Control and Cybernetics

vol. **25** (1996) No. 1

Simulated annealing and genetic algorithms for shape detection[1]

by

Miriam Di Ianni*, Ralf Diekmann, Reinhard Lüling**, Jürgen Schulze** and Stefan Tschöke****

*Dip. di Informatica e Sistemistica,
University di Roma 'La Sapienza', Italy
e-mail: diianni@dsi.uniroma1.it

**Department of Mathematics and Computer Science,
University of Paderborn, Germany
e-mail: {diek, rl, schlunz, sts}@uni-paderborn.de

Abstract: The paper presents three heuristical methods for the solution of the shape detection problem. This problem arises in a large number of applications and is therefore of large interest to develop effective methods for its solution. We formulate the shape detection problem as a combinatorial optimization problem and use methods based on simulated annealing and genetic algorithms for its solution. To speed up the detection process we develop a parallelization strategy for both methods.

As a third method we present a constructive heuristic based on filtering noise. Experimental results show that the simulated annealing and the constructive method give very good results, computed in short time.

1. Introduction

Digital images are used in several different applications. At first, two of the main users of digital images were medical sciences, using them e.g. to detect chromosome abnormalities, and remote sensing, where satellites daily produced new images of the earth. At present, digital images are also used in many other fields, such as other medical applications (tomography, X-rays), biology, some industrial applications. All these application fields stressed the relevance of finding good solutions to the problem of automatic image analysis.

1 This work was partly supported by the EC Esprit Basic Research Action Nr. 7141 (ALCOM II) and the EC Human Capital and Mobility Project "Efficient Use of Parallel Computers: Architecture, Mapping and Communication"

The tools which are used to deal with digitized images are generally called *image processing* procedures. A common problem in dealing with images is that they are often noisy. Thus, image processing procedures are required to remove the noise, restoring the "original" image. Furthermore, in some cases, they are also required to automatically analyze the image, distinguishing in it the objects from the background and recognizing particular objects contained in it. A lot of literature has been devoted to image processing problems, both with respect to the noise removal problem, which is strictly connected with the problem of separating objects points from background points, generally solved by *image thresholding* algorithms, and with respect to the recognition of particular objects (cf. e.g. Angell and Barber, 1977; Ballard and Sabbah, 1983; Chen and Lee, 1990; Hung, 1993; He and Kundu, 1991; Jakubowski, 1990; Lee, Lu and Tsai, 1990; Nakagawa and Rosenfeld, 1979; Rosenfeld, 1990, Slgale and Dixon, 1984; Sinha and Giardina, 1990; Wang, Hanson and Riseman, 1988; Wojcik, 1983).

In this paper we consider the problem of recognizing simple geometric shapes in a picture corrupted by noise. The algorithmic techniques we use for its solution are simulated annealing, genetic algorithms and a constructive method based on noise filtering.

Simulated annealing is a powerful stochastic technique for solving combinatorial optimization problems. One of the main drawbacks of simulated annealing is its high computational requirements. Because of this, a number of parallel implementations have been proposed (cf. Aarts, de Bont, Hbers and van Laarhoven, 1986; Banerjee, Jones and Sargent, 1990; Casotto, Romeo and Sangiovanni-Vincentelli, 1987; Diekmann, Lüling and Simon, 1993; Kravitz and Rutenbar, 1987; Poussel-Ragot and Dreyfus, 1990; Witte, Chamberlain and Franklin, 1990). In particular, in Diekmann, Lüling and Simon (1993) some problem independent parallel implementations of simulated annealing have been described.

Simulated annealing has been proposed to solve image recognition problems (cf. Bongiovanni, Crescenzi and Guerra, 1992; Carnevali, Coletti and Patternello, 1985; Tan, Gelfand and Delp, 1989). In particular, in Bongiovanni, Crescenzi and Guerra (1992) a parallel implementation of simulated annealing for the shape detection problem has been proposed. In this paper we present the results obtained using the farming implementation of simulated annealing as it was proposed in Diekmann, Lüling and Simon (1993) for other applications.

In Section 2. of this paper, the shape detection problem is formally defined and its representation in terms of a combinatorial optimization problem is described. In Section 3. the general simulated annealing algorithm is described together with some of the parallel implementations proposed for it. In Section 4. we describe a genetic algorithm for the shape detection problem. This algorithm is inherently parallel. In Section 5. we present a constructive heuristic for the shape detection problem which is based on a noise filter. Performance measurements presented in Section 6. for the different algorithms finish the paper.

2. The shape detection problem

The shape detection problem (from now on, in short, SDP) can be defined as follows: given a picture representing a set of (eventually overlapping) geometric shapes, i.e., parallelograms, ellipses, triangles and so on, compute the list L of the shapes included in the picture. In general, the picture is corrupted by noise and the goal is, according to what discussed in the introduction, to distinguish among "noise" pixels and "regular" ones and to recognize the shapes contained in the image.

The picture is represented as a binary matrix A such that

$$A(i,j) = \begin{cases} 0 & \text{if pixel (i,j) is white} \\ 1 & \text{if pixel (i,j) is black.} \end{cases}$$

where $1 \leq i, j \leq n$ and, in absence of noise, black pixels belong to a shape while white pixels represent background.

We assume that the shapes to be detected are represented in terms of a set of parameters. For instance, a parallelogram is represented by three of its vertices, an ellipse by the length of its two axes, and so on. The problem consists then in finding a set of parameters which describe the picture.

The SDP can be formulated in term of an optimization problem (Bongiovanni, Crescenzi and Guerra, 1992). A cost function is defined as the difference between the observed image and the image obtained by some estimated set of parameters

$$C = \sum_{i=1}^{n} \sum_{j=1}^{n} |A(i,j) - A'(i,j)| + kN$$

where $A'(i,j) = 1$ if some shape included in some guessed L contains pixel (i,j). The goal is finding a list L_0 such that C is minimum.

The term kN controls the number of shapes, N being the number of shapes included in L and k a weighting factor. This last term is used in order to avoid to consider as a shape a small set of pixels. It is particularly useful in presence of noise. Observe that, if very noisy pictures are considered, then it is likely that (relatively) large sets of close pixels are not to be considered as shapes but they are due to noise. Then, it is worth to assign k an higher value than in the case of a low noise picture.

To simplify the following algorithms and measurements we restrict the SDP from its general form (as described above) to the detection of images build of overlapping rectangular objects.

3. A simulated annealing algorithm for the SDP

Simulated annealing is a powerful stochastic technique for solving (hard) combinatorial optimization problems.

```
begin
    T := T_0;
    choose an initial configuration C_0;
    C := C_0;
    while T > T_1 do
    begin
        while equilibrium has not been reached do
        begin
            choose a neighbor configuration of C;
            Δ := cost(C') − cost(C);
            if Δ ≤ 0 then C := C'
            else C := C' with probability e^{−Δ/T};
        end;
        T := cooling(T);
    end;
end.
```

Figure 1. Structure of the simulated annealing algorithm

It differs from other techniques based on iterative improvements of the cost function in that it allows occasional worsenings in the cost function thus avoiding to get stuck into local minima.

A simulated annealing algorithm (SAA) starts from an initial configuration and tries to find an optimal configuration (with respect to the given cost function) by generating a sequence of small changes (*perturbations*) to the current solution. The changes that improve the value of the cost function are always accepted, like in an iterative improvement method, while the changes that worsen the value of the cost function are accepted with a probability that depends on the value of a control parameter called *temperature*. At the beginning, such a parameter is assigned with a high value and it is decreased during the evolution of the process. The algorithm ends when the temperature has reached some minimum value. The general structure of a SAA is presented in Figure 1.

3.1. Configuration space and cooling schedule

Simulated annealing has been proposed to solve image recognition problems (cf. e.g. Carnevali, Coletti and Patternello, 1985; Bongiovanni, Crescenzi and Guerra, 1992). As far as the shape detection problem is concerned, a configuration of the simulated annealing algorithm is defined to consist of a set of shapes. A neighbor configuration is obtained by performing a change to the current configuration which involves one single shape based on a random choice. Thus, the simulated annealing algorithm starts from any initial configuration and tries to improve the value of *cost* by attempting a change at each step. In particular, the following changes are considered:

GENERATE: Creates a new element at random position and of random size.
DELETE: Removes an existing element.
STRETCH/SHRINK: Stretching or shrinking of an existing element by a small number of pixels.
SPLIT: Divides an existing shape into two by removing one line of pixels inside of it. (Observe that this last operation is not defined for arbitrary shapes since, for instance, removing a line of pixels in an ellipse does not bring to a pair of ellipses.)

At each iteration of the simulated annealing algorithm one of the operations is chosen at random and applied to the current configuration. Thus there is no guidance in the search process.

As a cooling schedule we used the self adapting schedule presented in Huang, Romero, Sangiovanni-Vincentelli (1986). The advantage of using such a schedule is that the temperature decrease is guided by the search process itself, thus the parametrization of the algorithm is minimized. Self adaptive schedules decrease the temperature in dependence of the average cost function value and its variance.

3.2. Parallelization of simulated annealing algorithms

One of the major drawbacks of simulated annealing is its high computational requirements. One way to overcome this problem is using parallelism. There are a number of ways parallelism can be applied to simulated annealing (cf. e.g. Aarts, de Bont, Hbers and van Laarhoven, 1986, Banerjee, Jones and Sargent, 1990, Casotto, Romeo and Sangiovanni-Vincentelli, 1987, Diekmann, Lueling and Simon, 1993, Kravitz and Rutenbar, 1987, Poussel-Ragot and Dreyfus, 1990, Witte, Chamberlain and Franklin, 1990).

In general one can distinguish two principles for parallelizing simulated annealing: The first is based on partitioning the problem describing data and performing simulated annealing on each data subset, whereas the second strategy parallelizes the SAA itself.

The first principle was used in an implementation for the SDP presented in Bongiovanni, Crescenzi and Guerra (1992). There are two reasons why we chose the second method for our parallelization: One reason is that the parallel efficiency of the first principle is in general limited by the extend to which the problem can be split up into smaller subproblems, which itself can be treated by simulated annealing. The other and more striking reason is that the dependency relations between different subproblems in general lead to a worse solution quality for the parallel algorithm, compared to the sequential implementation. Therefore we used a method presented in Diekmann, Lueling and Simon (1993) which in general provides the solution quality of the sequential algorithm also for the parallel implementation and achieves reasonable speedup at the same time.

```
begin
    S = set of feasible solutions
    for i = 1 to L
        begin
            S' = MUTATE(S)
            S'' = RECOMBINE(S')
            S = SELECT(S'')
        end;
end
```

Figure 2. Structure of genetic algorithm for the shape detection problem

For the shape detection problem, the best performance can be achieved using a farming technique. Using this method one processor, the farmer, generates neighboring configurations which are evaluated by the workers. The local worker processor also decides about the acceptance of the current local solution. If the solution is accepted, it is sent back to the central farmer.

We implemented this method on a Transputer network. Performance measurements are presented in Section 6.

4. A genetic algorithm for the SDP

Genetic algorithms have shown to be one of the most promising approaches to apply concepts from nature to solve combinatorial optimization problems. Other concepts adopted from nature are for example neural networks which have also shown some performance when used for solving combinatorial optimization problems but are usually outperformed by classical methods. One of the main advantages of genetic algorithms are their general applicability. Thus, one has to invest very little knowledge about the structure of the optimization problem to get some solution. To find real good solutions one has to study the problem structure in more detail. This effect is comparable to the use of simulated annealing. Another advantage of genetic algorithms, which is especially important in our context, is the fact that these types of algorithms contain some kind of inherent parallelism, which makes it very easy to map them onto parallel/distributed computing systems.

A genetic algorithm basically consists of three different genetic operators: mutation, selection and recombination. These operators are applied iteratively to a set of feasible solutions, the so called population. While repeating this process, the solution quality of the population is improved until either all candidates stagnate at a local optimum or a fixed number of populations have been evaluated.

Figure 2 describes the overall structure of a genetic algorithms as it is used

```
begin
    B(i,j) = 0, ∀i,j ∈ {1,...,n}
    while moving a frame of size a × b in snake-like order over A do
    begin
        if the frame of size a × b in A contains at least a · b · r black pixels
            and this frame is not completely colored with black pixels in B
                repeat
                    add black rows and columns to the frame if the
                    ratio of black pixels in the enlarged frame is still
                    larger than r.
                until the frame has not been enlarged
        copy the resulting frame to B and color it black
    end
end
```

Figure 3. Computing feasible solutions

for our experiments. In the following we describe how the different procedures are constructed for the special case of the shape detection problem.

4.1. Computing the initial population

As genetic algorithms work on a number of feasible solutions, the so called population, one has to generate a number of solutions in the initial phase of the algorithm. We do this by using a simple heuristic for the shape detection problem.

The heuristic is parameterized by the size of a subpicture a and b and a given number r, $0 \leq r \leq 1$ and generates a feasible solution described by a matrix $B(i,j)$, $1 \leq i,j \leq n$. It starts moving a frame of size $a \times b$ in snake-like order over the noisy image A. Every time this frame covers a set of pixels which are colored by more than r percent with black, the algorithm tries to adjust a rectangle on this location. The rectangle is extended until the ratio of black pixels it covers decreases below r percent. The detailed algorithm is listed in Figure 3.

To construct the initial population, we run this algorithm for a set of parameters a, b and r. The overall population contains m feasible solutions at every time.

4.2. Selection operator

The selection operator usually takes the quality of configurations (the so called fitness) into account. Solutions with high fitness are selected more frequently

begin
sum = $C(s_1) + C(s_2) + \ldots + C(s_m)$
for i = 1 **to** m **do**
begin
choose solution $s_i \in \{s_1, \ldots, s_m\}$ with probability $C(s_i)/sum$
$S' = S' \cup s_i$
end
end

Figure 4. The selection operator

for recombination. Suppose the overall population is represented by a list of solutions $S = s_1, \ldots, s_m$ with associated cost function values $C(s_1), \ldots, C(s_m)$.

The selection algorithm constructs a new population S' by iteratively choosing one item of the old population according to its fitness. This solution is added to the new population. Repeating this m times, the new population contains a solution with good solution quality more often. Therefore, this element will be used for recombination with higher probability. The detailed algorithm is listed in Figure 4.

4.3. Recombination operator

The recombination operator iteratively chooses two elements from the actual population and builds a new element out of these two solutions. This process is repeated for a constant number of iterations.

To construct a feasible solution s_k out of two random chosen solutions s_i and s_j, s_i is first copied to s_k. Then a random shape element from both solutions s_i and s_j is chosen. If there is an overlap between this two shapes the minimal shape containing both is inserted into the s_k. If no overlap exists between the two selected elements, both are placed in the new solution s_k.

The two parents s_i and s_j are chosen according to their cost function value. Here we use the same method as in the selection operation. The newly generated element s_k is inserted into the population and replaces either s_i or s_j, depending on which of both has the smaller fitness. Thus, the overall number of elements in the population stays constant.

4.4. Mutation operator

The mutation operator randomly selects elements of the actual population and applies local modifications. The mutation operations are comparable to those already listed for the simulated annealing procedure (cf. Sec. 3.1.). To get a better convergence behavior we used slightly more powerful operations:

GENERATE: Creates a new element at random position and of random size.
DELETE: Removes an existing element.
UNION: Selects two elements at random and substitutes them by the smallest shape that contains both selected elements.
MOVE: Selects a random element and moves it into a random chosen destination. The maximal moving distance is equivalent to half of the maximal side length of the shape. After moving the element it can be stretched or shrunk as described for the simulated annealing algorithm.

The operations are applied to a number of randomly chosen elements. The actual number is a parameter of the genetic algorithm and depends on the number m describing the population size. The operator which is applied to the elements is also chosen at random in each case.

4.5. Parallelization of genetic algorithms

A genetic algorithm is inherently parallel since the overall population can be distributed onto a processor network in a very natural way. Every processor can run its local genetic algorithm on its own subpopulation. Thus, this parallelization follows Darwin's island approach. On each island (processor) a local population is resident, which evolves independently of the populations on the other islands. An exchange of individuals only happens in irregular intervals, when individuals are island-overlappingly selected for recombination.

Strategies of this type have been used for genetic algorithms (see e.g. Kröger, Schwenderling, Vornberger, 1991) and it has been observed that the solution quality is in general comparable to that of the sequential algorithms while achieving significant speedup.

5. A constructive heuristic

Additionally to general methods like simulated annealing and genetic algorithm we also developed a constructive heuristic tailored to the shape detection problem. The heuristic consists of two phases as there are an image preprocessing and a heuristic shape detection.

5.1. Image preprocessing

The image preprocessing phase of our heuristic tries to compute the best possible approximation to the supposed original image without any shape detection at all. It only consists of changing the pixel information of a given image. On a copy of a given noisy image some functions to swap pixels are applied, completely eliminating noise up to 40% and amplifying rectangular contours. We reached our best results with four functions applied in the order described below.

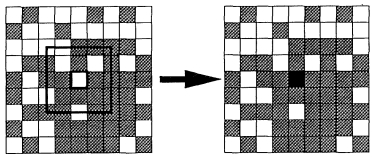

Figure 5. Filtering noise

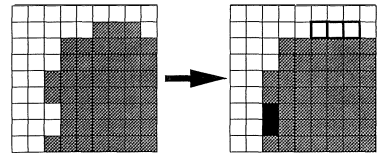

Figure 6. Smoothing edges

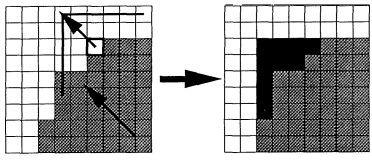

Figure 7. Amplifying corners

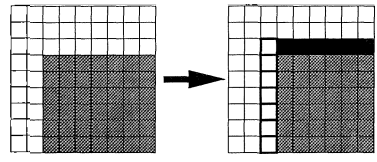

Figure 8. Error correction

5.1.1. Filtering noise

The noise filtering function computes for every pixel the number of black and white pixels in a surrounding square of given radius. If less than 45% of the surrounding pixels are white then the center pixel is turned to black. If less then 45% of the surrounding pixels are black the center pixel is turned to white. The function is applied iteratively with decreasing radius from 7 to 1 (Figure 5 shows an example with radius 2). In the resulting image noise of less or equal to 40% is completely eliminated with high probability, but edges of the shape are not straight and all corners of the shape are rounded with a radius of at least 5 pixels.

5.1.2. Smoothing vertical and horizontal edges

In the resulting image small hills and small sinks of up to 5 pixels are removed from all vertical and horizontal shape/background edges (see Figure 6).

5.1.3. Amplifying rectangular corners

In this function for all eight types of corners (north-west outer corner as in figure 7, north-west inner corner, NE outer, NE inner, SW-o, SW-i, SE-o, SE-i) every pixel of the image is scanned with two sweep lines in decreasing distance (3,2,1) from the examined pixel. If at least one sweep line is not crossing a shape background edge a white pixel of a round corner can be set to black when scanning shape corners and a black can be set to white if scanning background corners.

5.1.4. Error correction of vertical and horizontal edges

Errors in the position of shape/background edges are corrected by comparing processed image with the noisy input image. If a black line of an edge of the processed image has less than 45% black pixels in the noisy input image it is turned to white and if a white line has less than 45% white pixels is is turned to black.

After applying all four functions the pixel difference of the preprocessed image to the original shape is in average 0 for noise <=25%, about 1% for noise <=35% and about 5% for noise for <=42%. Images with more than 46% noise cannot be handled with this filter.

5.2. Rectangle detection

After preprocessing the input image we still have no information about the shape. But now the rectangle detection is very simple, because the approximation to the original image is very good in most cases. We use a heuristic, which is optimal if the pixel difference to the original shape is 0. Take a black pixel of the shape and expand the pixel to all direction as far as the growing rectangle is still completely part of the shape, i.e. consists of black pixels. If there is a black pixel left which is not covered by previously constructed rectangles then do the expansion once more. If not, stop with the minimal number of image covering, overlapping rectangles.

6. Measurement results

6.1. Benchmark suite

To test the described algorithms we used four images of size 100×100 consisting of a number of black rectangles placed on a plain white ground. Figure 9 shows the four images. Each of them is used with 10, 20 and 35 percent of uniform noise. Figure 10 shows the first test instance with increasing noise.

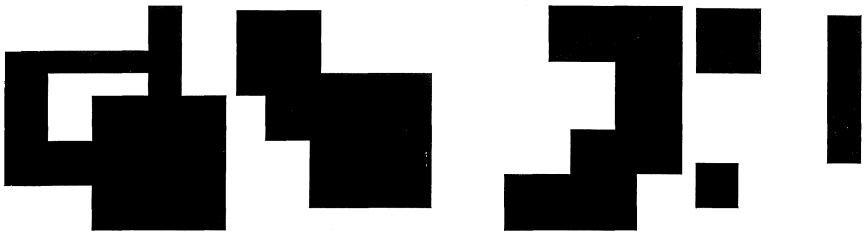

Figure 9. The four test instances

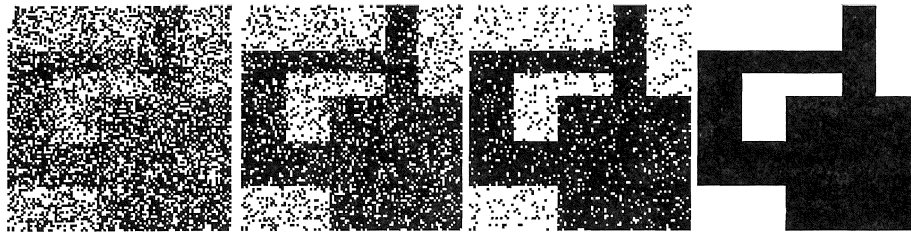

Figure 10. The first instance with 0%, 10%, 20% and 35% noise

6.2. Results: Detected shapes

Table 1 shows results of the three algorithms on each of the four test problems. We define $\Delta = \sum |A_o(i,j) - A'(i,j)|$ as the difference between the detected image and the original one without noise. R indicates the number of detected rectangles. SA are the results of the SAA, GA those of the genetic algorithm. Additionally we give the best solution found for the start population of the GA. This solution is calculated by the simple heuristic (SH) described in Section 4.1. The column named CH gives the results of the constructive heuristic from Section 5.

Im. (R)	N(%)	SA		SH		GA		CH	
		Δ	R	Δ	R	Δ	R	Δ	R
1	10	0	5	126	7	42	5	0	5
(5)	20	6	7	430	5	160	5	0	5
	35	80	5	3736	542	683	96	120	6
2	10	0	3	87	10	52	8	0	3
(3)	20	12	5	249	25	151	15	0	3
	35	81	4	3174	458	930	107	0	3
3	10	0	4	95	8	63	12	0	4
(4)	20	4	5	226	27	159	14	0	4
	35	64	4	3151	434	764	41	60	4
4	10	0	3	46	6	31	5	0	3
(3)	20	11	4	595	196	270	17	0	3
	35	86	4	1363	144	922	95	39	3

Table 1. Comparison of the results from the three approaches to the SDP.

The results of SAA and GA are the best results out of several runs with different random sequences. Our implementation of SA turns out to be very stable. The mean values of 10 runs do not differ significantly from the best results. The cooling schedule parameters correspond to the original setting of Huang et. al. (1986) as they are also reported in Diekmann, Lüling and Simon

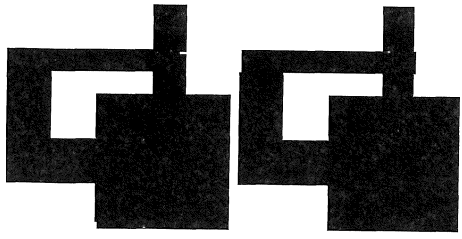

Figure 11. Solution of SAA, image 1, 20% and 35% noise

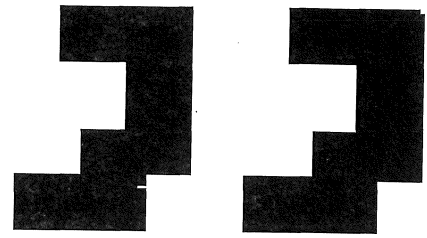

Figure 12. Solution of SAA, image 3, 20% and 35% noise

Figure 13. Initial and GA-solution of image 1 with 35% noise

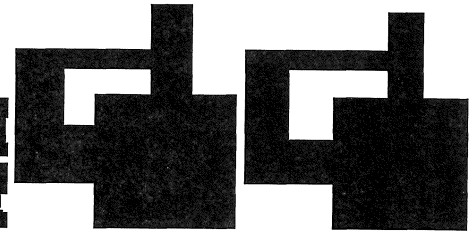

Figure 14. Initial and GA-solution of image 1 with 20% noise

(1993). The frozen criterium is fullfilled if either no cost variation occurs in 4 successive chains or if $\frac{\sigma^2(T_f)}{T_f C(T_f) - C_0} < \epsilon$ where $\epsilon = 0.01$ and $C(T_f)$ is the average of cost variations at temperature T_f. The equilibrium detection uses Huang's criterion, too, with a minimum of $n/2$ accepted and a maximum of $n^2/40$ generated perturbations.

The GA is run with a population size between 50 and 150 and between 10 and 30 generations. The mutation rate is varied between 10 and 50% and the recombination rate between 30 and 80%. After each change of an arbitrary

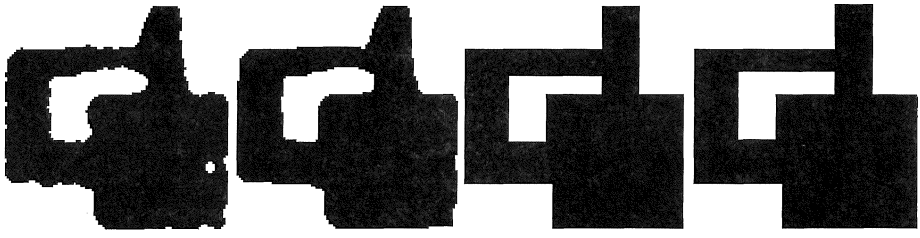

Figure 15. Solution of the constructive heuristic, image 1 (35% noise) after phase 1 (noise filtering, $\Delta = 461$), 2 (smoothing edges, $\Delta = 443$), 3 (amplifying corners, $\Delta = 362$) and 4 (error correction, $\Delta = 120$)

rectange, it is adusted by a value of up to ± 5 pixels in each direction in order to locally maximize the cost function. The results are the best found within 50 runs with different parameters.

Some of the results are shown in Figures 11 to 15. Figure 11 and 12 show the detected images of SAA on the test instance 1 and 3 with 20% and 35% noise. It can be seen that the results correspond directly to the original pictures. The small differences do not affect the overall impression.

Figures 13 and 14 show results of our genetic algorithm on image 1 with 20% and 35% noise compared to the best initial solutions found by the simple heuristic SH. The pictures show that the GA finds substantial improvements to the initial solution although the results are not comparable to those of SA or the constructive heuristic CH. If only noise reduction is considered, the overall impression is reasonable, i.e. the overall shape (not considering details too much) corresponds to the original picture.

Figure 15 shows image 1 with 35% noise after the different steps of the constructive heuristic. The noise reduction step is image independent and can therefore be used as preprocessing step to the other heuristics, too. The second, third and forth step work, as they are, only if rectangular corners and edges parallel to the coordinate axes are considered. These phases can, of course, be adapted to other types of images. The results of the constructive heuristic are very impressive. As it is tailored to images consisting of overlapping rectangles that were used as benchmark suite, it is able to find in almost every case the original image (cf. Table 1).

6.3. Results: efficiency

The running times of the heuristic optimization algorithms depend directly on the choice of parameters. The efficient self adapting cooling schedule of the SAA enables a sequential running time of about 30 seconds on a SUN SS10, nearly independent of the considered image. This running time is comparable to that of the constructive heuristic. The genetic algorithm takes about 10 times more running time.

The simple parallelization strategy which was chosen to speed up the SAA offers the potential to decrease the running time by a factor of 2 on 4 processors. The use of larger parallel systems makes no sense with this farming approach, as the time to calculated the cost difference within the simulated annealing loop takes only linear time. This observation corresponds to the results of Diekmann, Lüling and Simon (1993).

7. Conclusions

In this paper we formulated the shape detection problem as a combinatorial optimization problem. We showed how to apply simulated annealing and genetic algorithms to the problem and designed a specialized construction heuristic,

tailored to images consisting of overlapping rectangles which are used in the benchmark suite. The construction heuristic is mainly based on noise filtering, edge smoothing and corner correction and can detect shapes built up by rectangles very efficiently.

The use of simulated annealing with an efficient self adapting cooling schedule delivers results that are comparable to those of the specialized heuristic. Images with up to 35% of noise are detected by SA with an error of less than 1% in almost all cases within 30 seconds of sequential time on a standard workstation. We also showed how to speed up the simulated annealing algorithm by a factor of 2 without affecting the convergence behavior, using a simple farming approach.

Our implementation of a genetic algorithm for the shape detection problem delivers results which are, considered on there own, reasonable. However, compared to those of SA they are not very convincing. This may result from our difficulty to define good crossover operators and to find the right setting to the large number of parameters of this method. This problem does not arise with SA because of the possibility to use a self adapting cooling schedule.

In general we conclude that it is much easier to apply simulated annealing to the shape detection problem than to use genetic algorithms, at least if no kind of "automatic parameter setting" is given for the GA.

Our future work will focus on the generalization of the SA implementation to the detection of arbitrary objects.

Acknowledgements

We thank Derk Meyer and Robert Preis (Paderborn) for the implementation of the genetic algorithm. Thanks go also to Gianluca Trombetta (Rome) for the implementation of the simulated annealing algorithm.

References

AARTS, E.H.L., DE BONT, F.M.J., HABERS, E.H.A. and VAN LAARHOVEN, P.J.M. (1986) Parallel implementations of the statistical cooling algorithm. *Integration: the VLSI Journal*, **4**, 209-238.

ANGELL, P.H., BARBER, E. (1977) An algorithm for fitting circles and ellipses to megalithic stone rings. *Science and Archaeology*, **20**, 11-16.

BALLARD, D., SABBAH, E. (1981) On Shapes. *7th Int. Joint Conf. on Artificial Intelligence*, Vancouver, 607-612.

BALLARD D., SABBAH E. (1983) Viewer independent shape recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **PAMI-5**, 653-660.

BANERJEE, P., JONES, M.H. and SARGENT, J.S. (1990) Parallel simulated annealing algorithms for cell placement on hypercube multiprocessor.

IEEE Trans. on Parallel and Distributed Systems, **1**, 91-106.

BONGIOVANNI, G., CRESCENZI, P. and GUERRA, C. (1995) Parallel simulated annealing for shape detection. *Computer Vision and Image Understanding*, **61**, 1, 60-69.

CARNEVALI, P., COLETTI, L. and PATTERNELLO, S. (1985) Image processing by simulated annealing. *IBM J. Res. Develop.*

CASOTTO, A., ROMEO, R. and SANGIOVANNI-VINCENTELLI, A. (1987) A parallel simulated annealing for the placement of macro-cells. *IEEE Trans. on Computer-Aided Design*, **CAD-6**, 838-847.

CHEN, L.H. and LEE, K.L. (1990) A new method for circular object detection and location. *Patt. Recogn. Lett.*, **11**, 691-697.

DIEKMANN, R., LÜLING, R. and SIMON, J. (1993) Problem independent distributed simulated annealing and its applications, *Applied Simulated Annealing*. R.V.V. Vidal (ed.), Springer LNEMS 396, 17-44.

FREEMAN, R. (1978) Shape description via the use of critical points. *Pattern Recognition*, **10**, 159-166.

HE, Y. and KUNDU, A. (1991) 2-D shape classification using hidden Markov model. *IEEE Trans. on Patt. Analysis and Machine Intelligence*, **13**, 1172-1184.

HUNG, D.C.D. (1993) Non-conventional algorithm for representing and recognizing complicated two-dimensional objects. *Patt. Recogn.*, **26**, 495-504.

HUANG, M.D., ROMEO, F., SANGIOVANNI-VINCENTELLI, A. (1986) An efficient general cooling schedule for simulated annealing. *IEEE International Conference on Computer Aided Design*, 381-384

JAKUBOWSKI, R. (1990) Decomposition of complex shapes for their structural recognition. *Information Sciences*, **50**, 35-71.

KRÖGER, B., SCHWENDERLING, P., VORNBERGER, O. (1991) Genetic Packing of Rectangles on Transputers. In: *Transputing 91*, P. Welsh, D. Stiles, T.L. Knuii and A. Bakkers (ed), IOS Press Amsterdam, pp. 593-608.

KRAVITZ, S.A. and RUTENBAR, R.A. (1987) Placement by simulated annealing on a multiprocessor. *IEEE Trans. on Computer-Aided Design*, **CAD-6**, 534-549.

VAN LAARHOVEN, P.J.M. and AARTS, E.H. (1988) *Simulated annealing: theory and applications*. D. Reidel Publishing Company.

LANGRIDGE, B., (1972) On the computation of shape. In: *Frontiers in Pattern Recognition*, Academic Press.

LEE, R., LU, P.C. and TSAI, W.H. (1990) Moment preserving detection of elliptical shapes in gray-scale images. *Patt. Recogn. Letters*, **11**, 405-414.

MURRAY, D.W., KASHKO, A. and BUXTON, H. (1986) A parallel approach to the picture restoration algorithm of Geman and Geman on a SIMD machine. *Image and Vision Computing*, **4**, 3, 133-142.

NAKAGAWA, Y., ROSENFELD, A. (1979) A note on polygonal and elliptical approximation of mechanical parts. *Pattern Recognition*, **11**, 133-142.

POUSSEL-RAGOT, P. and DREYFUS, G. (1990) A problem independent parallel implementation of simulated annealing: models and experiments. *IEEE Trans. on Computer-Aided Design*, **CAD-9**, 827-835.

ROSENFELD, A. (1990) Fuzzy Rectangles. *Patt. Recogn. Lett.*, **11**, 677-679.

SINHA, D. and GIARDINA, C.R. (1990) Discrete black and white object recognition via morphological functions. *IEEE Trans. on Patt. Analysis and Machine Intelligence*, **12**, 275-293.

SLGALE, U., DIXON, N.A. (1984) Freedom descriptions: A way to find figures that approximate given points. *Pattern Recognition*, **17**, 631-636.

SOLTAN, V. and GORPINEVICH, A. (1993) Minimum dissection of a rectangular polygon with arbitrary holes into rectangles. *Discrete and Computational Geometry*, **9**, 57-79.

TAN, H.L., GELFAND, S.B. and DELP, E.J. (1989) A cost minimization approach to edge detection using simulated annealing. In: *Proc. of the 1989 Conference on Computer Vision and Pattern Recognition*, 89-91.

WANG, C.Z.., HANSON, A., RISEMAN, P.W. (1988) Fast Extraction of Ellipses. In: *Proc. of the 9th Int. Conf. on Pattern Recognition*, Rome, 508-510.

WITTE, E.E., CHAMBERLAIN, R.D. and FRANKLIN, M.A. (1990) Parallel simulated annealing using speculative computation. In: *Int. Conf. on Parallel Processing*, 286-290.

WEN, W. and LOZZI, A. (1992) Recognition and inspection of two-dimensional industrial parts using subpolygons. *Pattern Recognition*, **25**, 1427-1434.

WOJCIK, S. (1983) Method of contour recognition. *Digital Systems for Industrial Automation*, **2**, 63-83.