levels of noise obviously requires more filtering to restore the image. It is interesting to note that not only have more rules been selected as the noise level increases, but also that, for the basic CA, they are strictly supersets of each other.

To test that the training data was sufficiently representative to enable a good rule set to be learned, cross validation was performed. The training and test images were swapped, so that a new rule set was learned from the original test data, and then the CA was applied with these rules to the original training image. The RMS errors obtained were very similar to the values in Table I. Over the three versions of the CA and the three noise levels, the maximum difference in corresponding RMS values was 1.7, and the second largest was only 0.5.

The second experiment makes the noise filtering more challenging by setting 3×3 blocks, rather than individual pixels, to black or white. However, the CA still operates on a 3×3 neighborhood. Given the larger structure of the noise larger (5×5) median filters are used for comparison. However, at low noise levels, ($p = 0.01$) the 3×3 median gave a lower RMS error than the 5×5 although the later was better at high noise levels ($p = 0.1$). Nevertheless, the basic CA outperformed both medians and the morphological filter (Table II). At $p = 0.01$ the learned rule set was



and required 42 iterations for convergence. The B-rule CA further improved the result, and this can most clearly be seen in the fragment of text shown in Fig. 6. At $p = 0.1$ (Fig. 7) the learned rule set was



and even after 100 iterations, the CA had not converged. The two-cycle CA showed only occasional, marginal improvement over the basic CA.

As it was found that the CA was particularly effective for the portions of text in the noisy images, further tests were per-

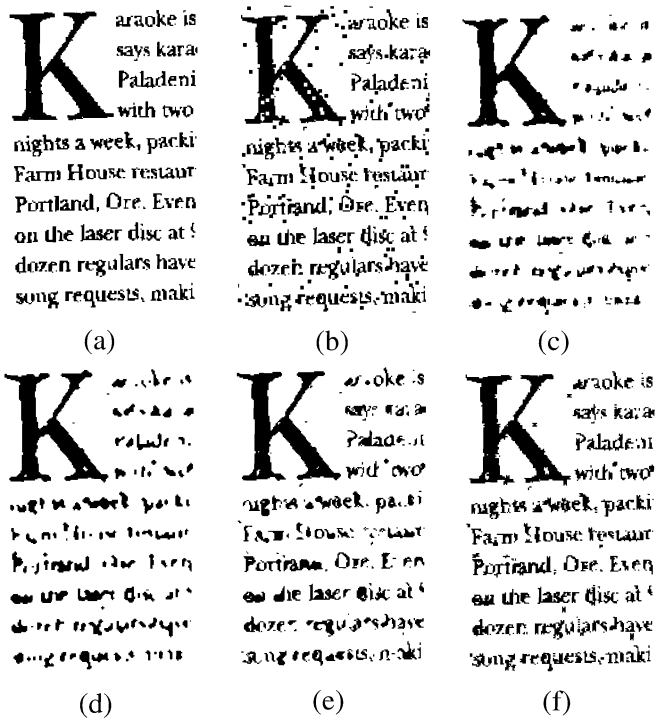


Fig. 6. Salt and pepper noise affecting 3×3 blocks occurring with a probability of 0.01; (a) original, (b) original with added noise, (c) three iterations of median filter, (d) one iteration of 5×5 median filter, (e) filtered with CA, and (f) filtered with B-rule CA.

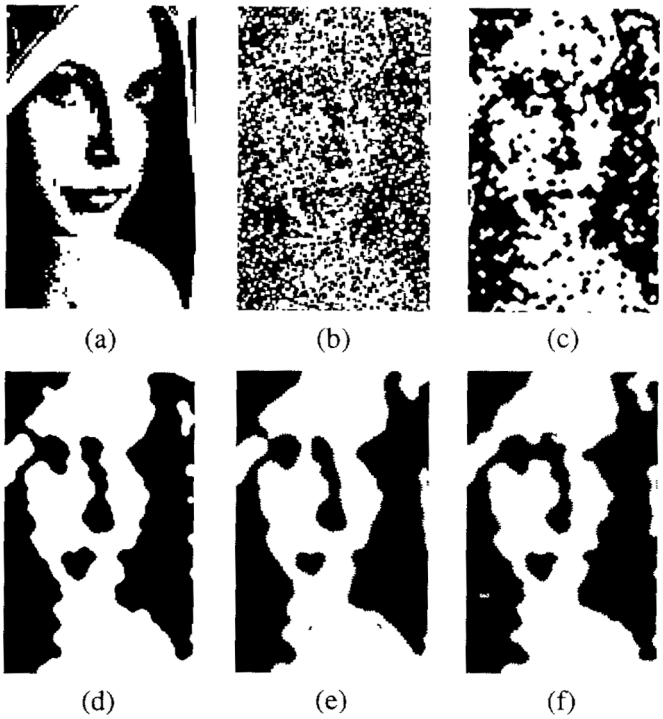


Fig. 7. Salt and pepper noise affecting 3×3 blocks occurring with a probability of 0.1; (a) original, (b) original with added noise, (c) 39 iterations of median, (d) 25 iterations of 5×5 median, (e) filtered with CA, and (f) filtered B-rule CA.

TABLE III
RMS ERRORS OF FILTERED VERSIONS OF THE SCANNED IMAGES OF TEXT CORRUPTED BY SALT AND PEPPER NOISE

noise		orig.	median	MM	CA	B-rule	2 cycle
single pixel	$p = 0.01$	18.16	69.22 (1)	18.16 (1)	**11.02**	11.56	**11.02**
		18.35	96.21 (2)	18.35 (1)	**13.23**	14.89	**13.23**
		18.04	88.18 (2)	18.04 (1)	**16.12**	16.08	**16.12**
		18.07	68.41 (1)	18.07 (2)	**12.22**	12.30	**12.22**
	$p = 0.1$	57.00	69.93 (1)	57.00 (1)	45.73	**41.51**	44.27
		57.21	96.40 (2)	57.21 (1)	52.02	**47.97**	51.65
		56.78	88.08 (2)	56.78 (1)	57.79	**54.07**	57.81
		56.89	69.65 (1)	56.89 (2)	43.28	**41.44**	42.75
	$p = 0.3$	99.00	71.66 (2)	94.51 (2)	68.96	68.98	**68.83**
		98.98	97.81 (2)	98.98 (1)	87.40	87.38	**87.03**
		99.05	89.40 (2)	99.05 (1)	83.40	83.38	**83.08**
		98.88	74.87 (2)	93.55 (2)	69.30	69.34	**69.17**
3×3 block	$p = 0.01$	53.52	75.68 (25)	53.52 (1)	42.40	**40.71**	42.42
		53.36	100.35 (2)	53.36 (1)	58.32	**50.50**	58.46
		53.60	93.06 (2)	53.60 (1)	58.02	**52.49**	57.78
		53.53	79.22 (2)	53.53 (1)	50.55	**44.07**	50.96
	$p = 0.1$	140.52	119.70 (35)	140.52 (1)	75.47	77.38	**75.37**
		140.46	133.98 (42)	140.46 (1)	104.59	106.52	**104.28**
		140.19	129.47 (46)	140.19 (1)	94.15	95.76	**93.97**
		140.91	121.30 (58)	140.91 (1)	80.48	82.74	**80.22**

formed on four images each made up from a scanned portion of text (each image containing a different font style/size). The results of applying the same rules learned during the previous experiments are shown in Table III, where it can be seen that the CA again outperformed the median and the mathematical morphology opening followed by closing. Since the images contain very fine detail (the height of an upper case character lies between eight and 13 pixels), median filtering was too severe (degrading rather than improving the image) for moderate amounts of noise. Likewise, for moderate amounts of noise, the opening/closing tended to degrade the image for structuring elements larger than 1×1. For the more severe 3×3 block noise, multiple iterations of the median could be effective, but were still outperformed by the CA. The opening/closing did not work at all well, however, as structuring elements large enough to eliminate the noise also removed all the finer detail of the text.

B. Grayscale Image Processing

While the previous discussion and results were restricted to binary images, it would obviously be advantageous to work with gray-level images, too. The natural difficulty is that increasing the range of intensities will also vastly increase the number and/or complexity of the rules. However, one way to avoid this consequence is to use threshold decomposition, in which the gray-level image is decomposed into the set of binary images obtained by thresholding at all possible gray levels. Binary filtering is applied to each binary image, and the results combined—in our case, we simply add the set of filtered binary images. Thus, there are two advantages of this approach: The same setup as binary image processing can be reused, which means that, given the smaller search space compared to that for gray-level processing, faster training of the CA is achieved. The downside is that, after training, running the CA is less efficient given the overhead of performing threshold decomposition. For L intensity levels the computational complexity becomes $O(LPN)$ per iteration. In addition, there is no equivalence between training and applying the CA on gray-level imagery as opposed to the set of thresholded, binary images. This means that the training is not necessary optimal for gray-level processing.

The results of filtering different types and magnitudes of noise on three images (Barbara, couple, and venice) are listed in Table IV. The same filters are used as in the previous section as well as two of the many modifications of the median in the literature: the relaxed median filter [33] and the median-rational hybrid filter (MRHF) [34]. These are additionally compared against two techniques based on hidden Markov trees (HMTs) applied to wavelet coefficients. The first constructs a "universal" model [35] of the image, while the second estimates the model parameters and applies a Wiener filter with those parameters [36]. It can be seen that the HMT performs the best for Gaussian distributed noise, but that the CA performs best on the salt and pepper noise. For the case of noise probability 0.1, it is noteworthy that the CA trained on $p = 0.3$ is actually consistently more effective than that trained on $p = 0.1$. This could either be an effect that arises because the CA was trained on binary images rather than grayscale, or else indicate poor generalization from the training data. While it is advantageous to train with data that matches the expected test set as closely as possible, it can be seen that for each of the Gaussian noise and single pixel salt and pepper noise filtering tasks the CA outperformed the median filter for a range of training data set parameter values.

TABLE IV
RMS ERRORS OF FILTERED VERSIONS OF GRAY-LEVEL IMAGES CORRUPTED BY VARIOUS TYPES AND LEVELS OF NOISE. EACH SET OF THREE ROWS LISTS RESULTS FOR THREE NOISY IMAGES: BARBARA, COUPLE, AND VENICE. THE MEDIAN FILTER WAS RUN FOR THE OPTIMAL NUMBER OF ITERATIONS DETERMINED FOR *EACH TEST* IMAGE. LIKEWISE, THE TWO PARAMETERS (h AND k) FOR THE MRHF FILTER, AND THE WIDTH OF THE MM STRUCTURING ELEMENT WERE OPTIMIZED FOR EACH TEST IMAGE. FOR THE TEST DATA WITH 3×3 BLOCK SALT AND PEPPER NOISE RESULTS ARE SHOWN FOR BOTH THE CA TRAINED ON 3×3 BLOCK NOISE ($p = \{0.01, 0.1\}$), AS WELL AS THE CA TRAINED ON SINGLE POINT NOISE ($p = 0.6$). FOR EACH ROW (I.E., EACH NOISY IMAGE THAT WAS FILTERED) THE LOWEST RMS VALUE IS HIGHLIGHTED

noise model	unfiltered	median	relaxed median	MRHF	MM	CA			HMT	
						0.1	0.3	0.6	universal	Wiener
Gaussian	10.00	14.48	6.94	8.46	10.00	9.79	11.91	16.19	10.60	**6.90**
$\sigma = 10$	9.97	10.27	8.90	6.97	9.40	7.86	7.78	11.67	7.22	**6.38**
	9.73	6.94	14.48	6.02	7.75	7.04	6.19	9.19	5.93	**5.59**
Gaussian	24.63	16.74	**10.62**	15.10	20.39	17.12	15.44	17.11	15.42	11.64
$\sigma = 25$	24.71	12.31	12.31	12.76	16.78	15.92	12.41	13.23	11.95	**10.88**
	23.26	10.62	16.74	11.48	15.19	15.10	11.10	11.13	10.79	**10.06**
salt and pepper	43.94	14.66	**6.41**	12.82	15.90	11.97	11.73	16.15	19.26	24.02
$p = 0.1$	42.72	10.29	8.40	8.62	9.50	9.56	**6.94**	11.54	16.21	19.87
	47.59	6.40	14.66	7.71	8.28	9.26	**4.87**	9.00	17.56	23.41
salt and pepper	107.65	23.72	**19.83**	62.42	84.75	86.26	54.46	23.36	39.14	40.56
$p = 0.6$	104.60	19.75	19.75	59.38	83.63	83.41	51.77	**19.64**	32.72	34.33
	116.37	**19.82**	23.72	66.79	88.56	92.37	57.13	19.96	52.05	52.95
	unfiltered	**median**	**r. median**	**MRHF**	**MM**	**(0.01) 3×3**	**(0.1) 3×3**	**0.6**	**universal**	**Wiener**
salt and pepper	41.08	17.85	**12.74**	36.30	24.96	13.13	20.25	17.79	33.22	39.67
3×3 blocks	40.03	13.55	13.56	33.24	15.89	**12.11**	19.15	13.60	33.17	39.00
$p = 0.01$	44.45	12.73	17.85	35.58	13.42	13.29	19.21	**12.32**	35.64	43.45
salt and pepper	108.23	76.64	82.15	97.85	82.05	81.91	**25.01**	72.48	60.23	97.8
3×3 blocks	105.37	73.94	73.94	94.80	81.29	79.35	**22.64**	69.79	56.7	95.07
$p = 0.1$	117.58	100.48	76.64	106.08	85.27	88.93	**26.04**	77.42	70.3	107.46

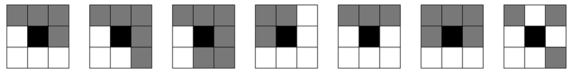
An example of the outputs of the filtering are shown in Fig. 8. The CA has removed most of the 3×3 salt and pepper noise, unlike the HMT method. While the optimal result from the 3×3 median filter has managed to eliminate slightly more noise, it has also blurred out most of the texture on the clothes. This is made clearer in Fig. 9, which shows the difference images between the results and the source image, with all values scaled by factor of three to help visualization. Unlike the median, the CA has managed to retain the majority of the detail.

Running times for applying the CA to the above images (512×512 pixels, 236–256 gray levels) varied depending on the number of iterations. The decomposition of the gray-level image into binary images took 20 s, while reconstruction of the filtered binary images to a gray-level image took 7 s—the difference in times occurs only because each of these tasks was carried out by separate stand-alone programs that handled I/O in different ways. Applying the CA to the set of decomposed binary images took between 50–220 s. Run-time would be improved by performing the threshold decomposition within the CA program, thereby minimizing the large amount of slow I/O currently performed.

IV. THINNING

The second application of CAs we show is thinning of black regions, and so rules were only triggered by black pixels. Training data was generated in two ways. First, some one pixel wide curves were taken as the target output, and were dilated by varying amounts to provide the prethinned input. In addition, some binary images were thinned by the thinning algorithm by Zhang and Suen [37]. Both sets of data were combined to form a composite training input and output image pair [see Fig. 10(a) and (b)]. Contrary to the image processing tasks in the previous sections, the RMS criterion did not produce the best results, and instead the summed proportions of black pixel errors and white pixel errors was used. Surprisingly, the summed distance and Hausdorff distance error measures gave very poor results. It had seemed likely that they would be more appropriate for this task given the sparse nature of skeletons which would lead to high error estimates for even small mislocations if spatial information were not incorporated. However, it was noted that they did not lead the SFFS procedure to a good solution. Both of them produced rule sets with higher errors than the rule set learned using RMS, even according to their own measures.

The test image and target obtained by Zhang and Suen's thinning algorithm are shown in Fig. 10(c) and (d). The basic CA does a reasonable job [Fig. 10(e)], and the rule set is



The last rule has little effect, only changing three pixels in the image. Some differences with respect to Zhang and Suen's

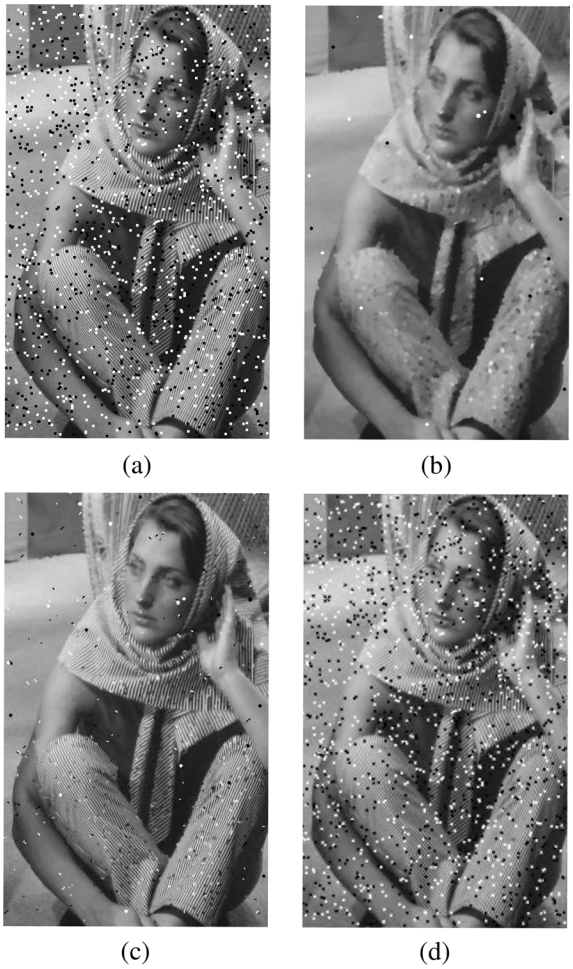


Fig. 8. (a) Portion of the barbara image with 3×3 salt and pepper noise ($p = 0.01$), (b) nine iterations of median, (c) filtered with CA, and (d) universal HMT.

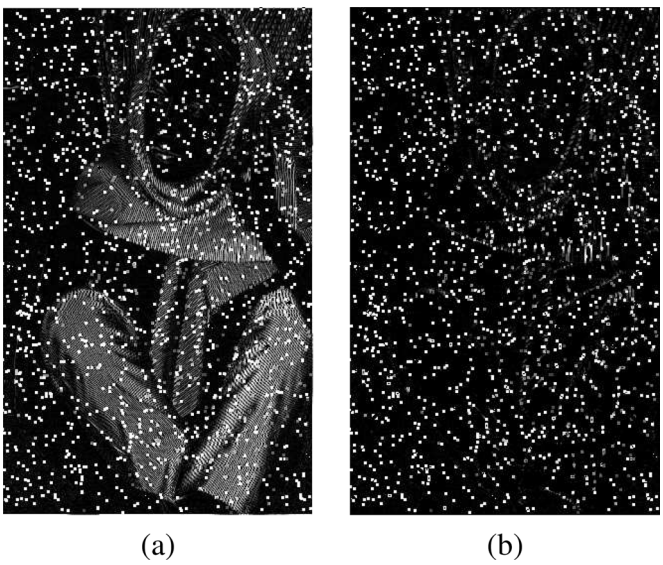


Fig. 9. Difference images (intensities scaled by a factor of three) between the uncorrupted barbara image and (a) nine iterations of the median and (b) the CA. Note the texture removed by the median filter.

output can be seen. In the wide black regions, horizontal rather than diagonal skeletons are extracted, although it is not obvious

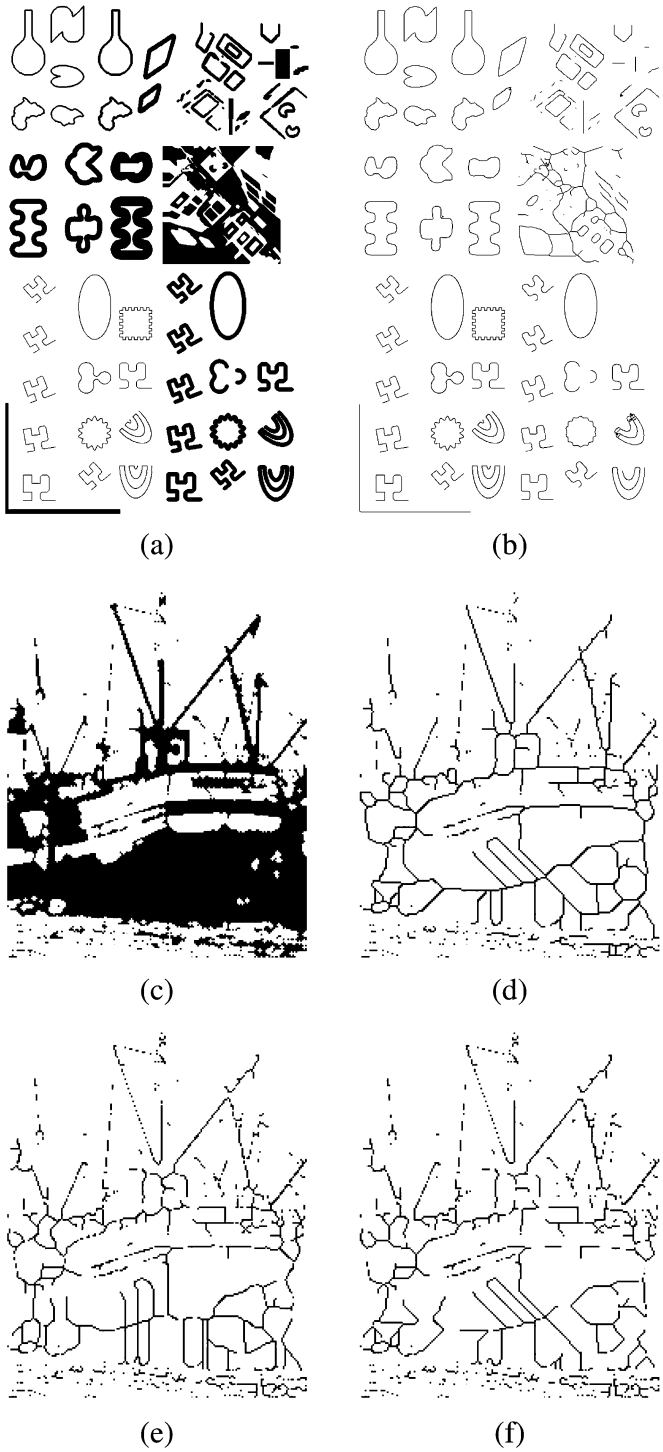
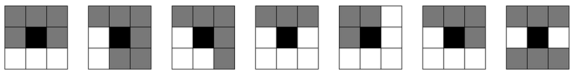


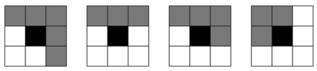
Fig. 10. Image thinning. (a), (b) Training input and target output, (c) test input, (d) test image thinned using Zhang and Suen's algorithm, (e) test image thinned with CA, and (f) test image thinned with two-cycle CA.

which is more correct. Also, a more significant problem is that some lines were fragmented. This is not surprising since there are limitations when using parallel algorithms for thinning, as summarized by Lam *et al.* [38]. They state that to ensure connectedness either the neighborhood needs to be larger than 3×3. Alternatively, 3×3 neighborhoods can be used, but each iteration of application of the rules is divided into a series of subcycles in which different rules are applied.

This suggests that the two cycle CA should perform better. The rule set learned for the first cycle is



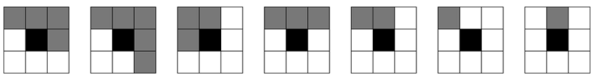
and the second cycle rule set is a subset of the first



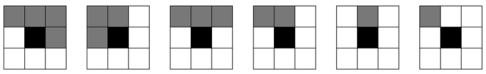
Again, the last and least important rule from the first cycle has little effect (only changing six pixels) and so the basic CA and the first cycle of the B-rule have effectively the same rule set. As Fig. 10(f) shows, the output is a closer match to Zhang and Suen's, as the previously vertical skeleton segments are now diagonal. However, connectivity has not been improved.

V. Convex Hulls

The next experiment tackles finding the convex hulls of all regions in the image. If the regions are white then rules need only to be applied at black pixels since white pixels should not be inverted. Again, like the thinning task, the summed proportions of black pixel errors and white pixel errors was used. After training the learned rule set was applied to a separate test image [Fig. 11(a)]. Starting with a simple approximation as the output target, a four-sided hull, i.e., the axis aligned minimum bounding rectangle (MBR), the CA is able to produce the correct result as shown in Fig. 11(b). The rule set learned is



Setting as target the digitized true convex hull [see Fig. 11(c)], the CA learns to generate an eight-sided approximation to the convex hull [Fig. 11(d)] using the rule set



Interestingly, in comparison to the eight-sided output, the only difference to the rules for the four-sided output is the removal of the single rule

The limitations of the output convex hull are to be expected given the limitations of the current CA. Borgefors and Sanniti di Baja [39] describe parallel algorithms for approximating the convex hull of a pattern. Their 3×3 neighborhood algorithm produces similar results to Fig. 11(d). To produce better results, they had to use larger neighborhoods and more complicated rules.

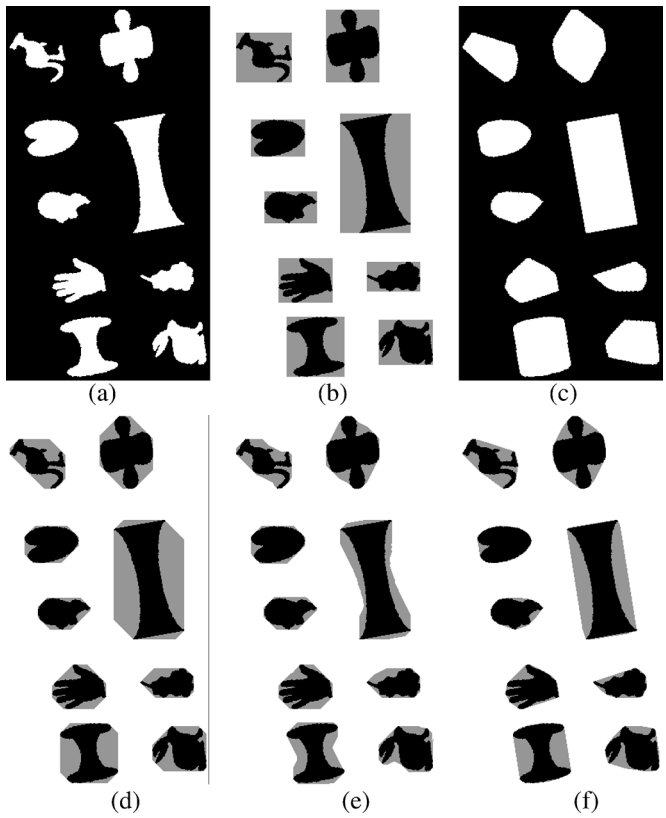


Fig. 11. Results of learning rules for the convex hull. (a) Test input; (b) CA result with MBR as target overlaid on input; (c) target convex hull output; (d) CA result with (c) as target overlaid on input; (e) two-cycle CA result with (c) as target overlaid on input; (f) overlaid CA result generated by combining five orientations.

Therefore, extending the basic CA's capability by applying the two-cycle version should enable the quality of the convex hull to be improved. As Fig. 11(e) shows, the result is no longer convex although is a closer match to the target in terms of its RMS error. This highlights the importance of the evaluation function. In this instance, simply counting pixels is not sufficient, and a penalty function that avoids nonconvex solutions would be preferable, although computationally more demanding.

Another approach to improving results is inspired by the threshold decomposition described in Section III-B. Rather than develop more complicated rules, the simple rules are applied to multiple versions of the data. In this case, the image is rotated by equal increments between 0° and 45°. The basic CA is applied, and the outputs rotated back to 0°. The eight-sided outputs are combined by keeping as convex hull pixels only those that were set as convex hull pixels in *all* the outputs. This process is demonstrated in Fig. 11(f), in which the five orientations 0°, 9°, 18°, 27°, and 36° have been combined to produce a close approximation to the convex hull.

VI. Conclusion and Discussion

The initial experiments with CAs are encouraging. It was shown that it is possible to learn good rule sets to perform common image processing tasks. Moreover, the modifications to the standard CA formulation (the B-rule and two-cycle CAs)

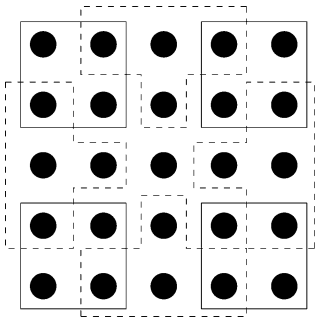


Fig. 12. 5×5 neighborhood, showing subwindows from which the majority pixel color is extracted. Many other arrangements of subwindows are possible.

were found to improve performance in several instances. In particular, for filtering salt and pepper noise, the CA performed better than standard median filtering.

While the examples in this paper demonstrated covered fairly traditional tasks, the important benefit of the trained CA approach is its flexibility. Having shown the capabilities of such a system, the next step will be to apply it to less common image processing tasks, e.g., filtering of more specific and unusual types of noise, and more specialized feature detection.

To further improve performance, there are several areas to investigate. The first is alternative neighborhood definitions (e.g., larger neighborhoods, circular and other shaped neighborhoods, different connectivity), possibly in combination (e.g., not all rules need to have the same neighborhood size or shape). Of course, larger neighborhoods can lead to computational difficulties. For example, ignoring symmetries, a 7×7 window yields $2^{48} \approx 2.8 \times 10^{14}$ rules, and $2^{2^{48}}$ different rule sets to consider. One possibility we explored was to build larger neighborhoods, but aggregate values within subwindows. For instance, Fig. 12 shows a 5×5 neighborhood, and the solid and dashed lines indicate the eight overlapping subwindows. From each subwindow, only the majority pixel color was used, and so the total number of patterns is 2^8 as before. Thus, a larger neighborhood has been achieved without increasing the search space, but at the cost of coarser granularity within the neighborhood. However, experiments on both the noise removal and thinning tasks did not demonstrate any improvements in results over the basic 3×3 neighborhood approach. An alternative is a modification of the two-cycle approach, in which the image is split into subfields (e.g., each field containing alternate pixels) and each subfield processed at separate, interleaved iterations [5].

Larger numbers of rules leads to the second consideration: Can additional constraints be included to prune the search space, improving efficiency and possibly effectiveness? Third, alternative training strategies to SFFS should be considered, such as evolutionary programming.

Fourth, in the current formulation, a cell's state is equivalent to its intensity. If cells were allowed extra states, separate from or in addition to their intensities, the power of the CA system would be substantially increased. A simple example of this is filling holes in a binary image, which would be difficult to perform with the current CA architecture. A simple solution for this task was given by Yang [40], which differed from our approach in two ways: 1) the original source image was available at all iterations (effectively providing an additional state), and 2) the CA was not initialized by the input image to be processed. Instead, the initial state was an all black image, which was subsequently eroded around the holes.

Fifth, most CAs use identical rules for each cell. To enhance the flexibility it may be necessary to extend the approach to nonuniform CA, in which different rules could be applied in different locations, and possibly also at different time steps. For instance, as the state of a cell changes, this could cause the rule set to switch.

Finally, an important topic to develop is the objective function, which is critical to the success of the system. Although several, fairly general, objective functions were evaluated, there may be better ones available—particularly if they are tuned to the specific image processing task. For instance, for thinning, it would be possible to include some factor relating to connectivity so as to penalize fragmentation of the skeleton. A similar approach was taken by Kitchen and Rosenfeld [41] for assessing edge maps, using a combination of good continuation and thinness measures which were calculated within 3×3 windows. Likewise, it was previously noted that for the convex hull task nonconvex solutions should be explicitly penalized.

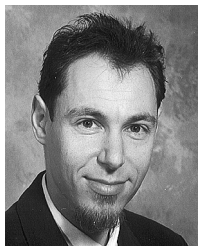
ACKNOWLEDGMENT

The author would like to thank J. Romberg and H. Choi for providing the code for wavelet-domain HMT filtering.

REFERENCES

[1] J. von Neumann, *Theory of Self-Reproducing Automata*. Urbana, IL: Univ. Illinois Press, 1966.
[2] J. Holland and A. Burks, "Logical theory of adaptive systems," in *Essays in Cellular Automata*. Urbana, IL: Univ. Illinois Press, 1970.
[3] S. Wolfram, *Cellular Automata and Complexity Collected Papers*. Reading, MA: Addison-Wesley, 1994.
[4] M. Gardner, "The fantastic combinations of John Conway's new solitaire game life," *Sci. Amer.*, pp. 120–123, 1970.
[5] K. Preston and M. Duff, *Modern Cellular Automata-Theory and Applications*. New York: Plenum, 1984.
[6] A. Rosenfeld and J. Pfaltz, "Digital distance functions on digital pictures," *Pattern Recognit.*, vol. 1, no. 1, pp. 33–61, 1968.
[7] C. Dyer and A. Rosenfeld, "Parallel image processing by memory-augmented cellular automata," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-3, no. 1, pp. 29–41, Jan. 1981.
[8] T. de Saint Pierre and M. Milgram, "New and efficient cellular algorithms for image processing," *CVGIP: Image Understand.*, vol. 55, no. 3, pp. 261–274, 1992.
[9] G. Hernandez and H. Herrmann, "Cellular automata for elementary image enhancement," *Graph. Models Image Process.*, vol. 58, no. 1, pp. 82–89, 1996.
[10] I. Karafyllidis, A. Ioannidis, A. Thanailakis, and P. Tsalides, "Geometrical shape recognition using a cellular automaton architecture and its VLSI implementation," *Real-Time Imag.*, vol. 3, pp. 243–254, 1997.
[11] L. Chua and L. Yang, "Cellular neural networks: Applications," *IEEE Trans. Circuits Syst.*, vol. 35, no. 10, pp. 1273–1290, Oct. 1988.
[12] H. Harrer and J. Nossek, "Discrete-time cellular neural networks," *Int. J. Circ. Th. Appl.*, vol. 20, pp. 453–467, 1992.
[13] C. Reynolds, "Flocks herds and schools a distributed behavioral model," in *Proc. SIGGRAPH*, 1987, pp. 25–34.
[14] B. Viher, A. Dobnikar, and D. Zazula, "Cellular automata and follicle recognition problem and possibilities of using cellular automata for image recognition purposes," *Int. J. Med. Inf.*, vol. 49, no. 2, pp. 231–241, 1998.
[15] M. Sipper, "The evolution of parallel cellular machines toward evolware," *BioSystems*, vol. 42, pp. 29–43, 1997.
[16] G. Adorni, F. Bergenti, and S. Cagnoni, *A Cellular-Programming Approach to Pattern Classification*, 1998, vol. 1391, Lecture Notes in Computer Sciences, pp. 142–150.

[17] T. Toffoli and N. Margolus, *Cellular Automata Machines*. Cambridge, MA: MIT Press, 1987.
[18] M. Halbach and R. Hoffmann, "Implementing cellular automata in fpga logic," in *Proc. 18th Int. Parallel and Distributed Processing Symp.*, 2004, pp. 258a–258a.
[19] N. Ganguly, B. Sikdar, A. Deutsch, G. Canright, and P. Chaudhuri, "A Survey on Cellular Automata," Centre for High Performance Computing, Dresden University of Technology, Tech. Rep. 9, 2003.
[20] P. Somol, P. Pudil, and J. Kittler, "Fast branch and bound algorithms for optimal feature selection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 7, pp. 900–912, Jul. 2004.
[21] T. Cover and J. V. Campenhout, "On the possible orderings in the measurement selection problem," *IEEE Trans. Systems, Man, Cybern.*, vol. 7, no. 9, pp. 657–661, 1977.
[22] M. Mitchell, P. Hraber, and J. Crutchfield, "Evolving cellular automata to perform computation: Mechanisms and impedients," *Phys. D*, vol. 75, pp. 361–391, 1994.
[23] H. Juillé and J. Pollack, "Coevolving the ideal trainer: Application to the discovery of cellular automata rules," in *Proc. 3rd Conf. Genetic Programming*, 1998, pp. 519–527.
[24] D. Andre, F. B. III, and J. Koza, "Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem," in *Proc. 1st Conf. Genetic Programming*, 1996, pp. 3–11.
[25] F. J. Morales, J. Crutchfield, and M. Mitchell, "Evolving two-dimensional cellular automata to perform density classification: A report on work in progress," *Parallel Comput.*, vol. 27, pp. 571–585, 2001.
[26] P. Pudil, J. Novovicova, and J. Kittler, "Floating search methods in feature-selection," *Pattern Recognit. Lett.*, vol. 15, no. 11, pp. 1119–1125, 1994.
[27] A. Jain and D. Zongker, "Feature-selection: Evaluation, application, and small sample performance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 2, pp. 153–158, Feb. 1997.
[28] H. Hao, C. Liu, and H. Sako, "Comparison of genetic algorithm and sequential search methods for classifier subset selection," in *Proc. 7th Int. Conf. Document Analysis and Recognition*, 2003, pp. 765–769.
[29] M. Raymer, W. Punch, E. Goodman, L. Kuhn, and A. Jain, "Dimensionality reduction using genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 4, no. 2, pp. 164–171, Feb. 2000.
[30] R. Roy, *A Primer on the Taguchi Method*. Dearborn, MI: Soc. Manufact. Eng., 1990.
[31] D. Yu, C. Ho, X. Yu, and S. Mori, "On the application of cellular automata to image thinning with cellular neural network," in *Cell. Neural Netw. Appl.*, 1992, pp. 210–215.
[32] T. Chen and H. Wu, "Application of partition-based median type filters for suppressing noise in images," *IEEE Trans. Image Process.*, vol. 10, no. 6, pp. 829–836, Jun. 2001.
[33] A. ben Hamza, P. Luque-Escamilla, J. Martínez-Aroza, and R. Román-Roldán, "Removing noise and preserving details with relaxed median filters," *J. Math. Imag. Vis.*, vol. 11, no. 2, pp. 161–177, 1999.
[34] L. Khriji and M. Gabbouj, "Median-rational hybrid filters for image restoration," *Electron. Lett.*, vol. 34, no. 10, pp. 977–979, 1998.
[35] J. Romberg, H. Choi, and R. Baraniuk, "Bayesian tree-structured image modeling using wavelet domain hidden Markov models," *IEEE Trans. Image Process.*, vol. 10, no. 7, pp. 1056–1068, Jul. 2001.
[36] M. Crouse, R. Nowak, and R. Baraniuk, "Wavelet-based statistical signal-processing using hidden Markov-models," *IEEE Trans. Signal Process.*, vol. 46, no. 4, pp. 886–902, Apr. 1998.
[37] T. Zhang and C. Suen, "A fast parallel algorithm for thinning digital patterns," *Commun. ACM*, vol. 27, no. 3, pp. 236–240, 1984.
[38] L. Lam and C. Suen, "An evaluation of parallel thinning algorithms for character-recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 9, pp. 914–919, Sep. 1995.
[39] G. Borgefors and G. Sanniti di Baja, "Analysing nonconvex 2D and 3D patterns," *Comput. Vis. Image Understand.*, vol. 63, no. 1, pp. 145–157, 1996.
[40] T. Yang, *Cellular Image Processing*. Commack, NY: Nova, 2001.
[41] L. Kitchen and A. Rosenfeld, "Edge evaluation using local edge coherence," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, no. 5, pp. 597–605, Sep. 1981.



Paul L. Rosin received the B.Sc. degree in computer science and microprocessor systems from Strathclyde University, Glasgow, U.K., in 1984, and the Ph.D. degree in Information Engineering from City University, London, U.K., in 1988.

He was a Research Fellow at City University, developing a prototype system for the Home Office to detect and classify intruders in image sequences. He worked on the Alvey project "Model-Based Interpretation of Radiological Images" at Guy's Hospital, London, before becoming a lecturer at Curtin University of Technology, Perth, Australia, and later a Research Scientist at the Institute for Remote Sensing Applications, Joint Research Centre, Ispra, Italy, followed by a return to the U.K., becoming Lecturer with the Department of Information Systems and Computing, Brunel University, London. Currently, he is Senior Lecturer at the School of Computer Science, Cardiff University, Cardiff, U.K. His research interests include the representation, segmentation, and grouping of curves, knowledge-based vision systems, early image representations, low-level image processing, machine vision approaches to remote sensing, methods for evaluation of approximations, algorithms, etc., medical and biological image analysis, and the analysis of shape in art and architecture.