Image Processing using 3-State Cellular Automata

Paul L. Rosin
School of Computer Science
Cardiff University
Cardiff
UK
Paul.Rosin@cs.cf.ac.uk

correspondence to:
Paul L. Rosin
Cardiff School of Computer Science,
Cardiff University,
5 The Parade, Roath,
Cardiff, CF24 3AA,
UK
Tel: +44 (0)29 2087 5585; Fax: +44 (0)29 2087 4598

Abstract

This paper describes the application of cellular automata (CA) to various image processing tasks such as denoising and feature detection. Whereas our previous work mainly dealt with binary images, the current work operates on intensity images. The increased number of cell states (i.e. pixel intensities) leads to a vast increase in the number of possible rules. Therefore, a reduced intensity representation is used, leading to a three state CA that is more practical. In addition, a modified sequential floating forward search mechanism is developed in order to speed up the selection of good rule sets in the CA training stage. Results are compared with our previous method based on threshold decomposition, and are found to be generally superior. The results demonstrate that the CA is capable of being trained to perform many different tasks, and that the quality of these results is in many cases comparable or better than established specialised algorithms.

keywords: cellular automata, threshold decomposition, state reduction, feature selection, denoising

1 Introduction

In their simplest form, cellular automata (CA) are made up from a regular grid of cells, each of which can be in one of a finite set of states. At a given time step the state of each cell is updated in parallel, and is determined as a function of the values in the cell's neighbourhood during the previous time step, i.e. by a set of cell state transition rules. CA are discrete dynamical systems, and their simplicity coupled with their complex behaviour has made them popular for simulating complex systems. Cellular automata have a number of advantages over traditional methods of computations:

- 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 behaviour. That is, although each cell has an extremely limited view of the system (just its immediate neighbours), localised 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.
- This simplicity of implementation and complexity of behaviour means that CA can be better suited for modelling complex systems than traditional approaches. For example, for modelling shell patterns, CA were found to avoid the considerable numerical problems inherent with partial differential equation based models, and were also substantially faster to compute [1].
- CA are both inherently parallel and computationally simple. This means that they can implemented very efficiently in hardware using just AND/OR gates and are ideally suited to VLSI realisation [2]. A recent example of their application in computer vision was as part of a processing pipeline to compute stereo disparity maps at 275 frames per second [3]. The first step was to compute disparity using block matching based on the sum of absolute differences. Small block sizes were used to ensure high accuracy, but at the cost of producing many spurious responses. Therefore, the second step was to filter the disparity maps using a few simple CA rules, which provided better results than various other filtering schemes. The high frame rate was achieved by implementing both block matching and CA filtering on a single FPGA device (see [3] for details).
- CA are extensible; rules can easily be added, removed or modified.

CA is an area of active research – recent years have seen the introduction of several dedicated conferences and a specialist journal – but conversely, there has possibly been less work in applying (CA) to computer vision tasks compared to 25 years ago. One of the practical difficulties of using CA is the determination of the rule set. The traditional approach of specifying rules manually is obviously a slow and tedious process, and therefore can make the use of CA appear unattractive.

The 1990s saw several researchers applying evolutionary algorithms (typically genetic algorithms) to automatically generate rule sets, although most typically this was applied to synthetic problems (e.g. the Majority Problem) [4] or undemanding tasks such as boundary detection in binary images [5]. Also, in some cases the techniques were not applicable to image processing; for instance, Adamatzky [6] only considered the situation in which the desired rule set must exactly produce the target output (and continually increased the neighbourhood size until this was achieved). In contrast, for realistic image processing tasks, the goal should be to minimise some error term without any realistic expectations of ever driving it down to zero.

The more recent years have seen only limited progress. There are still papers attempting to solve the trivial task of binary image boundary detection using genetic algorithms and CA [7, 8]. Another example at the same level attempts (with limited success) to generate simple shapes such as a square, circle, etc., again using genetic algorithms [9]. Craiu and Lee [10] use a minimum description length criterion to automatically learn both neighbourhood size and rules for

stochastic CA to perform the task of synthesising binary patterns. However, the system exhaustively considers first all neighbourhood sizes and then all rule parameters, and so they could only demonstrate results on very small CA examples. Billings has developed several schemes for "CA identification", i.e. learning rules and neighbourhoods. However, his published results are limited to the task of identifying only a single rule that generated a synthetic binary pattern [11, 12].

Applying GAs remains a dominant theme in research into learning CA rules [13, 14]. An exception is Straatman *et al.* [15] who developed a form of hill climbing and backtracking to identify rules. However, again the published results were limited to small synthetic examples. Maeda and Sakama [16] used genetic programming to tackle a different but related problem given a sequence of configurations (i.e. the CA's grid of cell states at several time steps) they identified the rules that produce the sequence.

Our previous work on training binary CA for image processing tasks also took a different approach [17]. Rather than use an evolutionary approach such as genetic algorithms, a deterministic method was employed, namely sequential floating forward search (SFFS). The advantages of SFFS are 1/ it is simple to implement, 2/ not randomised, and therefore repeatable (both for the developer and other researchers attempting to duplicate published results), and 3/ it does not require the many parameters necessary for genetic algorithms. While this work concentrated on processing binary images it was also adapted to gray level images by applying the rules learnt on binary images to the set of binary images produced from gray level images by threshold decomposition. The focus of this paper is to develop an alternative and more direct approach to training and applying CA to gray level images. Testing and comparison of results will be made against the threshold decomposition approach as well as more traditional methods. The effectiveness and generality of the CA method will be demonstrated by applying it to a variety of low and mid level image processing tasks, namely: noise filtering, edge detection, connected set morphology, ridge and valley detection, and computation of the gray level convex hull.

2 Relating the Number of Cell States to the Number of Rules

For a 3×3 neighbourhood with cells taking 256 possible intensities there are 256^8 possible neighbourhood patterns (not considering the central cell's value). However, this number can be reduced by considering symmetries. To determine the number of distinct patterns after removing equivalent symmetric versions we use the Pólya-Burnside counting lemma [18]. If G is a set of permutations of a set A, then the number of equivalence classes is

$$N = \frac{1}{|G|} \sum_{g \in G} |\operatorname{Fix}(g)| \tag{1}$$

where $\operatorname{Fix}(g)$ is the number of elements of A that are invariant under g. Figure 1 shows the patterns which are invariant under examples of the desired transformations: $\pm 90^\circ$ rotation, 180° rotation, and mirror symmetry through a vertical line of reflection respectively. Thus, the number of distinct patterns N in terms of the number of possible intensities n is

$$N = \frac{n^8 + 2n^2 + n^4 + 4n^5}{8} \tag{2}$$

where the terms in the numerator correspond to: the identity transformation (i.e. 0° rotation), two rotations ($\pm 90^\circ$), a single rotation (180°), and four rotations corresponding to mirror symmetry through horizontal, vertical, and diagonal lines of reflection. Setting $n = 2$ for the binary image CA that our previous work focused on [17] we have $N = 51$ rules as previously reported. The problem with adapting this to a larger range of intensities is that it can be seen that even after eliminating symmetries the number of patterns still scales as $O(n^8)$ as before. And so, if

$n = 256$ intensities are to be represented then the number of patterns is greater than 2×10^{18}, which is obviously too large to feasibly enumerate practicably. Even if only a portion of the space of all combinations of rules is searched for a suboptimal subset of rules the large search space leads to problems. The available training data will be necessarily be sparse, making it difficult to find good solutions that will generalise to unseen test data. Moreover, in order to achieve a reasonable solution any optimisation method will probably require large memory and computation time overheads. Finally, a large number of states will generally require a large number of transition rules to be specified to achieve the desired effect, and so even after the rule selection process there could be problems with high storage requirements for the rules. Therefore, in this paper we will use a smaller number of cell states in order to maintain a more tractably sized search space.

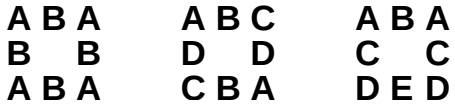


Figure 1: Patterns of a 3×3 neighbourhood that remain invariant under $\pm 90^\circ$ rotation, 180° rotation, and mirror symmetry through a vertical line of reflection respectively.

3 Threshold Decomposition

The high complexity arising from directly using pixel intensity values as cell states led our previous work to transform the image into multiple binary representations using thresholding decomposition [17]. This involves decomposing the gray level image into the set of binary images obtained by thresholding at all possible gray levels. A binary CA is applied to each binary image, and the results combined by adding the set of processed binary images. In that work the CA were trained to perform denoising on a *single* binary image and subsequently the learnt rule set was reapplied to a gray level image using threshold decomposition. Results of this approach will be shown in section 7.

In addition, the more computationally expensive approach will be taken of training the CA on gray level images. That is, a search is made for a set of rules that when applied to the elements of the threshold decomposed input image and reconstructed produces a gray level image that provides a good match to the gray level target image. This has the advantage that it directly optimises the desired error function unlike the first approach which does not use threshold decomposition in the training phase but only during the subsequent application phase. However, if the image has G gray levels then computation during training increases by a factor G, which is a substantial cost. Moreover, since the rule selection process is suboptimal, the results are not guaranteed to be superior to the first approach. Of course, there are trade-offs to be made, and it is possible to quantise the training image's intensities more coarsely to speed up training. Likewise, if desired, the learnt rules can be applied to images with reduced intensity resolution to speed up their application. For clarity, the first approach will be referred to as TD/1 as threshold decomposition is only involved at the application stage, while the second approach will be referred to as TD/2 since the threshold decomposition is involved twice.

4 A 3-State Representation

This paper introduces an alternative approach to reducing the number of cell states to enable more efficient training and application of CA to intensity images. It is based on the texture unit texture spectrum (TUTS) method of texture analysis [19]. This involves using a pixel's value as

a threshold for its eight neighbours. That is, for a central pixel value v_c its neighbours v_i are thresholded as

$$v_i' = \begin{cases} 0 & \text{if } v_i < v_c \\ 1 & \text{if } v_i = v_c \\ 2 & \text{if } v_i > v_c \end{cases} \tag{3}$$

Each neighbourhood can then be represented by a code formed from the eight ternary values: $\sum_{i=1}^{8} v_i' 3^i$, and histograms of the $3^8 = 6561$ different texture unit codes (the so called texture spectrum) make up the textural description of an image (or sub-image).

We propose to use this local thresholding approach in the context of our CA to reduce the large number of neighbourhood patterns. The basic idea is to maintain at each cell the image intensity as its primary state, but during the rule matching phase to consider the ternary pattern of the neighbourhood determined relative to the central cell's state. The CA rules are defined in terms of the 3 states rather than 256 states, and so from eqn. 2 we find that 954 patterns need be considered.

The scheme is as follows, and is illustrated in figure 2. The rules are applied to all the matching cells in parallel at each time step. This involves 1/ first generating at each cell its ternary neighbourhood pattern by thresholding the neighbourhood according to eqn 3. 2/ at each cell check for any rule that matches its ternary neighbourhood pattern, 3/ if a rule matches then apply it to update the central cell. In a standard binary CA, application of a rule inverts the cell's state. In the present system despite using ternary rules it is the cell's intensity that needs to be updated. This is done by modifying its value such that its ternary neighbourhood changes. The minimal modification in either direction is to either increase its intensity to its closest neighbourhood intensity value, or alternatively to decrease it to its closest neighbourhood value. Thus for each pattern there are two possible rules (i.e. increasing or decreasing types). Note that for each type of rule there are certain neighbourhood patterns that preclude the use of that rule type. That is, there are 51 of the 954 basic patterns without any neighbourhood pixel intensity greater than the central pixel, and the same number without a smaller value. Therefore, the total number of rules in this CA system is $2 \times (954 - 51) = 1806$.

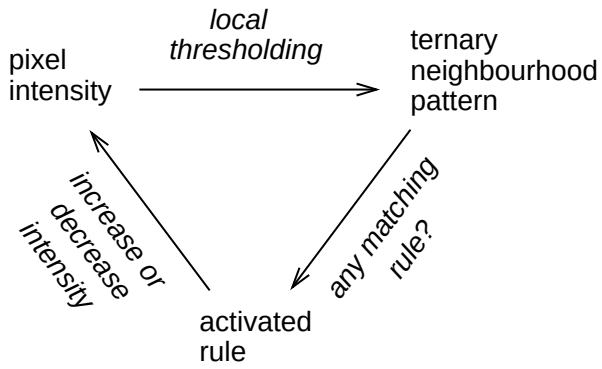


Figure 2: The CA is represented using both pixel intensities for cell states as well as ternary neighbourhood patterns. Transition rules involve changing a cell's intensity sufficiently such that the ternary neighbourhood pattern changes.

In recent years the local binary pattern (LBP) method has eclipsed the TUTS approach in terms of popularity. While LBP has some refinements such as multiple scales, rotation invariance, etc., its basic version is essentially the same as TUTS except that the neighbourhood is thresholded into two levels instead of three

$$v_i' = \begin{cases} 0 & \text{if } v_i < v_c \\ 1 & \text{if } v_i \geq v_c \end{cases} \tag{4}$$

(a) (b) (c)

Figure 3: Problems with the LBP CA approach. (a) input image, (b) target image, (c) best output image achieved.

thereby providing a more compact description.

It is tempting to simplify the CA representation yet further in the same manner. Thus the CA rules could be defined in terms of the binary neighbourhood patterns, and so only the 51 patterns used previously in the binary CA [17] would need to be considered. However, this is unfortunately unattractive on two counts. First, in many applications such as the denoising example in subsection 7.1, if the results of processing image I are $f(I)$ then processing the inverted image $-I$ should produce $f(-I) = -f(I)$. However, due to the asymmetry in eqn 4 this does not hold true for the LBP CA system. Secondly, and more seriously, the representation is too limited, and leads to problems of inconsistency. This can easily be demonstrated in the small test example in figure 3. Without loss of generality, for simplicity the image only contains two values. The task is to eliminate the two instances of salt and pepper noise, which consists of the single dark pixel in the light half of the image, and vice versa. However, the binary neighbourhood pattern at the isolated dark pixel on the left (which is all ones) is identical to the binary neighbourhood patterns of the dark pixels on the right. In fact, it is not possible to distinguish between the isolated dark pixel and a dark pixel with *any* combination of dark or light pixels in its neighbourhood (so long as the dark pixels are not darker than the central pixel) since they all produce the same LBP code. This implies that it is not possible to find a set of rules that could process the input image in figure 3a to achieve the target image in figure 3b. Removing the single "pepper" pixel would require the same "increasing" rule to be applied to *all* the dark pixels. Only those adjacent to a light pixel would actually be altered; nevertheless this would still incorrectly affect a large number of pixels. Consequently, the best solution found within the LBP CA system is to avoid this drastic change in favour of removing only the "salt" while failing to remove the "pepper".

Given the above problems we use the ternary CA scheme in this paper rather than the binary CA. Even though it will require many more rules than the latter, it has the advantage that it overcomes the latter's inconsistencies. Two variations to the scheme are possible. The first allows all 1806 rules to be learnt independently. Thus there is no constraint that $f(-I) = -f(I)$, which can be useful in some situations (e.g. computing the gray level convex hull in subsection 7.5). The second variation enforces the constraint by treating the "increasing" and "decreasing" versions of a rule, $R^{inc}_{TU_i}$ and $R^{dec}_{TU_i}$, as equivalent when applied to inverted neighbourhood intensities. That is, for a neighbourhood N, $R^{inc}_{TU_i}(N) \equiv R^{dec}_{TU_i}(-N)$ and $R^{inc}_{TU_i}(-N) \equiv R^{dec}_{TU_i}(N)$, and so there are only 903 rules to be learnt. In this paper we will refer to the two variations as TU/2 and TU/1 respectively since they use essentially one set of two sets of rules.

One of the features of the TUTS (and LBP) schemes is that they provide a textural description that is invariant to a wide range of gray level transformations. Thus, for the TU CA this means that $f(\mathcal{T}(I)) = \mathcal{T}(f(I))$ where $\mathcal{T}$ is a monotonic, non-linear mapping that is information preserving in the sense that separate intensities are not collapsed into single values. In some situations this could be beneficial, but in others a lot of important information is lost by discarding quantitative information, and so the TU CA is not suited to solve certain quantitative tasks, e.g. computing edge magnitudes.

5 Rule Optimisation

As discussed previously, an issue with training the CA is the high dimensionality of the search space in determining which subset of possible rules achieves the image processing task most effectively. Even if the number of states is reduced from 256 to 3 using the above scheme, and symmetries are eliminated, then there still remain $2^{1806} \approx 5 \times 10^{543}$ possible subsets. As before, we will find a suboptimal solution using sequential floating forward search (SFFS) [20, 17]. SFFS is a greedy algorithm that starts from the empty set of rules and performs a series of forward steps, each of which adds the transition rule from the available set of patterns to the rule set such that it best improves the objective function. Interleaved with the forward steps are backward steps, which remove the rule from the rule set that best improves the objective function. After each forward step, SFFS performs backward steps as long as the objective function increases.

Although SFFS is faster than approaches such as branch and bound it is still moderately costly. Moreover, the cost of each iteration of the forward and backward steps is proportional to the number of patterns. This means that, given the larger number of patterns in the ternary state CA compared to the binary state CA used previously, SFFS will be substantially slower than before (20 or 40 times depending on whether inverse rules are considered equivalent – see section 4). This has led to two approaches to improve efficiency. The first is based on the observation that the distribution of TU codes is very uneven. A histogram (with the bins reordered by size) of the codes in the image in figure 6 is shown in figure 4 and reveals that the majority of the neighbourhood patterns occur only rarely. This suggests that an initial SFFS phase could be run using only the top N most common patterns (as estimated from training data), and then the learnt rule set would be extended and refined by applying SFFS to the full set of patterns. Experiments were performed using $N = 100$. While there was a speed up over basic SFFS, better results were obtained using the second approach, and so the former will not be discussed further.

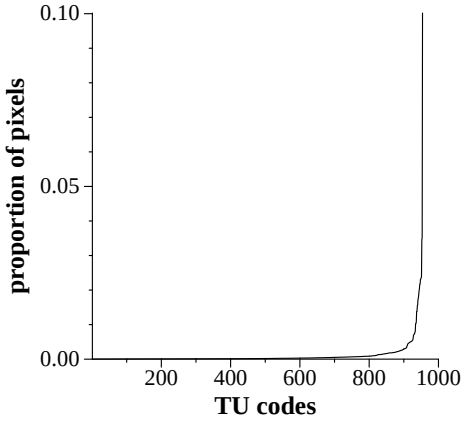


Figure 4: Reordered distribution of all the neighbourhood patterns (TU codes) that appeared in the image shown in figure 6 demonstrating that many codes are rare.

The second approach was inspired by one of the extensions of LBP: "uniform" patterns [21]. This involved counting the number of transitions between zero and one when traversing the thresholded neighbourhood in angular (e.g. clockwise) order. "Uniform" patterns were defined to have two or less transitions. Ojala *et al.* observed that "uniform" patterns accounted for the majority of neighbourhood patterns in many images, and so grouped all the remaining patterns (with 3-8 transitions) into a single code. For our purposes, ignoring the distinctions between the more complicated neighbourhood patterns is undesirable, as it would (like the basic LBP approach) provide an impoverished representation. For instance, a one pixel width line through

the neighbourhood centre could not be properly represented. Nevertheless, we make use of the idea of grouping the patterns into a small set of classes. Specifically, ternary neighbourhood patterns with same number of zeros and twos are put into the same class, making up 45 classes in total. To keep the number of classes small the groups are not further subdivided by the number of ones (corresponding to neighbour intensities equal to the central intensity) in the neighbourhood pattern. SFFS is then applied to the rule classes such that all the rules in a class are considered for inclusion or removal by SFFS as a whole. The motivation of the approach is to replace the slow process of individually testing each potential rule by the faster process of testing groups of rules. The goal is to improve computation time without sacrificing the quality of the rule set too much. Of course, this initial rule selection stage is relatively crude, and so now a further refinement stage is necessary. This is made up of interleaved forwards and backwards steps, in which rules are added or removed from the rule set. Again, to speed up the process, more than one rule at a time is added or removed. For the forward step all the unused rules are tested, and the value of the objective function achieved when each rule is included is recorded. Now, rather than just adding the best rule (as in SFFS) all the tested rules that individually provide an improvement on the objective function before the forward step started are ranked by their objective function scores, and the top fractions $\frac{1}{1}$, $\frac{1}{2}$, $\frac{1}{4}$, ... of these ranked rules are tested as a group for inclusion in the rule set. The fraction yielding the best objective function is then retained in the rule set. The analogous process is carried out on the backwards step. To maintain efficiency, the interleaved forwards and backwards steps are repeated at most 50 times, or until the objective function value stops improving.

6 Objective Functions

The objective function used to drive the rule selection obviously has a crucial effect on the final results. The simplest and most obvious scheme is to use the root mean squared (RMS) error between the processed and the target image. However, it is well known that RMS (and related) values have limitations. In particular, they often do not capture perceptual similarity. Therefore, we also consider here the SSIM index which measures image similarity taking luminance, contrast and structure into account [22]. Following Wang *et al.* the constants are set to $C_1 = (0.01 \times 255)^2$ and $C_2 = (0.03 \times 255)^2$, and the similarity between images A and B is defined as

$$SSIM(A,B) = \frac{(2\mu_A\mu_B + C_1)(2\sigma_{AB} + C_2)}{(\mu_A^2 + \mu_B^2 + C_1)(\sigma_A^2 + \sigma_B^2 + C_2)}.$$

The SSIM index is then applied locally and the mean over the image used as the final similarity measure. In [22] the authors use an 11×11 circular-symmetric Gaussian weighting function to compute the local SSIM index. Since the objective function will be extensively used in the training process we speed up its computation by eliminating the weighting function and applying the local SSIM index directly on 8×8 windows, as in the related "universal quality index" [23].

7 Examples and Experiments

As discussed in section 5, increasing the number of CA states from 2 to 3 would significantly increase the computational cost of rule optimisation, which led us to develop the two stage rule block version of SFFS. The effect of the different CA schemes and also the different rule optimisation strategies on computational time is demonstrated more clearly in the following experiment based on the denoising task described in the next subsection. Programs were run on a 2.40 GHz Intel Core 2 Duo with 4GB of RAM. To avoid excessive runtime the various CA are timed training on several small image patches for a few iterations, and these times were

extrapolated. The timings were averaged over ten subimages, each containing 100×100 pixels, and ran for up to 10 iterations (or less if they converged early). Table 1 shows these actual timings as well as their extrapolation (by linear scaling) to 1000×1000 pixels with 100 iterations. The 2 state CA using a single image (TD/1) is of course much faster than the other schemes. The 2 state CA using threshold decomposition (TD/2) could potentially be 256 times slower than TD/1, but in practise it was only 61 times slower. This is because many of the thresholded images at high or low threshold values tend to have little significant image content, and therefore the CA converges quickly in these cases. If the full SFFS rule optimisation approach that was used for the 2 state CA was applied to the 3 state CA (TU/2) then training on full sized images would take 1.67 years! In contrast, while the rule block approach is still computationally expensive, it is substantially faster. Although we have not actually run the TU/2 CA for a year or more, the extrapolation seems reasonable as the extrapolated timings of the other faster CA match their actual timings on larger images.

Of course, while training the CA is expensive, applying a learnt set of rules is much faster. For a 1000×1000 image, running 100 iterations typically takes about 30 seconds currently, although this could be substantially improved by optimising the code.

Table 1: Timings for training CA – alternative units (seconds/hours, minutes/days) provided for convenience. Actual timings were averaged over 10 images, of size 100×100, running for 10 iterations. Also, timings were extrapolated to 1000×1000 images for 100 iterations.

version of CA	**actual timings**		**extrapolation**	
	seconds	**hours**	**minutes**	**days**
TD/1 CA	2.84	0.00	47.36	0.03
TD/2 CA	175.53	0.05	2925.47	2.03
TU/2 CA (RMS) – full SFFS	52626.82	14.62	877113.65	609.11
TU/2 CA (RMS) – block SFFS	1167.62	0.32	19460.37	13.51
TU/2 CA (SSIM) – block SFFS	1145.41	0.32	19090.2	13.26

7.1 Denoising

In this subsection we demonstrate the training and application of the CA to denoising. The results of filtering different types and magnitudes of noise which have been added to three images (`barbara, couple`, and `venice`) are listed in table 2, with the results for the `barbara` image also displayed as a barchart in figure 5. Comparison is made with the 3×3 median filter and two of its variants: the median-rational hybrid filter (MRHF) [24] and the adaptive median (AM) filter [25]. The former combines three bidirectional median filters, while the latter increases its window size until the median output is not a maximum or minimum of the window. In addition, comparison is made with two techniques based on hidden Markov trees (HMT) applied to wavelet coefficients. The first constructs a "universal" model [26] of the image, while the second estimates the model parameters and applies a Wiener filter with those parameters [27]. Further comparison is made with methods based on the complex shock filter [28] and on minimising total variation [29].

The number of rule iterations was generally set to 100 for the CA. Appropriate parameter values for the other methods were set as necessary, in some cases selected to optimise results on each test image, thereby optimistically biasing their results.[1] The CA were trained on the

[1]The number of iterations for the median filter was chosen so as to produce the best results for each test image.

Table 2: RMS errors of denoised gray level images corrupted by various types and levels of noise. Each block of three rows lists results for three noisy images: `barbara, couple` and `venice`. For the median filter and total variation methods the optimal number of iterations determined for *each test* image. Likewise, the two parameters (h and k) for the MRHF filter were optimised for each test image. The CA used the rules that were learnt on training data corrupted with the corresponding noise model. For each row (i.e. each noisy image that was filtered) the lowest RMS value is highlighted.

noise model			raw	median	AM	MRHF K=0 h=	CA TD/1	CA TD/2	CA TU/2	HMT U	HMT W	shock filter	total variation
Gaussian	$\sigma = 10$		10.00	14.48 (1)	11.37	8.23 (4)		8.21	8.72	10.60	**6.90**	12.34	8.10 (10)
			9.97	8.91 (1)	8.88	6.80 (3)		7.47	7.79	7.22	**6.38**	8.99	6.63 (25)
			9.73	6.94 (1)	7.84	5.88 (2)		6.93	7.31	5.93	5.59	8.85	**5.43** (50)
	$\sigma = 25$		24.64	16.74 (2)	19.87	15.09 (3)		15.41	15.39	15.42	**11.64**	16.79	14.42 (100)
			24.71	12.31 (2)	18.43	12.76 (2)		12.25	12.51	11.96	10.88	14.64	**10.83** (100)
			23.26	10.62 (3)	17.13	11.48 (2)		10.96	11.35	10.79	10.06	13.76	**9.08** (100)
salt and pepper	single pixel	$p = 0.1$	43.94	14.66 (1)	**9.32**	12.82 (3)	11.97	20.83	10.21	19.27	24.02	16.54	16.69 (200)
			42.73	8.40 (1)	**5.23**	8.62 (2)	9.56	19.41	6.58	16.21	19.87	13.44	11.97 (200)
			47.59	6.40 (1)	**3.71**	7.71 (2)	9.26	19.33	5.42	17.56	23.41	13.79	11.32 (200)
		$p = 0.6$	107.65	23.72 (24)	**17.93**	62.42 (2)	23.36	22.42	18.11	39.14	40.57	54.00	30.05 (800)
			104.60	19.76 (15)	13.32	59.38 (2)	19.64	18.93	**12.80**	32.72	34.33	52.00	26.55 (800)
			116.37	19.83 (21)	12.47	66.79 (2)	19.96	18.80	**11.35**	52.06	52.95	59.11	32.84 (1600)
	3×3 block	$p = 0.01$	41.09	17.85 (10)	11.63	36.30 (2)	13.13	13.08	**9.75**	33.22	39.68	36.47	22.69 (800)
			40.04	13.55 (7)	7.97	33.24 (1)	12.11	11.02	**7.95**	33.17	39.01	34.48	19.62 (800)
			44.45	12.75 (11)	**6.91**	35.58 (1)	13.29	11.71	8.49	35.64	43.45	38.20	21.16 (1600)
		$p = 0.1$	108.23	76.64 (47)	24.34	97.85 (2)	25.01	25.51	**21.01**	60.23	97.83	93.03	37.87 (3200)
			105.38	73.94 (46)	20.77	94.80 (2)	22.64	23.38	**17.84**	56.76	95.07	90.32	32.05 (3200)
			117.59	82.15 (46)	21.71	106.08 (2)	26.04	26.96	**17.65**	70.31	107.46	101.62	42.30 (3200)
stripes	$p = 0.2$		37.33	21.68 (34)	20.90	27.25 (2)	18.15	17.33	**13.81**	27.39	36.10	25.73	24.76 (200)
			34.75	16.37 (4)	15.96	22.03 (1)	13.48	12.31	**10.14**	26.03	34.20	21.14	18.53 (200)
			43.99	21.60 (7)	23.11	28.45 (1)	21.08	19.18	**11.23**	32.83	43.22	26.92	24.78 (200)
	$p = 0.8$		74.61	50.26 (12)	55.19	58.33 (2)	48.63	47.16	**28.32**	48.61	68.73	54.91	52.37 (1600)
			67.92	40.80 (11)	46.30	49.91 (2)	37.96	37.22	**20.87**	41.09	62.29	46.55	43.77 (1600)
			87.78	61.30 (12)	66.89	70.16 (2)	53.08	54.01	**34.01**	60.58	80.89	65.59	61.21 (800)

1536×1024 image mosaic shown in figure 6 except for TD/2 CA which used a 512×512 subset due to memory limitations. All were then tested on the unseen noisy `barbara, couple` and `venice` images.

The RMS errors resulting from denoising are listed in table 2, with the best result for each test image given in bold font. Since TU/2 CA always gave lower errors than TU/1 CA the latter is not included in the table. The CA are outperformed when the noise is Gaussian; the HMT with Wiener filter and the total variation methods performed best in these circumstances. For small amounts of salt and pepper the adaptive median performed best, but for larger amounts the TU/2 CA performed best. For larger structured noise the TU/2 CA also performed best. The first test was on salt and pepper noise using 3×3 blocks. The TU/2 CA outperformed the adaptive median even though the CA only used a 3×3 neighbourhood in contrast to the (dynamically selected) larger windows of the adaptive median. Note that the standard 3×3 median performed poorly even when a large number of iterations were applied to maximise performance. The second type of structured noise was to randomly replace (with probability p) each row or column in the image with a random intensity (constant along the stripe). Thus the probability of corrupting a pixel is $p - \left(\frac{p}{2}\right)^2$.

The resulting outputs from the various methods are displayed for a small subwindow in fig-

For the MRHF the best value for k was found to be zero except for the case of Gaussian noise with $\sigma = 10$, where the values 0.03, 0.01 and 0.01 produced the best results. Gilboa's code for his complex shock filter and Rudin *et al.*'s total variation method were used with their default parameters; for the complex shock filter: number of iterations = 30, $|\lambda| = 0.1$, $\tilde{\lambda} = 0.2$ and $a = 2$, and for total variation: $\Delta t = 0.2$, $\epsilon = 1$ and $\lambda = 0$, while the number of iterations was selected to optimise the result.

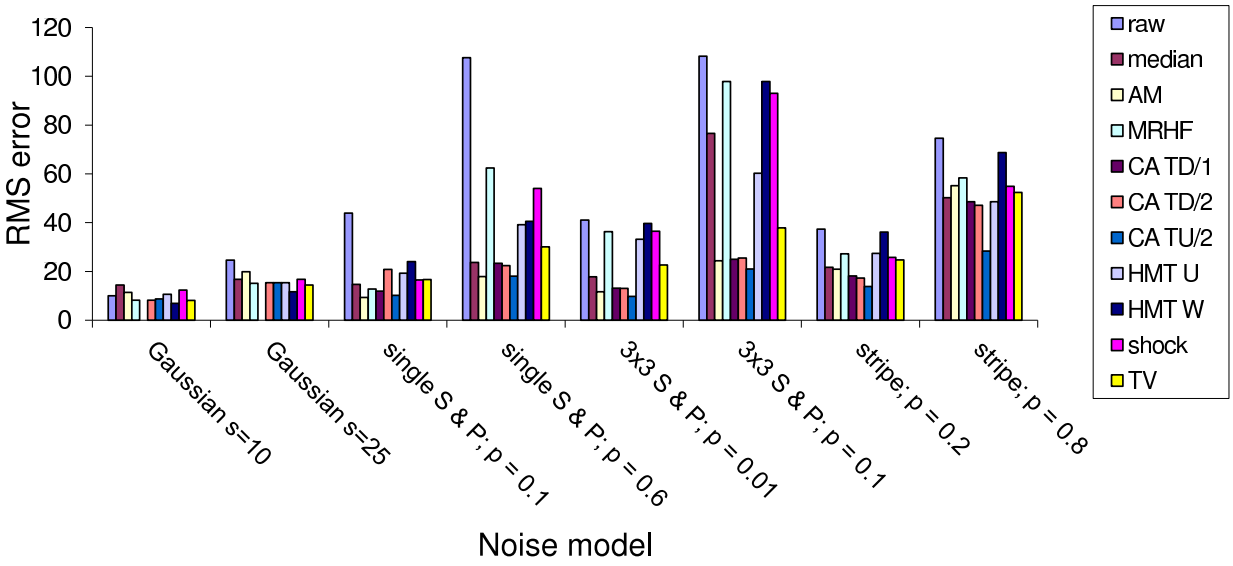


Figure 5: The RMS errors from table 2 are displayed as a barchart for the denoised versions of the `barbara` image generated from each noise model.

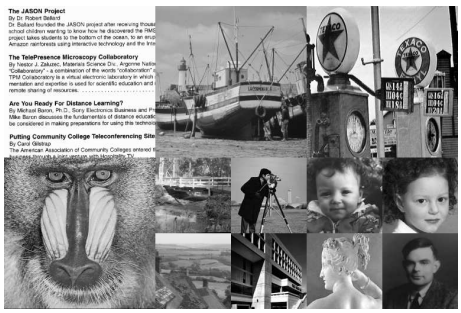


Figure 6: Ground truth training image (without added noise) used for denoising experiments.

ure 7. It is apparent that the TU/2 CA copes very well with all types of impulse noise, and in the extreme case of the stripe noise does very much better than the non-CA approaches. On the other hand, for the Gaussian noise the CA's non-linear filtering is less of an advantage. The loss of most of the image intensity information causes TU/2 CA to have slightly higher errors than TD/2 CA. Nevertheless, both TD/2 and TU/2 CA perform reasonably, even for Gaussian noise, outperforming the median, adaptive median and shock filters.

Since, for each of the impulse type noise models that was used for training, the proportion p of corrupted pixels is known, this suggests that when processing unseen images with this same type of noise only that given proportion of pixels should be modified. Therefore, we have considered a post-processing step in which the $1 - p$ proportion of filtered pixels that have the smallest intensity differences from the input image are reset to their unfiltered values. Results are just presented for the adaptive median and the TU/2 CA in table 3; a general improvement (indicated by underlined values) for both of these non-linear filters is evident. Of course, it would also be possible to include the post-processing as part of the training process that is optimised.

Training of the CA as well as optimisation of the median, MRHF and total variation methods was also carried out using the SSIM objective function. The best methods for each noise model remained essentially the same, except that the adaptive median now outperformed the TU/2 CA on high levels of single pixel salt and pepper noise ($p = 0.6$). However, in general the change from RMS to SSIM objective function did not have a substantial effect on the resulting denoised

(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l)

Figure 7: Results of denoising; for those methods involving optimisation the RMS error criterion was used. a) subwindow of input image, b) subwindow of target image, c) median filter, d) adaptive median filter, e) MRHF, f) TD/1 CA, g) TD/2 CA, h) TU/2 CA, i) universal HMT, j) Wiener HMT, k) complex shock filter, l) total variation. Rows correspond to the following noise models: Gaussian $\sigma = 25$; salt and pepper $p = 0.6$, single pixel; salt and pepper, 3×3 block, $p = 0.01$; stripe $p = 0.8$.

images – see figure 8.

7.2 Edge Detection

In this subsection we demonstrate training and application of the CA to edge detection. To create a good set of training data a 750×750 image mosaic was created using sub-images from the University of South Florida data set which contains images along with manually generated ground truth edges. Since there is likely to be some positional error in the ground truth edges (which are one pixel wide) the target edge map was dilated twice, with the new edges set each time to an increasingly lower intensity. This process is similar to blurring the edge map whilst avoiding creating local intensity maxima at junctions. Note that all edge maps are inverted in the paper for display purposes.

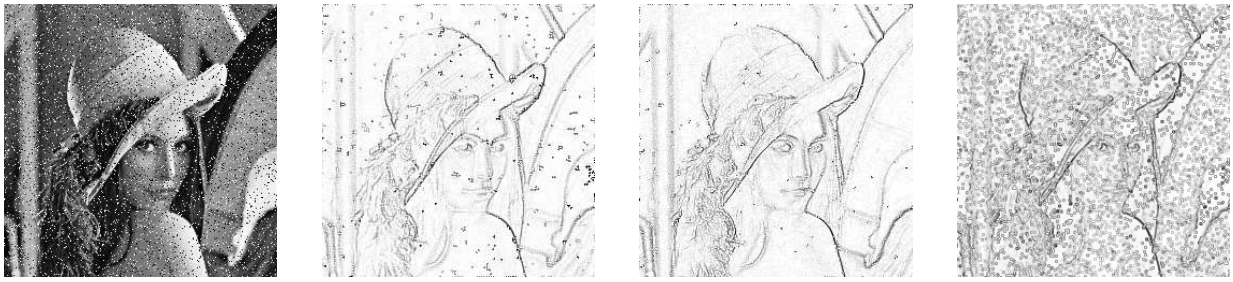
As expected, the TU/2 CA fails to perform well. Since the TU representation collapses the

(a) (b)

Figure 9: Training data for edge detection task. (a) input image, (b) target image (inverted for display purposes).

(a) (b) (c)

Figure 10: Edge detection using the TD/2 CA. (a) results from CA trained on dilated manual ground truth (figure 9b), (b) results of Sobel edge detector, (c) results from CA trained on Sobel edge detector applied to figure 9a.



(a) (b) (c) (d)

Figure 11: Edge detection using the TD/2 CA. (a) input image: `lena` with 0.1 probability salt and pepper noise, (b) results from CA trained on corrupted version of figure 9a, (c) results from CA trained on corrupted version of figure 9a applied to positive and inverted versions of `lena`, (d) results of Sobel edge detector.

Table 3: RMS errors for adaptive median and TU/2 CA before and after post-processing (+PP) step which resets some filtered pixels to their unfiltered values. Improvements are shown underlined.

noise model			AM	+PP	CA	+PP
salt and pepper	single pixel	$p = 0.1$	**9.32**	8.90	10.21	**8.70**
			5.23	**4.41**	6.58	5.65
			3.71	**3.58**	5.42	5.20
		$p = 0.6$	**17.93**	17.93	18.11	**17.87**
			13.32	13.33	**12.80**	**12.35**
			12.47	12.47	**11.35**	**11.32**
	3×3 block	$p = 0.01$	11.63	11.34	**9.75**	**9.67**
			7.97	**7.44**	**7.95**	7.88
			6.91	**6.81**	8.49	8.46
		$p = 0.6$	24.34	24.37	**21.01**	**20.58**
			20.77	20.80	**17.84**	**17.20**
			21.71	21.71	**17.65**	**17.54**
stripes		$p = 0.2$	20.90	20.90	**13.81**	13.81
			15.96	15.93	**10.14**	**10.09**
			23.11	23.13	**11.23**	**11.20**
		$p = 0.8$	55.19	55.19	**28.32**	28.42
			46.30	46.30	**20.87**	**20.84**
			66.89	66.89	**34.01**	34.05

image intensities down to three states there is insufficient information to compute edge magnitudes.

The TD/2 CA is more effective. Applying the rule set (restricted to rules that flip white pixels) learnt on the training data shown in figure 9 to the `lena` image produces an edge map (figure 10a) that looks very similar to the output of the Sobel edge detector (figure 10b). Even though the target edge map was essentially binary in nature it was not possible to eliminate edge magnitude from the trained CA output without compromising the quality. The results shown are for rules learnt using the RMS objective function. When the SSIM index was used instead, the CA failed to detect edges since the SSIM index is more sensitive to the qualitative difference in the edge magnitude map that the CA is capable of producing and the binary target edge map.

We note that the rule set that generated figure 10a is exceedingly simple, since it consisted of a single rule! The rule specifies that any pixel in a 3×3 homogeneous neighbourhood is flipped. Since in this version of the CA the rule is only applied to white pixels, then all white pixels are replaced by black except for pixels adjacent to black pixels in the input image. This leaves a black image with a one pixel wide white strip along the original black/white transitions, which when summed at the reconstruction stage of the threshold decomposition produces the edge magnitudes.

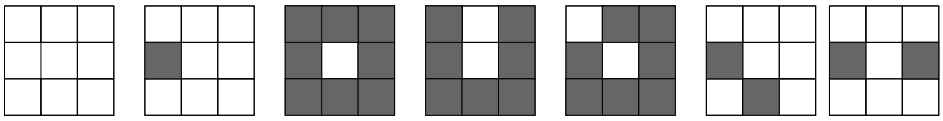
For comparison, the TD/2 CA was retrained on figure 9a using a target output provided by the Sobel edge detector. The results remain very similar – see figure 10c. Although 10 rules were learnt most had little effect apart from the one above.

(a) TD/2 CA (b) TU/2 CA (c) TD/2 CA (d) TU/2 CA (e) TD/2 CA (f) TU/2 CA

Figure 8: Each image pair provides a comparison of denoising with CA trained using RMS (left) versus SSIM (right) objective functions. Results for noise models: (a) & (b) Gaussian $\sigma = 25$; (c) & (d) salt and pepper, 3×3 block, $p = 0.01$; (e) & (f) stripe $p = 0.8$.

Next, the TD/2 CA was trained again using the ground truth edges in figure 9b, but with the input image corrupted by salt and pepper noise with occurrence probability of 0.1. The goal was to learn a rule set that could simultaneously reduce the effect of noise whilst detecting edges. Figure 11 shows the results of applying the rules to the `lena` image which was corrupted in the same manner. In comparison to the Sobel edge detector the TD/2 CA is much more successful in being able to robustly detect the edges, the results being only minimally affected by the salt and pepper noise (compare figure 11b and figure 11d). Isolated salt noise is very effectively removed (using the rule shown below). However, because only white pixels are flipped the CA is unable to deal with pepper in the same way, which is why they remain, but in an attenuated fashion. This suggests a simple solution: the inverted image is also edge detected, and while the edges will be in common, the unfiltered salt or pepper will not. The logical AND operation is applied to the two edge maps to remove these inconsistent noise artifacts. In practise, there is a small shift between the edge maps, so that the true edges also become somewhat attenuated by the AND. Nevertheless, the result in figure 11c (which has been gamma corrected by $\gamma = 0.4$ to provide a more similar balance with the other edge maps) shows that this approach was successful in removing further noise without significantly compromising the quality of the edges.

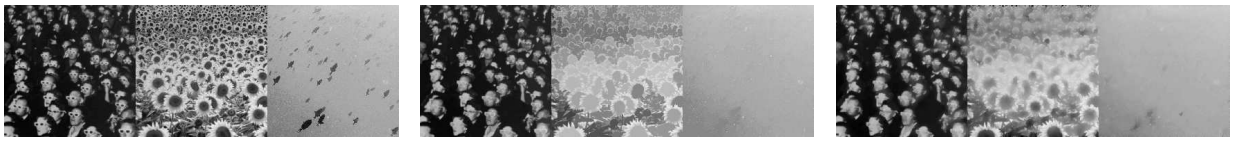
The learnt rules are shown below, ordered from left to right according to their importance (i.e. their effect on the RMS values). For each rule the neighbourhood pattern of eight white and/or black (displayed as gray) pixels is given.



The first rule performs the edge detection as before. The next rule does the same task, but allows the presence of a single noisy neighbouring pixel. The following three rules eliminate salt-like noise pixels. The remaining rules have much less effect, and essentially perform edge detection like the first rule, but in the presence of two noisy neighbouring pixels.

7.3 Connected Set Morphology

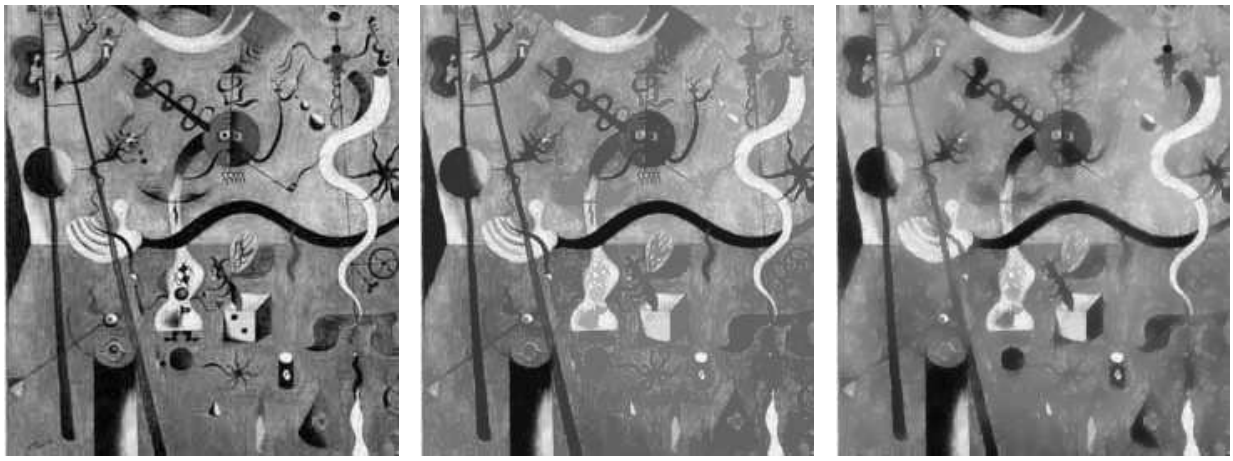
In this subsection we demonstrate the training and application of the CA to perform connected set morphology – in particular the closing operation. In connected set morphology, rather than use fixed structuring elements, the operations are applied to connected components. A binary area closing fills all background components with an area smaller than the given threshold, while a gray scale closing assigns each point the lowest threshold at which it belongs to a connected background component with an area greater than or equal to the given threshold [30].



(a) (b) (c)

Figure 12: Gray level connected set closing. (a) input image, (b) target image generated using a closing with an area threshold of 512, (c) results using the TU/2 CA with the RMS objective function.

The training data was made up from a 570×190 input image mosaic plus the target output which was generated by applying a conventional connected set closing with an area threshold of 512 (see figure 12). The best result was obtained using the TU/2 CA with the RMS objective function. Like the closing, it can be seen (figure 12c) to have eliminated the majority of spectacles in the cinema scene, the sunflower seeds in the middle, and the fish on the left, while retaining the majority of the rest of the detail. The TU/2 CA with the SSIM objective function produced similar, but slightly worse, results. The TD/2 results were distinctly worse; compared to the closing they failed to remove some details but incorrectly removed others. Further testing of the rules on an unseen image is given in figure 13. The TU/2 CA and closing results can be seen to be very similar in the main, although the CA has missed or insufficiently modified some dark regions.



(a) (b) (c)

Figure 13: Gray level connected set closing. (a) input image, (b) image generated using a closing with an area threshold of 512, (c) results using the TU/2 CA trained on figure 12.

7.4 Ridge and Valley Detection

Our previous work demonstrated the binary CA performing thinning [17]. Here it is adapted to intensity images, in which the goal is to perform gray level thinning to produce something akin to intensity ridge/valley detection. Training data was simply obtained by drawing some arbitrary curved and intersecting lines, which are initially binary, and blurring them to provide the input image. The target output was created by using the original binary lines as a mask for the blurred image so that pixels not directly underneath the original line were set to background. To simplify the tasks the goal was just to learn rules for finding ridges; valleys could then be generated by applying the same rules to the inverted image. Since thinning works by successive erosions, a large

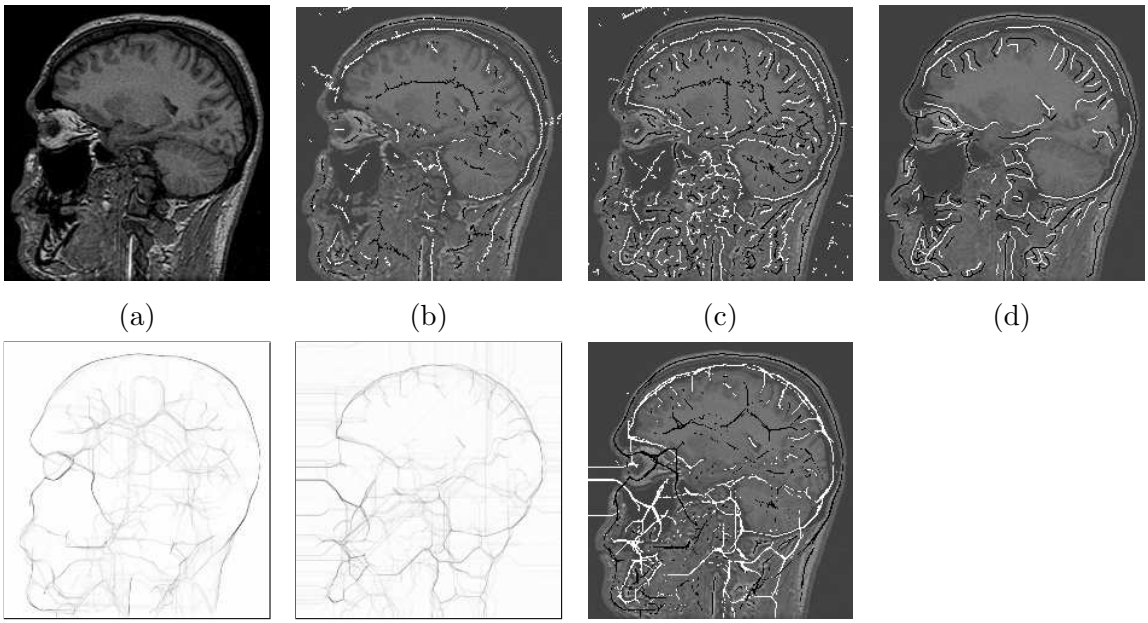


Figure 14: Ridge and valley detection. a) input MRI image, b) overlay of TU/2 CA applied after Gaussian smoothing, $\sigma = 4$, c) overlay of TU/2 CA applied after Gaussian smoothing, $\sigma = 2$, d) overlay of results from Steger's method, $\sigma = 2$, hysteresis thresholds = $\{2, 4\}$, e) ridge magnitudes produced by TD/1 CA, $\sigma = 2$, f) valley magnitudes produced by TD/1 CA, $\sigma = 2$, g) overlay of thresholded TU/2 CA ridges.

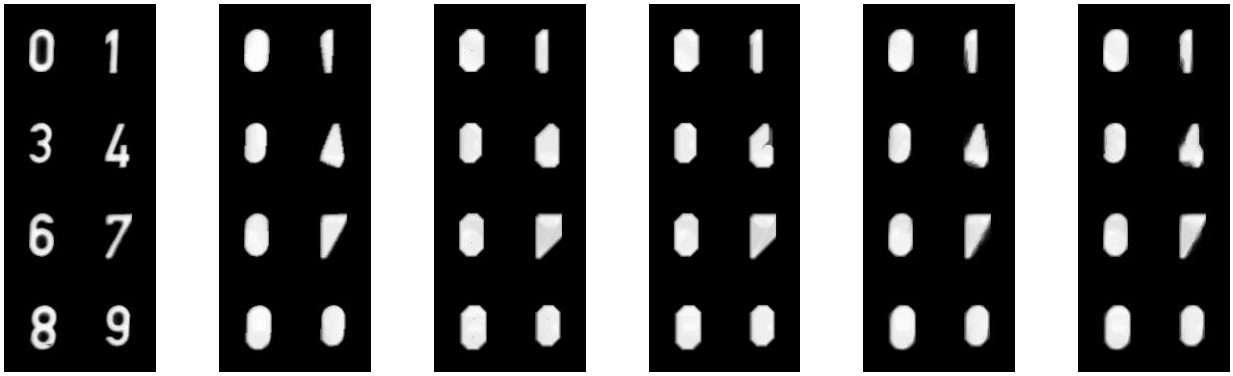
number of iterations of the rule set may be necessary, and so an upper limit of 1000 iterations was set rather than the default value of 100 used for many of the other experiments reported in this paper.

Like edge detectors, ridge/valley detectors would normally require quantitative measurements of intensity. As discussed in subsection 7.2 this is a problem for the TU/2 CA. However, here we overcome the limitation by the use of pre- and post-processing, specifically by smoothing the image to eliminate minor intensity fluctuations, and then thresholding (using Tsai's algorithm [31]) the CA result to eliminate low intensity ridges (or equivalently high intensity valleys). Otherwise, the loss of intensity magnitude information would result in *all* ridges and valleys being found, which would make the method extremely sensitive to noise and minor details.

Having trained the CA on the simple synthetic data we now show that it generalises successfully to real data such as the MRI head scan in figure 14a. Figure 14b and 14c show the effect of smoothing the MRI head image by different amounts and applying the TU/2 CA; ridges/valleys are overlaid on the original in black/white. Similar results were obtained using either the RMS or SSIM index objective functions. The results are reasonable – being generally thin and in the correct place – although comparison with Steger's specialised operator [32] in figure 14d reveals the CA ridges/valleys to be more fragmented. This will be due in part to the Steger implementation benefiting from hysteresis based thresholding, which aids good continuity.

The binary CA rules previously learnt for thinning were applied using threshold decomposition to the MRI and its inversion. As seen in the edge detection example (subsection 7.2) the TD/1 CA has the advantage that it can generate feature magnitudes, and the ridge/valley strengths are shown in figure 14e and 14f. When thresholded and overlaid (figure 14g) the results can again be seen to be mostly reasonable, although it can seen of course that the some undesirable valleys have been detected in the background and the features are not always thin.

7.5 Grey level Convex Hull



(a) (b) (c) (d) (e) (f)

Figure 15: Grey level convex hull. a) input image, b) target image, c) result using TD/1 CA, d) result using TD/2 CA, e) result using TU/2 CA with RMS objective function, f) result using TU/2 CA with SSIM objective function.

In this subsection we demonstrate training and application of the CA to compute the gray level convex hull, which is used to describe shapes of gray level objects. It can be considered to be the result of applying threshold decomposition and computing the traditional convex hull at each binary slice [33]. An alternative definition is given by Nyström *et al.* [34]. Figures 15a and 15b show the input and output images obtained by applying Soille's algorithm.

Since the convex hull distinguishes between object and background the binary CA only applied rules to invert black pixels (leaving white pixels untouched) while TU/2 CA (which does not consider inverted rules to be equivalent) was used.

Using the binary CA rules previously learnt for computing an approximate (eight sided) convex hull [17] and applying them to the gray level image using the TD/1 scheme produces a reasonable result as expected, given Soille's definition (figure 15c). The TD/2 CA scheme's results are similar but less successful on the concavity of the digit '4' – see figure 15d. While unexpected, this degradation in quality must be attributable to the non-optimality of SFFS. The TU/2 CA which operates more directly on the intensity values gave results that matched the target much more closely (figures 15e and 15f) although they are no longer perfectly convex. Whereas only 6 rules were required for the TD/1 CA, the TD/2 CA learnt 19 rules (which included all of the TD/1 CA's 6 rules), while the TU/2 CA learnt 243 and 248 rules for the RMS and SSIM index objective functions respectively.

8 Sensitivity Analysis

An aspect of CA often not tested is the effect of modifying any system parameters, and a sensitivity analysis carried out as in [35] would be useful. Since we are not considering different neighbourhoods, our only free parameter is the number of iterations. Our preliminary testing has shown that for certain types of image processing operations (such as edge and ridge detection) the CA converged to a steady state, and therefore the only requirement is to ensure that a sufficient number of iterations are performed. For the other image processing operations the CA did not necessarily completely converge within a reasonable number (e.g. a thousand) of iterations. Nevertheless, the rate of change in the results was often small enough to indicate that increasing the number of iterations would have an insignificant effect on the results.

This is demonstrated for computing the gray level convex hull (from figure 15) – see figure 16a.

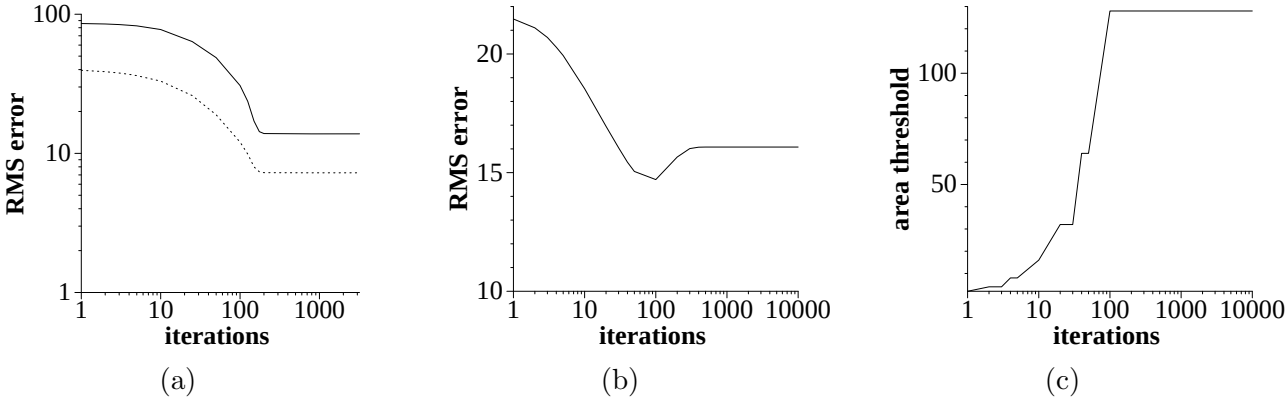


Figure 16: Effects of varying the number of iterations when running the CA. (a) gray level convex hull error, (b) gray level connected set closing error, (c) correlation between gray level connected set area threshold and number of iterations.

The graph shows the RMS error between the target image and the CA's output at each iteration (the dotted line was calculated over the complete image while the solid line was only calculated at non-zero, i.e. foreground, pixels). After about 200 iterations the differences between successive outputs are small enough to be visually indistinguishable.

In the case of the gray level connected set closing there is a closer correlation between the error and the number of iterations. The graph in figure 16b is based on applying the closing to the Miro image in figure 13. During the training phase 100 iterations were used, which can be seen to also produce the best result for the Miro test image. In fact, for this image processing task the number of iterations is related to the one free parameter of the operator, namely the area threshold. This was demonstrated by comparison of the CA results with those from the true closing algorithm [30]. For each CA result the area threshold which produced the minimum RMS error was determined (in Meijster and Wilkinson's algorithm the thresholds are all powers of two). When these area thresholds are plotted against the number of iterations (figure 16c) there is a clear correlation.

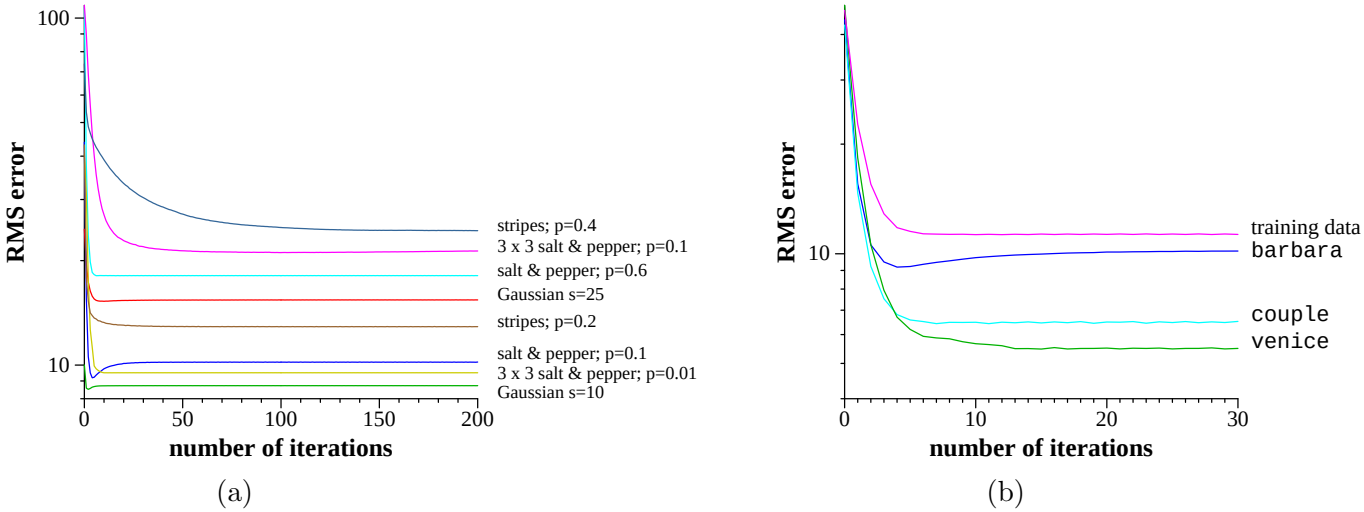


Figure 17: Effects of varying the number of iterations when running the CA for denoising. (a) results for the various noise models applied to `barbara`, (b) results for the salt and pepper $p = 0.1$ noise applied to various images.

Finally, figure 17a shows that for the denoising experiments, as applied to `barbara`, that the

CA converges within a few tens of iterations. However, while the errors for the methods generally monotonically decrease as the number of iterations increases, for the salt and pepper $p = 0.1$ noise a significantly better solution is achieved within just 4 iterations than the final converged solution. This can be explained by the unusually large proportion of thin striped patterns (with a width of only 2 pixels) in the `barbara` image. Since such patterns are not represented to the same degree in the training image the rules do not preserve the patterns, which are removed by the denoising. For the other images this is not an issue, as demonstrated in figure 17b. Note that if the CA has been terminated after 4 iterations its RMS value reduces from 10.21 to 9.19, and therefore outperforms the best result (previously achieved by the adaptive median) for denoising that image.

9 Conclusions and Future Work

This paper has described a new approach to building CA systems such that they can be applied to process gray level images while still enabling good rule sets to be learnt in a practical manner. The new 3 state representation provided the best results in many cases. However, the threshold decomposition methods were also effective, and were especially beneficial when extracting magnitudes of features. Comparing the two objective functions, although SSIM is considered to better capture perceptual similarity than RMSE it was found that SSIM was *not* generally found to provide an improvement, and for several applications RMS errors were distinctly better. Also, in our experiments it was found that TU/2 CA was more appropriate or performed better than TU/1 CA.

We have demonstrated the effectiveness of the CA on a variety of image processing tasks: denoising, computation of the gray level convex hull, connected set morphology, edge detection, and ridge and valley detection. However, the goal of the paper was not to replace standard algorithms for those tasks, but rather to show the capability of the CA to perform such tasks. Consequently, its greater usefulness will be if a method is required to carry out a specialised operation not covered by the standard algorithms. There is then a good chance that one of the CA schemes presented in this paper could be easily used to perform that operation. The benefit of the CA is that, unlike standard image processing methods, it does not need to be redesigned, but simply requires retraining using examples of the desired input and target output images to generate an appropriate rule set.

Many areas of future research remain. As discussed in this paper, while the TU CA representation makes the encoding of intensity information practical by reducing the number of cell states, it also discards a lot of important information. While several approaches to overcoming the loss of quantitative information were described in this paper (i.e. pre- and post-processing the image, or using the TD CA) an alternative would be to attempt to extend the CU representation. For instance, *totalistic* CA produce compact representations since the rules only depend on the total (or equivalently the mean) of the neighbouring cell values. If the mean, or some other function, of the neighbouring pixel intensity values were used then this would provide valuable quantitative information with which to augment the TU description. This representation would be similar in a sense to the LBP/C texture descriptor that uses LBP along with a local contrast measure [36]. However, the question still remains how this would be incorporated without vastly increasing the number of states and rule set search space.

Further areas for future investigation are consideration of alternative objective functions, in particular problem specific ones. An example would be in computing the convex hull, in which any non-convexity in the output should be explicitly recognised and penalised.

Many alternative optimisation techniques are also available, and evaluation of them would be useful. Also, combinations of them may be beneficial; an example would be to use stochastic techniques such as simulated annealing or genetic algorithms to initialise the deterministic SFFS.

Another interesting approach was taken by Vérel *et al.* [37] who used the commonalities of good solutions previously found (and for their well studied problem of the density classification task they had many previously published available) to define a subspace in the fitness landscape within which to focus search for improved solutions.

Acknowledgments

I would like to thank Kevin W. Bowyer for making the data set of ground truthed images with edge maps available, Michael Wilkinson for providing his code for connected set morphology, and Carsten Steger for providing his code for detecting curvilinear structures.

References

[1] I. Kusch and M. Markus. Mollusc shell pigmentation: cellular automaton simulations and evidence for undecidability. *J. Theor. Biol.*, 178:333–340, 1996.

[2] P.P. Chaudhuri, D.R. Chowdhury, S. Nandi, and S. Chattopadhyay. *Theory and Applications: Additive Cellular Automata.* IEEE Press, 1997.

[3] C. Georgoulas, L. Kotoulas, and G. Sirakoulis. Real-time disparity map computation module. *J. Microprocessors and Microsystems*, 32(3):159–170, 2008.

[4] M. Mitchell, P.T. Hraber, and J.P. Crutchfield. Evolving cellular automata to perform computation: Mechanisms and impedients. *Physica D*, 75:361–391, 1994.

[5] P. Sahota, M.F. Daemi, and D.G. Elliman. Training genetically evolving cellular automata for image processing. *Proc. Speech, Image Processing and Neural Networks*, 2:753–756, 1994.

[6] A. Adamatzky. Automatic programming of cellular automata: identification approach. *Kybernetes*, 26(2):126–135, 1997.

[7] M. Batouche, S. Meshoul, and A. Abbassene. On solving edge detection by emergence. In *Int. Conf. on Industrial, Engineering and Other Apps. of Applied Intelligent Systems*, volume LNAI 4031, pages 800–808, 2006.

[8] S. Slatnia, M. Batouche, and K.E. Melkemi. Evolutionary cellular automata based-approach for edge detection. In *Int. Workshop on Fuzzy Logic and Applications*, volume LNAI 4578, pages 404–411, 2007.

[9] A. Chavoya and Y. Duthen. Using a genetic algorithm to evolve cellular automata for 2D/3D computational development. In *Genetic and Evolutionary Comp. Conf.*, pages 231–232, 2006.

[10] R.V. Craiu and T.C.M. Lee. Pattern generation using likelihood inference for cellular automata. *IEEE Trans. on Image Processing*, 15(7):1718–1727, 2006.

[11] S.A. Billings and S.S. Mei. A new fast cellular automata orthogonal least-squares identification method. *Int. J. Systems Science*, 36(8):491–499, 2005.

[12] S.A. Billings and Y. Yang. Identification of the neighborhood and CA rules from spatio-temporal CA patterns. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, 33(2):332–339, 2003.

[13] L. Bull and A. Adamatzky. A learning classifier system approach to the identification of cellular automata. *J. Cellular Automata*, 2(1):21–38, 2007.

[14] G. Terrazas, P. Siepmann, G. Kendall, and N.O. Krasnogor. An evolutionary methodology for the automated design of cellular automaton-based complex systems. *J. Cellular Automata*, 2(1):77–102, 2007.

[15] B. Straatman, R. White, and G. Engelen. Towards an automatic calibration procedure for constrained cellular automata. *Computers, Environment and Urban Systems*, 28(1-2):149–170, 2004.

[16] K.I. Maeda and C. Sakama. Identifying cellular automata rules. *J. Cellular Automata*, 2(1):1–20, 2007.

[17] P.L. Rosin. Training cellular automata for image processing. *IEEE Trans. on Image Processing*, 15(7):2076–2087, 2006.

[18] F.S. Roberts and B. Tesman. *Applied Combinatorics*. Pearson/Prentice-Hall, 2005.

[19] L. Wang and D.C. He. Texture classification using texture spectrum. *Pattern Recognition*, 23:905–910, 1990.

[20] P. Pudil, J. Novovicova, and J.V. Kittler. Floating search methods in feature-selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994.

[21] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.

[22] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. on Image Processing*, 13(4):600–612, 2004.

[23] Z. Wang and A. C. Bovik. A universal image quality index. *IEEE Signal Processing Letters*, 9(3):81–84, 2002.

[24] L. Khriji and M. Gabbouj. Median-rational hybrid filters for image restoration. *Electronic Letters*, 34(10):977–979, 1998.

[25] H. Hwang and R.A. Haddad. Adaptive median filters: new algorithms and results. *IEEE Trans. on Image Processing*, 4(4):499–502, 1995.

[26] J. Romberg, H. Choi, and R. Baraniuk. Bayesian tree-structured image modeling using wavelet domain hidden markov models. *IEEE Trans. on Image Processing*, 10(7):1056–1068, 2001.

[27] M.S. Crouse, R.D. Nowak, and R.G. Baraniuk. Wavelet-based statistical signal-processing using hidden markov-models. *IEEE Trans. on Signal Processing*, 46(4):886–902, 1998.

[28] G. Gilboa, N.A. Sochen, and Y.Y. Zeevi. Image enhancement and denoising by complex diffusion processes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(8):1020–1036, 2004.

[29] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[30] A. Meijster and M. Wilkinson. A comparison of algorithms for connected set openings and closings. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(4):484–494, 2002.

[31] W.H. Tsai. Moment-preserving thresholding. *CVGIP*, 29:377–393, 1985.

[32] C. Steger. An unbiased detector of curvilinear structures. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(2):113–125, 1998.

[33] P. Soille. Grey scale convex hulls: definition, implementation, and application. In H. Heijmans and J. Roerdink, editors, *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 12, pages 83–90. Kluwer Academic Publishers, 1998.

[34] I. Nyström, G. Borgefors, and G. Sanniti di Baja. 2D grey-level convex hull computation: A discrete 3D approach. In *Proc. Scand. Conf. Image Anal.*, volume 2749 of *LNCS*, pages 763–770. Springer, 2003.

[35] V. Kocabas and S. Dragicevic. Assessing cellular automata model behaviour using a sensitivity analysis approach. *Computers, Environment and Urban Systems*, 30(6):921–953, 2006.

[36] T. Ojala, M. Pietikainen, and D. Harwood. A comparative study of texture measures with classification based on feature distributions. *Pattern Recognition*, 29(1):51–59, 1996.

[37] S. Vérel, P. Collard, M. Tomassini, and L. Vanneschi. Fitness landscape of the cellular automata majority problem: View from the 'Olympus'. *Theor. Comput. Sci*, 378(1):54–77, 2007.