Journal of Electronic Imaging 6(4), 504–516 (October 1997).

Parallel genetic algorithms in the optimization of morphological filters: a general design tool

Peter Kraft
Neal R. Harvey
Stephen Marshall
University of Strathclyde
Signal Processing Division
Department of Electronic and Electrical Engineering
204 George Street
Glasgow, G1 1XW, United Kingdom

Abstract. *Mathematical morphology has produced an important class of nonlinear filters. Unfortunately, design methods existing for these types of filter tend to be computationally intractable or require some expert knowledge of mathematical morphology. Genetic algorithms (GAs) provide useful tools for optimization problems which are made difficult by substantial complexity and uncertainty. Although genetic algorithms are easy to understand and simple to implement in comparison with deterministic design methods, they tend to require long computation times. But the structure of a genetic algorithm lends itself well to parallel implementation and, by parallelization of the GA, major improvements in computation time can be achieved. A method of morphological filter design using GAs is described, together with an efficient parallelization implementation, which allows the use of massively parallel computers or inhomogeneous clusters of workstations.* © *1997 SPIE and IS&T.* [S1017-9909(97)01104-5]

1 Introduction

Nonlinear techniques are becoming an increasingly popular area of image and signal processing. Problems inherent in linear techniques cause poor and unacceptable performance in a multitude of applications. Among the many nonlinear techniques available, mathematical morphology[1] has emerged as one of the most important and useful, providing an important class of image and signal processing operators. These morphological operators are simple and the filters and techniques which have arisen out of these operators provide powerful tools, giving excellent results in a wide range of areas such as object recognition, edge detection, and noise reduction. Unfortunately, optimization methods existing for morphological filters and operators tend to have such a high computational complexity that they are unrealizable on current computer hardware.[2–5]

Genetic algorithms[6] provide powerful search and optimization methods, which are based on the evolutionary processes found in nature. They provide tools for investigating problems made difficult by substantial complexity and uncertainty. A major advantage of genetic algorithms is that the underlying mechanisms are such that they lend themselves well to parallelization, allowing a significant reduction in computation time.[7,8]

Previous work on the application of genetic algorithms to nonlinear filters includes Ref. 9, which utilized genetic algorithms in shape analysis; Ref. 10, which used genetic algorithms in the optimization of binary Matheron-representation morphological filters; Ref. 11, which used genetic algorithms in the configuration of stack filters; and Ref. 12, which utilized genetic algorithms in the optimization of the structuring systems of function-set processing soft morphological filters. This paper shows how genetic algorithms may be applied in the optimization of standard function processing (gray scale) morphological filters. It also shows how these genetic algorithms can be implemented in parallel on both massively parallel computers and clusters of loosely coupled workstations.

The paper is divided into 12 sections. Sections 2 and 3 give brief introductions to morphological filters and genetic algorithms, respectively. Section 4 describes the morphological filter design problem and then goes on to describe how a genetic algorithm can be utilized in morphological filter optimization. Section 5 shows the application of this morphological filter genetic algorithm to an actual optimization problem to prove its capabilities. Section 6 and 7 introduce and review parallel genetic algorithms. In Sec. 8 a new parallel genetic model is developed, and its implementation is described in Sec. 9. This section shows results in terms of optimization and parallelization efficiency, which attest to the algorithm being capable of excellent performance. Section 10 shows some examples of applications of the developed design tool.

2 Morphological Filters

Morphological filters, applied to images, are nonlinear signal transformations that locally modify their geometric features. They are based on the concepts of mathematical morphology and are related to the basic operations of set theory and integral geometry. Morphological transformations employ specific sequences of neighborhood transformations to measure certain geometric features, which may be useful.

Paper 97-017 received Sep. 1995; accepted for publication May 1997.
1017-9909/97/$10.00 © 1997 SPIE and IS&T.

Structures within images can be analyzed by selecting a filter able to interact properly with these structures. Describing the structures of an image by linking geometrical patterns at various locations is an idea quantified with the concept of the structuring element. The structuring element is a set or function describing some simple shape. Application of this structuring element allows various types of information to be obtained from the original image.

The fundamental morphological operations are dilation and erosion. Generally, in practice, dilations and erosions are usually employed in pairs, either dilation of an image followed by the erosion of the dilated result, or erosion of an image followed by dilation of the eroded result. In either case, the result of iteratively applied dilations and erosions is an elimination of specific image detail smaller than the structuring element without the global distortion of unsuppressed features. These iterative procedures constitute the so-called basic morphological filters.

Morphological opening can be described as an erosion operation followed by a dilation of the eroded result, using the symmetric structuring element, with respect to the origin.

The dual operation of opening is the closing operation. Morphological closing can be described as a dilation operation followed by an erosion of the dilated result, using the symmetric structuring element with respect to the origin.

If the structuring element has a regular shape, both the opening and closing operations can be thought of as nonlinear filters, which smooth the contours of the original image.

In general, morphological filters are lattice operators that are idempotent and increasing.[13] In this paper we restrict ourselves to the class of morphological filters known as structural operations and to their influence on binary and gray scale images.

3 Genetic Algorithms

Genetic algorithms provide optimization tools in areas that do not yield readily to standard approaches.[14] They are simple to implement and easy to understand. A genetic algorithm (GA) is essentially an adaptive method that may be used to solve search and optimization problems using processes that are based on the mechanics of natural selection and natural genetics. In nature, populations evolve through many generations according to the principles of natural selection and "survival of the fittest." By mimicking this process, genetic algorithms are able to "evolve" solutions to real world problems, provided that they have been suitably encoded.

In natural environments, there is competition between individuals in a population for available resources such as food, water, etc. In addition, there is often competition between members of the same species to attract a mate. Individuals that are most successful in surviving and attracting mates will produce relatively large numbers of offspring. Those individuals that perform poorly will tend to produce relatively few or even no offspring at all. This means that genetic material from highly adapted or "fit" individuals spreads to an increasing number of individuals in each successive generation. The combination of beneficial characteristics from different ancestors can sometimes produce "superfit" offspring, which are more fit than either parent. In this manner, species undergo the process of evolution to become better suited to their environment.

GAs use an analogy of behavior in nature. A population of individuals is maintained, with each individual representing a possible solution to a given problem. Each of these individuals is assigned a "fitness value" according to how well it solves the problem. (This is equivalent in nature to assessing the effectiveness of an organism in competing for available resources.) Those individuals with high fitness values are given opportunities to "reproduce," by "cross breeding" with other individuals in the population, producing new individuals as "offspring." These offspring share some features taken from each parent. The members of the population having lower fitness values are less likely to get selected for reproduction and so die out.

Thus, a new population of possible solutions is created by the selection of the best performing individuals from the current generation and mating them to produce a new set of individuals to be tested. This new generation will contain a higher proportion of those characteristics possessed by the fitter members of the previous generation. In this manner, over successive generations, beneficial characteristics are dispersed throughout the population, being mixed and exchanged with other beneficial characteristics. By favoring mating of the fitter individuals, the most promising areas of the search space are explored. Providing the GA has been well designed, the population will tend to converge to an optimal solution.

For a more detailed overview of genetic algorithms the reader is referred to Refs. 15 and 16, which can be obtained at the internet site: http://www.cs.cf.ac.uk/Papers. Further recommended reading is Refs. 6, 14, 17, and 18.

4 Using Genetic Algorithms in the Optimization of Morphological Filters

4.1 *Optimization Problem*

The morphological filters under consideration in this paper are function processing or gray scale morphological filters,[19,20] comprising of iterated applications of the basic gray scale morphological operations of erosion and dilation, including filters such as opening, closing, open-closing, and close-opening.

For a given filtering application, when considering the choice of morphological filter, there exist a vast number of alternative filters from which to choose. Factors such as size and shape of structuring element, sequence of morphological operator, etc., all contribute to the variety of possible morphological filter choice. For example, considering the sequence of morphological operators, if the situation exists where the number of operators in the sequence is four, and the choice of morphological operators is from the set {erosion, dilation, do-nothing}, (where do-nothing is, essentially, an identity operation), then there are a total of 31 possible sequences of morphological operators (i.e., $2^0+2^1+2^2+2^3+2^4=31$, since some sequences, such as {erosion, dilation, do-nothing, do-nothing} and {do-nothing, do-nothing, erosion, dilation}, are equivalent). One then considers that for each of these possible sequences of morphological operators there is an associated structuring element. If, for example, this structuring element has a 5×5

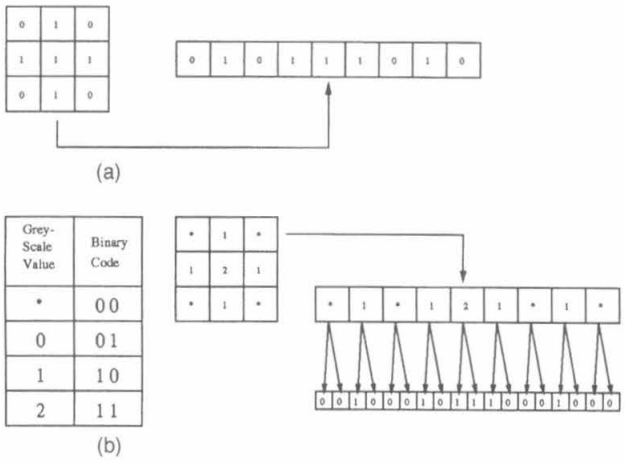


Fig. 1 Coding structuring elements: (a) coding a binary structuring element and (b) coding a gray scale structuring element.

region of support, and the range of each position within the structuring element's region of support is 0 to 4, then there are a total of 5^{25} possible structuring elements from which to choose for each possible choice of morphological operator sequence. Thus, the total number of variations of morphological operator sequence and structuring element, for this particular illustration, is $31\times(5^{25})$. This gives an idea of the size of the search space for the morphological filter design problem.

Genetic algorithms have been established as a valid approach to problems requiring efficient, effective, and robust search in complex spaces. It would therefore seem entirely appropriate to use genetic algorithms in the search for optimal morphological filters.

In order for a GA to be used in the design of morphological filters, the problem has to be encoded for genetic search, i.e., the parameters of the optimization problem have to be mapped to a finite length string over some alphabet. As a rule, when coding a problem for genetic search, one should select the smallest possible alphabet that permits a natural expression of the problem (the principle of minimal alphabets[14]). This leads to the use of the binary digits {0,1} as the alphabet. How then should the parameters of the morphological filter design problem be coded?

In the case of the basic morphological filters there are two basic parameters, the structuring element and the sequence of morphological filters. It is therefore necessary to code these parameters and map them to positions in the chromosome.

4.2 *Coding the Structuring Element*

Figures 1(a) and 1(b) show the general method for coding a structuring element. A binary structuring element is simple to code merely by mapping the 0s and 1s to their appropriate positions in the structuring element portion of the chromosome. Figure 1(a) shows the coding of a binary structuring element. For gray scale structuring elements, if the range of possible values for each position within the structuring element's region of support is $0-I_{\max}$, these integer values can be coded to binary strings, the length of which is $\lfloor \log_2(I_{\max}) \rfloor + 1$ binary digits. Each of these binary strings representing an integer value for each position within the structuring element's region of support can then be mapped to their appropriate positions in the structuring element portion of the chromosome. Figure 1(b) shows the coding of a gray scale structuring element.

Table 1 Example of a code suitable for encoding positions both within and outside the structuring element's region of support.

Binary Code	Corresponding value in position within structuring element's overall dimensions
0 0 0	*
0 0 1	0
0 1 0	1
0 1 1	2
1 0 0	3
1 0 1	4
1 1 0	5
1 1 1	6

If the overall dimensions of the structuring element are fixed, it may be that not all positions within this structure are within the structuring element's region of support. In order for this to be taken into account in the GA optimization, it is necessary for positions outside the structuring element's region of support but within the overall dimensions of the structuring element (don't-care positions, if you like) to be distinguished from those positions within the region of support. This must be incorporated into the coding technique. To code the positions outside the region of support, the possible range of values for each position within the overall dimensions of the structuring element now includes $0-I_{\max}$ and * (don't care). An example of a suitable code, able to encode the range of possible values for each position within the structuring element's region of support as well as those special case positions which lie outside the region of support but within the overall dimensions of the structuring element, is illustrated in Table 1.

4.3 *Coding the Sequence of Morphological Operations*

When considering the sequence of morphological operations in the context of design of the basic morphological filters, there are two basic decisions to be made: (1) the set of individual morphological operators from which to choose, and (2) the number of morphological operations in the sequence. For example, for the close-open morphological filter, the set of morphological operators necessary is {dilation, erosion} and the sequence length required is four, i.e., four separate morphological operations: dilate-erode-erode-dilate.

In order that the GA should be able to perform optimization over the entire search space, it is necessary to include the do-nothing operation (which is, as mentioned previously, essentially an identity operation) to the set of morphological operations. This is due to the fact that the length of the sequence of morphological operations is fixed in the genetic algorithm, but it is desirable to include in the search space all the combinations of basic morphological operations from the simple erosion and dilation through the

Table 2 Example of a code for morphological operations, if choice is from {erosion, dilation, do-nothing}.

Binary Code	Morphological Operator
0 0	Do-Nothing
0 1	Do-Nothing
1 0	Erosion
1 1	Dilation

close and open filters, to the open-close and close-open filters. Hence, if one were not to include the do-nothing (or identity) operation, then the search space would only include those filters that contain exactly four operations, each chosen from the set {erode, dilate}, and the search space would be severely restricted. So, as in the method discussed in this paper, the fundamental morphological operations are considered, and we have a set of morphological operations {dilation, erosion, do-nothing} from which to choose. If it is intended to search the range of basic morphological filters from the "do-nothing" filter through the simple erosion and dilation operations to the close-open and open-close filters, a sequence length of four is required. In general, having a choice of X_{op} individual morphological operations, the total number of choices of morphological operations to be considered in the GA optimization is in fact $X_{opact}=X_{op}+1$ (i.e., X_{op} plus the do-nothing operation). This number of morphological operations can be coded using a minimum of $\lfloor \log_2(X_{opact}) \rfloor + 1$ binary digits. Each member of the set of morphological operations to be considered in the GA optimization can be coded to binary strings, and these strings can then be mapped to their appropriate positions in the morphological sequence portion of the chromosome. Table 2 shows an example of codings for the set {dilation, erosion, do-nothing}. Figure 2 shows an example of the codings for a sequence length of four and the set of morphological operations as shown in Table 2.

It is necessary to ensure that each possible sequence of operations is unique, i.e., so that no combinations of operations in a sequence can be coded in more than one way, since some combinations of filter sequences are equivalent, e.g., {erode, dilate, do-nothing, do-nothing} and {do-nothing, erode, do-nothing, dilate}. This is accomplished by checking the sequence of operations portion of the chromosome after forming each new generation, and ensuring that all do-nothing operations are moved to the end of the sequence.

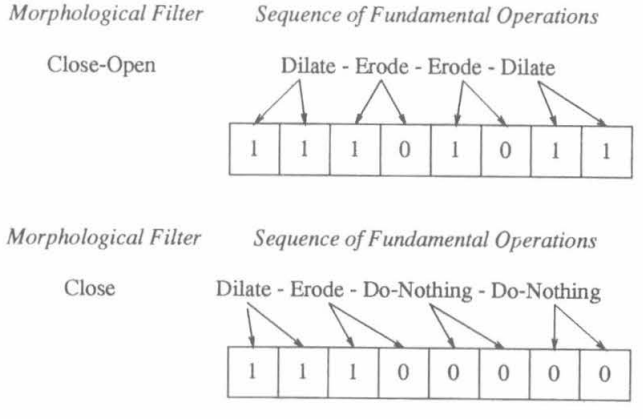


Fig. 2 Coding the sequence of operations.

4.4 *Combining the Coded Structuring Element and Sequence of Morphological Operations*

To form the complete chromosome, it is simply a matter of concatenating the two separate coded binary strings for the structuring element and the sequence of morphological operations.

Hence, the size of the search space is fixed: the overall dimensions of the structuring elements, i.e., the size of the region of the support and the maximum gray level values, and the maximum length of morphological operation sequence together with the choice of morphological operations. The GA is capable of searching for any gray level (function processing) morphological filter that is any combination of four operations from the set {erode, dilate, do-nothing}, which will use a structuring element (and it's symmetric counterpart, depending on the particular combination of morphological operations) chosen from all the possible variations within the overall region of support and maximum gray level value. Instances of filters employing structuring elements having gray level values of zero in the positions within the structuring element's region of support are essentially function set processing morphological filters.

4.5 *Fitness Function*

The fitness function has to provide some measure of the GA's performance in a particular environment and is based on some objective function. The GA was used for the purposes of this paper to search for the optimum filter for image processing problems, namely that of noise reduction, in the examples shown here. A common goal of optimization in noise reduction in image processing is to minimize the mean absolute error (MAE) between a reference image and the filtered noisy version of this image. However, as the objective of this type of optimization is the minimization of a cost function, it is necessary to map this objective function, $g(x)$, to a fitness function, $f(x)$.

$$f(x)=\begin{cases} C_{\max}-g(x), & g(x)<C_{\max} \\ 0 & \text{otherwise} \end{cases}.$$

Therefore, for the GA described in this paper:

1. For each chromosome, the morphological filter configuration represented by that particular chromosome is obtained by a decoding process, i.e., the reverse of the coding process described previously.
2. The fitness function for each chromosome is calculated as follows: Let d_1 denote the mean absolute error between two signals x and y, of length l.

$$d_1(x,y)=\frac{1}{l}\sum_{n=1}^{l}|x(n)-y(n)|.$$

Let r, s, and $\tilde{r}_i$ denote the ideal signal, the corrupted (noisy) signal, and the filtered result using the filter con-

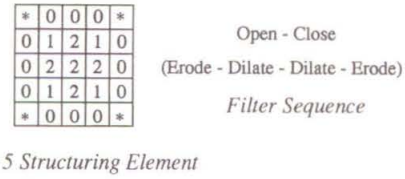


Fig. 3 Morphological filter used to test the GA.

figuration represented by the i'th chromosome, respectively. Let v_i denote the calculated fitness value for the filter configuration represented by the i'th chromosome. Then, if $d_1(s,r)<d_1(\vec{r}_i,r)$, the filtered result is actually worse than the corrupted signal with respect to mean absolute error and therefore the fitness value is set to zero. Otherwise set the fitness value $v_i=d_1(s,r)-d_1(\vec{r}_i,r)$, i.e., the improvement gained by the filtering using the morphological filter configuration corresponding to that particular chromosome.

5 Application of the GA

In order to test whether or not the GA described was capable of finding the optimum morphological filter for a particular application, and to observe the convergence characteristics, the GA was tested using a set of training images for which the optimum filter was known. An image that had been previously filtered with a known morphological filter, together with the original unfiltered image, were therefore used as the training set. Figure 3 shows the morphological filter used to filter the image. This filter was an open-close operation using a 5×5 spheroid structuring element. To further demonstrate the GA's ability to find the optimum morphological filter, the region of support of the structuring element in the GA was set at 7×7. Thus, if the GA performed correctly it would still find the optimum 5×5 structuring element within its 7×7 region of support.

Symmetric structuring element's are often employed for morphological transformations[20] and it is thus possible, but not necessary, to restrict the search space by encoding only the left-hand corner of a structuring element. The other positions within the structuring element can then be determined symmetrically.

To indicate how the GA proceeded toward the optimum morphological filter configuration, Fig. 4 shows the best morphological filters found by the GA at various stages through its process. It can be seen that by the 100th generation the GA has determined the correct sequence of morphological operations. By the 300th generation the GA has determined the correct overall size and shape of the region of support of the structuring element. By the 800th generation it can be seen that the GA has found the optimum morphological filter.

6 Introduction to Parallel Genetic Algorithms

Genetic algorithms have been applied successfully to the problem of morphological filter design. The only restriction to these approaches is the typically high demand of computation power for genetic algorithms. The fitness value for the chromosomes is determined by the performance of its associated morphological filter. Therefore several basic filter operations have to be computed for each chromosome of the population in every generation. The arithmetic complexity O for the calculation of a single fitness value results in

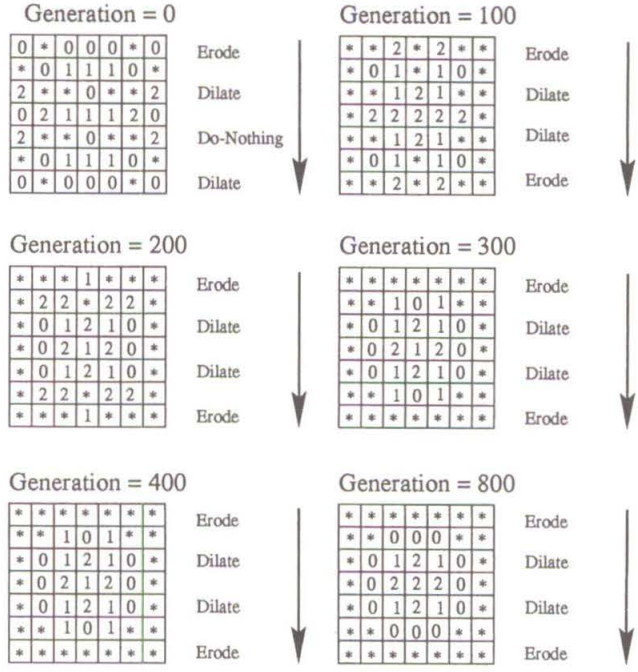


Fig. 4 Illustration of progression of GA toward optimum.

$$O=(n\times n)(m\times m)M.$$

Here n refers to the dimension of the filtered images, m refers to the dimension of the filter mask, and M is the number of performed morphological operations (erosion and dilation). For average problems, e.g., images of size 512×512 pixel, mask size 7×7, and 4 morphological operations, this leads to a calculation time of more than 30 s (on a T805 transputer or a Sun Sparc-station IPC), giving a total run time for a normal sequential GA (one population with 100 individual and 200 generations) of about 7 h. To achieve more acceptable computation times, which are required for real-time image analysis, it is necessary to use high performance parallel computers.

With the recent advancement and availability of various parallel computers, the next promising step is to map the genetic algorithm model onto a parallel computing model. Two major forces are driving this task: (1) the theoretical point of view, which has the aim of developing the most efficient parallelisation techniques for genetic algorithms, and (2) the application point of view, which requires practical, useful algorithms. This means algorithms that do not need special hardware or software and which give satisfactory speed-up on the available parallel computer system.

To fulfill both requests, this work is split into two parts. First, a new theoretical model of a population organization for a parallel genetic algorithm is introduced and discussed. Second, it is implemented on both a massively parallel, high-performance computer and a common cluster of workstations, where the comparative performances are evaluated.

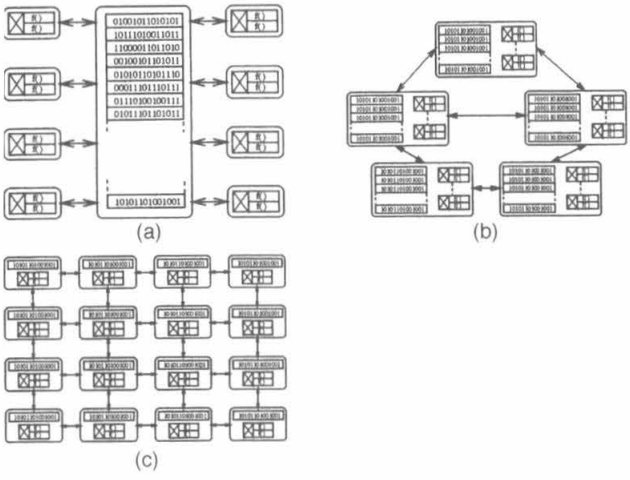


Fig. 5 Parallelization methods for parallel genetic algorithms: (a) standard parallel genetic algorithm, (b) coarse grained parallel genetic algorithms, and (c) fine grained parallel genetic algorithm.

7 Review of Parallel Genetic Algorithms

In recent years much research work has been done in the area of parallel genetic algorithms. In general, three different types of parallel genetic algorithms have been introduced. These approaches are known as standard or centralized, coarse grained or distributed, and fine grained parallel genetic algorithms.[21–23] (Other authors have different names for similar subdivisions in Refs. 7 and 24.) Figure 5 shows the structure of these methods, which can be described as follows.

Standard parallel genetic algorithm. The standard model basically uses a sequential GA but does the genetic operations and fitness evaluation in parallel. Due to the fact that any two individuals of the population can be paired, the pairs must be selected sequentially, otherwise a fully connected graph of individuals is required, which would lead to an excessive communication overhead. In the selection step a knowledge of global information, such as minimum, maximum, and average fitness values, is necessary to calculate the number of copies for each individual selected for the next generation.

Coarse grained parallel genetic algorithms. Coarse grained parallel genetic algorithms divide the entire GA population into subpopulations and execute a standard GA on each. One subpopulation can be mapped onto a single processor or onto a group of processors using the parallelization method described before. The single subpopulations exchange information (individuals) between each other. Different communication strategies, using blocking, nonblocking, or virtual parallel communication, have been employed by various authors.

Fine grained parallel genetic algorithms. Fine grained parallel genetic algorithms distribute the entire population onto a low dimension, locally connected graph such as a grid. The GA acts on each member of the population in parallel. Evaluation, selection, and reproduction then occur only on a local basis between neighbors on the graph.[7]

7.1 *Search Efficiency*

It is important that all the parallelization models, which modify the sequential GA, do not search less efficiently than the sequential model.

The sequential GA is based on an idealized population model for which Holland[6] and others[25] have theoretically proven its validity. This model is based on the description of an evolutionary process which uses a random selection strategy, allowing a pairing between any two individuals. Mülhlenbein pointed out in Ref. 26 that the basic problem was already mentioned by Darwin[27]: ''Intercrossing plays a very important part in nature in keeping the individuals uniform in character. In animals which unite for each birth, but which wander a little, a new and improved variety might be quickly formed on any spot.... A local variety when once thus formed might subsequently slowly spread to other districts.'' In GAs we want a fast evolution and not to keep the individuals uniform.

Mülhlenbein also mentioned that Darwin's conjecture of local populations with some isolation evolve faster than a large population was confirmed later by experiments and qualitative analysis.[28] Nearly all experiments with a parallel genetic algorithm have shown that the search efficiency in distributed parallel genetic algorithms using subpopulations is higher than in sequential algorithms, which use only one entire population.

7.2 *Parallel Genetic Algorithms on Workstation Clusters*

Most of the parallel genetic algorithms that have been developed so far have been tested on homogeneous parallel computers.

Bierwirth et al. published in Ref. 29 a realization of a parallel genetic algorithm which is based on a virtual shared memory communication environment. This concept has the advantage that the algorithms can be outlined in a powerful high level programming syntax, which allows for ease of design. The disadvantage is that the realization of a virtual shared memory concept on a workstation cluster suffers from the long offset time for any interprocess (intermemory) communication, which results with a poor parallelization efficiency. This rises dramatically with an increase in the number of workstations that are used.

Opaterny compared the parallel genetic algorithm for workstation clusters, which he developed in his thesis,[30] with a similar algorithm for massively parallel computers. Without giving explicit run time results, he announced that his coarse grained genetic algorithm did not run with a satisfying efficiency due to the communication bottleneck on the workstation cluster.

Both approaches to implementing parallel genetic algorithms on inhomogeneous parallel computers use common genetic models without adapting them to the special properties of the hardware. This results in a waste of computation power and longer run times. Until now there have been no parallel genetic models which can cope efficiently with inhomogeneous multiuser parallel computers.

8 General Parallel Genetic Algorithm

All the models invented so far have a common bottleneck, the synchronization between either single subpopulations or within the processing of the following generation for a

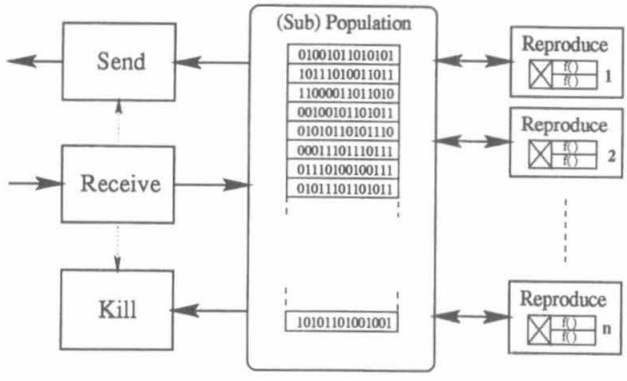


Fig. 6 Population model of the GPGA.

single subpopulation. Even if the synchronization can be made as a soft synchronization, like the handing-over of the baton in a relay race, where a certain interval is defined in which the communication has to take place, a loss of efficiency could occur due to idle processors if, for example, the execution time for a generation differs dramatically between subpopulations. This could be caused by (1) a different processor load, a different processor speed or a different number of processors working on a subpopulation, or (2) a different number of instructions required to process the transition from one generation to the following for a subpopulation. The latter can be caused by different population sizes or different times needed to calculate a single fitness value. This is due to the fact that individuals can differ in their number of morphological operations and that, for offspring identical with one of their parents, the fitness value is simply copied and not calculated a second time. A new population organization and communication scheme has to be developed which does not suffer from the previously outlined disadvantages. The following subsections describe these models.

8.1 *Genetic Model*

In the general parallel genetic algorithm (GPGA) a population is represented as a list of individuals ordered in terms of fitness values. Each individual has a rank defined by its position in the list: the lowest for the last individual (worst fitness value) and the highest for the first one (fittest individual). At any time, new individuals can be added to the list (sorted into the right position) or deleted, which may change the rank of other individuals. The access to the list of individuals is organized with a semaphore concept, so that different tasks can concurrently add, copy, or delete individuals. Due to this, the entire number of members within a population can change at run time. A control task monitors the population size and deletes individuals from the population if the population size grows over a threshold. The selection of individuals to be deleted is made at random, with a higher probability for individuals with a lower rank to be selected, which is a ''survival of the fittest'' strategy.

8.1.1 *Population organization*

Each population is modified by four different types of tasks (Fig. 6) that can operate in parallel. The first type of task (reproduce) selects a couple of individuals as parents and performs the genetic operations on them, calculates the fitness value for the offspring, and sorts them back into the population. The second type of task (receive) involves collecting messages from other subpopulations. These messages contain a single individual, its fitness value, and a time value. The individuals are sorted into the ordered population and the time value is used to modify the optimal population size (see Sec. 8.3). The third type of task (send) involves transmitting copies of individuals to other subpopulations. The selection from the population is made in the same way as the one for selecting parents. The target population is chosen at random from a given set of subpopulations. Depending on the network, this set could contain only local neighborhood populations (for a grid array) or all subpopulations (for bus networks). The fourth type of task (kill) is used in supervising the population size.

8.1.2 *Selection, genetic operations, and reproduction*

The reproduction task performs selection parents, evaluation of fitness values, and reproduction. The probability for individuals being selected for reproduction is made referring to their rank and not to their fitness value by selecting parents at random while using a skewed probability distribution, preferring fitter individuals to be selected. This has the advantage that no fitness scaling is necessary to overcome the problem of unpleasant fitness distributions, such as an accumulation of good fitness values close together and some poor values at the far end of the fitness range. Individuals selected as parents are copied and genetic operations are performed to produce offspring. The fitness value of the offspring is calculated and they are sorted back into the population, which has now increased by two individuals. This procedure of reproducing is repeated until a termination condition stops the algorithm. This mechanism allows individuals from different generations to crossover or the same parents to have more than one pair of offspring.

8.2 *Levels of Parallelism*

The parallelization of the algorithm takes part on three different levels. Each of these levels can be scaled to different sizes to match the algorithm as much as possible to the underlying parallel computer. The first level of parallelization is based on a number of subpopulations calculated in parallel. The second level consists of concurrent working reproduction tasks (Fig. 6). These task do not have any supervising master and are working completely asynchronously. If an appropriate parallel computer is available in the third level of parallelization, each of the tasks calculating the fitness values can employ more processors for calculating the fitness values of the offspring. Here a data parallel concept used.

8.3 *Synchronization and Communication Scheme*

The size for each population is allowed to vary within a given range. If the time value, which represents the average time needed to calculate a fitness value multiplied by the actual population size for the processor sending the message, is less than the value on the local processor the population size is decreased, otherwise it is increased. This mechanism implies a load balance to achieve a soft syn-

chronization between subpopulations. The ratio of sending individuals is controlled with a function having the calculation time for a single fitness value and a free eligible value as parameter, so it can be adapted to the available communication power of the network used. A dramatic change of the communication power (e.g., a complete temporary disconnection of a processing node) between parts of the parallel computer just leads to an increase in the isolation of the effected subpopulations, but does not disturb the run of the algorithm.

The fault of a processing node results in the loss of the individuals contained in the affected populations and not in the termination of the whole algorithm, which is a partial fault tolerance.

9 Implementation of the GPGA

9.1 *Hardware Requirement and Scalability*

The available parallel computer is the deciding factor for the grade of parallelization in the different levels. Splitting up the entire population into subpopulations leads to better optimization results, but can only reduce the run time to a definite limit, because each subpopulation still has to reproduce itself a couple of times to allow the genetic scheme to improve the fitness values to the desired result. If the number of subpopulations rises over a certain level, it will be more likely to have similar subpopulations searching for the same optimum in different subpopulations, leading to redundancy and poor efficiency. The communication network demanded for the subpopulation method needs to deal with a load depending on the time it takes to calculate a single fitness value and the number of subpopulations. Experimentally, it was found that each subpopulation should be able to send a message of 10^2 to 10^3 bytes of individual and additional information, at least after a number of calculated fitness values equal to its population size. If the processing nodes are connected with a single bus, the available bus capacity should equal the sum of the requirements of all subpopulations.

The second level of parallelization can be used with a reasonable efficiency up to a number of processing nodes less than half of the population size, where only individuals for the current generation can be calculated in parallel. This causes the best possible achievable speed-up factor to be less than the number of individuals within a population. It was experimentally found that the connection between the master and each worker should be able to transfer a message in a time slot, which is less than 10% of the time needed to calculate a single fitness value.

If a single processing node cannot cope with the problem of calculating fitness values, for example, if the available memory is not big enough to store a whole image or further processing nodes are available, then the data parallel method can be used. The maximal number of efficiently used processing nodes depends strongly on the size of the image and the communication capacity between the nodes. It is also possible to use a pipeline concept for calculating the sequence of filter operations on different processing nodes. This has the disadvantage of the filtered images being transferred between different nodes, which is only effective if the pipeline elements are connected with a high speed communication structure.

In general, it has been found that the total amount of processing nodes that can theoretically be used to calculate an average optimization problem, with an acceptable efficiency, sums up to several thousand processing nodes. This number is based on experiments and should give an impression of the possible scalability. Tests have shown that 10^2 to 10^3 is a satisfactory value for the number of subpopulations with a population size of approximately 10^2 individuals. The appropriate number of processing nodes calculating a single fitness value with an efficiency above 0.85 results in 10^2 using a transputer network. It is not assumed that a generalization of this value to other GAs or all possible morphological tasks is always allowed.

9.2 *Test Hardware*

The computers used to test the algorithm can be divided into massively parallel computers and into loosely coupled clusters of processing nodes (workstations).

Two massively parallel computer systems have been used, a Parsytec Super Cluster and a Parsytec Giga Cluster. Both located at the Technical University Hamburg-Harburg. The Super Cluster consists of 128 identical T805 transputers modules (30 MHz, 4 Mbytes). A torus topology is used as the interconnection structure. The operating system is a single user multiple processing system. This computer is a strictly homogeneous parallel computer. The Giga Cluster is based on 128 M601 (PowerPC) processors with 16 Mbyte local RAM memory. The processors are interconnected in a grid topology, using 256 transputers as communication processors. This sums up to a total semiconductor memory capacity of 1.0 Gigabyte and a peak performance of 10 GFLOPS.

The second type of parallel computer is a cluster of workstations. One cluster (divided into five subclusters) is made up of 100 Sun Sparc-station1s and a second cluster contains 10 Sun Sparc-station20s or multiprocessor Sun Sparc-stations. These clusters are shared with other users.

9.3 *Software Environment*

The algorithms are written in ISO C and make use of special libraries for parallel processing. While on the workstation cluster and on the PowerPC computer, only the PVM message passing environment[31] was used. On the transputer based system, the algorithms have been integrated in the parallel image processing system PIPS (Ref. 32) using ParC.

9.4 *Assessment of the Results*

9.4.1 *Inhomogeneous multiuser parallel computers*

Measuring values for speed-up or efficiency on inhomogeneous computer systems is complicated. It is not possible to run an algorithm in different scales under the same environment. The ratio between the computation power and the number of processors is not linear and the behavior of the algorithm is also dependent on the mapping onto the machine. If the parallel computer has a multiuser operating system, the available computation and communication power changes between different test runs and also during a

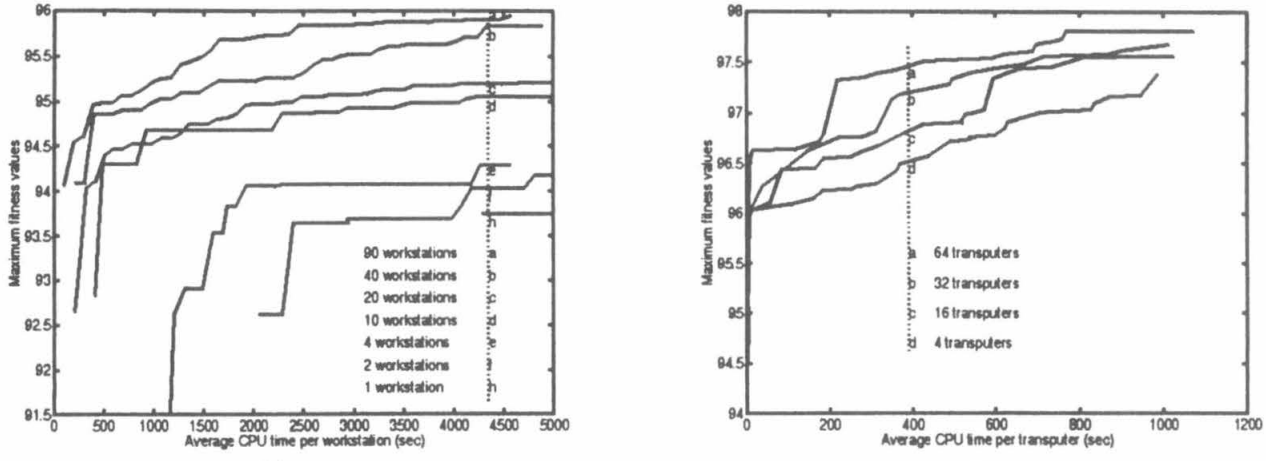


Fig. 7 Fitness values on (a) a workstation cluster and (b) a massively parallel computer (Super Cluster).

single run. In this case, it is not possible to repeat runs under the same circumstances. This is the case on workstation clusters. To compare the results, the load of the cluster that occurs as a result of other users is classified into three types: "heavy" for peak hours, "normal" for average times, and "low" for weekend and late night times. All results published here are calculated during low load times, which means that the used CPU time nearly equals the real run time (allowance less than 10%).

9.4.2 *Quality of a genetic search algorithm*

Although every GA has, theoretically, the ability to find the optimal solution for the given problem, it is not possible to recognize it (if the optimum is not known *a priori*) and it might take an intractable period of time for the search. It is quite common to stop the GA after a termination condition is reached and take the best individual at that time as a (sub)optimal result. To test the GPGA, three different termination conditions have been used: (1) a desired fitness value, (2) a maximal number of offsprings to calculate, and (3) a maximal real run time. Next to the termination conditions used, the chosen values for the threshold are also decisive factors. Some parameters of a GA lead very quickly to relatively good values but take a long time to achieve fitness values close to the optimum. This means that it is quite complicated and needs a lot of expert knowledge if GAs are to be compared and assessed in general.

Even if the real time values seem to be the most important ones for a user, they do not give a fair comparison on multiuser machines due to the different load of the computers.

9.5 *Results*

9.5.1 *Search efficiency*

Figure 7 shows the maximal fitness values achieved by the GPGA for a standard task and plotted against the CPU time needed to calculate them. Figure 7(a) is measured on the workstation cluster and Fig. 7(b) the Super Cluster. Comparing the different curves, which resume from runs with different number of processing nodes used, it can be seen that the time needed to achieve certain fitness values decreases when the number of nodes increases. Due to different behaviors in time, a general speed-up diagram is of no use. For example, the best value of 95.9 is already reached on 90 workstations before the initialization (calculation of the fitness values of the random start population) is finished on a single workstation, while for the short period between 1000 and 1400 s the run with four workstations had a better maximal fitness value than the one with ten workstations. This is reasonable because of the stochastic elements in the genetic methods, which allow any population at any time to produce excellent offspring. Considering the performance over longer time intervals, it can be seen that a general improvement of the performance is proportional to the number of workstations used. It is even more important that runs with a few subpopulations do not reach a satisfactory fitness at all.

9.5.2 *Parallelism efficiency*

The diagram in Fig. 8(b) shows parallelization efficiencies for the Super Cluster and the workstation cluster. The values for the workstation cluster are calculated as

$$\text{eff} = \frac{t_1}{i \times t_i}; \quad i = 1,2,4,10,20,40,90,$$

where i is the number of workstations used and t_i is the averaged CPU time needed to calculate the first 2000 fitness values. The graph for the massively parallel Super Cluster is made up in the same way, using the number of calculated fitness values after 1000 s in relation to the number of processing nodes used. It is obvious that the efficiency for the Super Cluster decreases due to longer communication paths between different processing nodes, which leads to an increased communication load on nodes in between the paths while the efficiency for the workstation cluster seems to have a stochastic behavior. Figure 8(a) displays the efficiency values resulting from the Giga Clus-

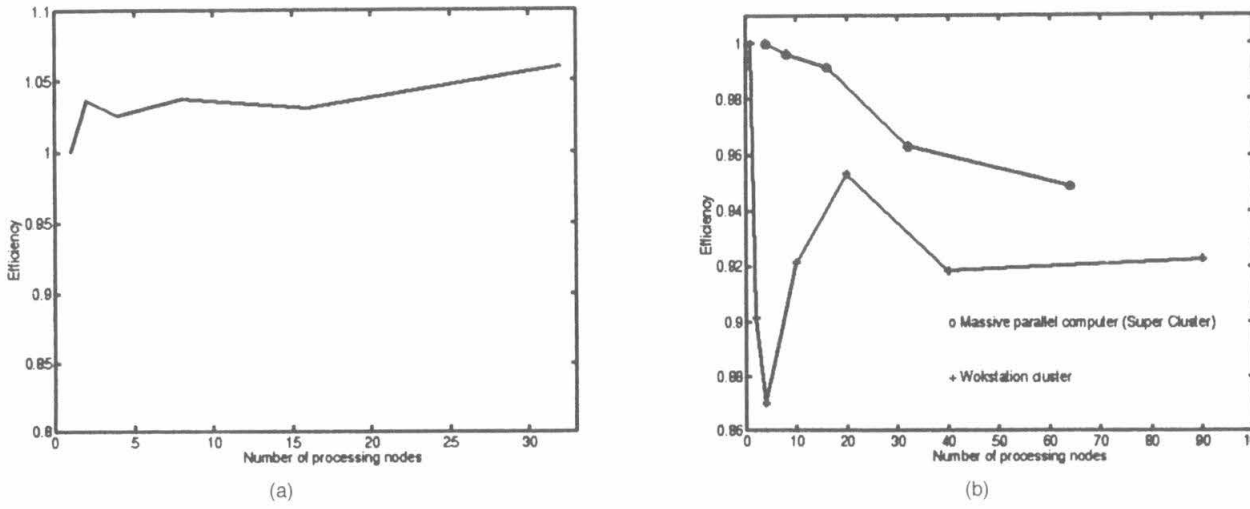


Fig. 8 Speed-up for (a) the Giga Cluster and for (b) the workstation cluster and the Super Cluster.

ter. These values are greater than 1 (super linear) and do not decrease when the number of processing nodes increases. The first is due to a constant task for the user interface, which does not increase in proportion with the number of processing nodes. The latter can be explained with the powerful interconnection network between the nodes, which does not provide an additional load for the computing units.

The overall high values certify the algorithm on both parallel systems having a very good performance.

10 Results on Tampere University of Technology Images

10.1 *Tampere University of Technology Database*

Within the ESPRIT basic research action nonlinear and adaptive techniques (NAT) in digital image processing, analysis, and computer vision, ESPRIT BRA 7130, a database of test images, has been established at the Tampere University of Technology (TUT) in Finland.[33] [The images

Table 3 Results of filtering TUT test images with morphological filters. Signal to noise ratio (SNR), peak SNR (PSNR), mean absolute error (MAE), and mean squared error (MSE).

	Noisy image				Filtered image				
Image	SNR	PSNR	MAE	MSE	SNR	PSNR	MAE	MSE	Comments
Airm0s3l01	3.03	15.60	26.46	1788.42	11.57	24.14	11.34	254.64	
Airm0s9l01	9.01	21.58	10.33	451.13	15.24	27.80	6.96	107.68	
Airm0s15l01	15.01	27.59	4.93	113.30	16.83	29.41	5.22	74.72	
Airm0s3l02	2.98	15.56	27.79	1807.93	11.76	24.34	11.17	239.25	
Airm0s9l02	9.02	21.59	12.83	450.01	14.52	27.10	7.81	126.67	
Airm0s15l02	15.00	27.57	6.34	113.75	15.93	28.51	6.30	91.93	no improvement
Airm0s3l10	3.02	15.60	33.74	1790.89	10.14	22.72	13.96	348.20	
Airm0s9l10	9.01	21.58	16.82	451.37	12.99	24.99	10.08	205.85	
Airm0s15l10	15.01	27.58	8.42	113.46	15.37	27.95	7.24	104.41	
Brim0s3l01	2.99	16.36	23.44	1503.54	11.54	24.90	10.74	210.26	
Brim0s9l01	8.99	22.36	9.38	377.90	13.48	26.85	7.50	134.44	
Brim0s15l01	14.98	28.35	4.53	95.29	14.99	28.35	4.52	95.12	no improvement
Brim0s3l02	2.98	16.34	25.71	1508.07	10.95	24.32	11.40	240.46	
Brim0s9l02	9.03	22.39	11.74	374.53	13.42	26.79	8.13	136.16	
Brim0s15l02	14.99	28.36	5.82	95.11	14.99	28.36	5.82	95.02	no improvement
Brim0s3l10	3.00	16.37	31.00	1498.81	9.70	23.07	13.54	320.54	
Brim0s9l10	9.01	22.38	15.46	375.96	12.26	25.63	10.10	178.05	
Brim0s15l10	15.01	28.37	7.75	94.74	15.01	28.37	7.74	94.71	no improvement

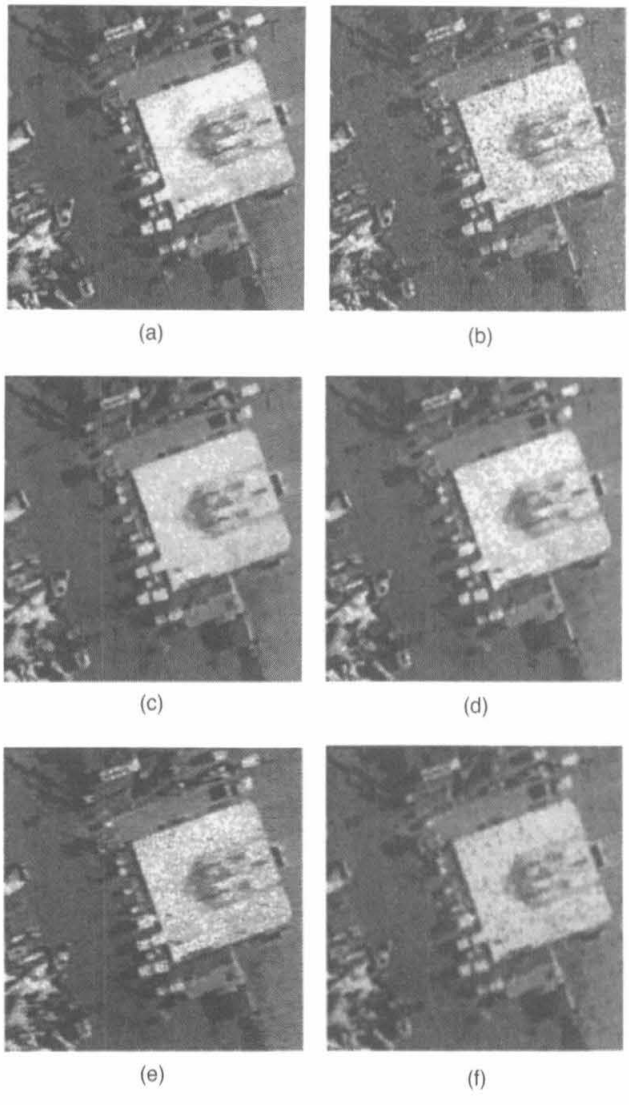


Fig. 9 Filter results; extracted parts from the test image Airm0s9l01: (a) original image, (b) noisy image, (c) rank-order morphological filtered, (d) median filtered, (e) standard-morphological filtered, and (f) linear filtered (normalized average).

can be obtained from the ftp server sigftp.cs.tut.fi (directory/pub/nat/imagedatabase).] This database should allow a comparison of several families of nonlinear filter classes.

Table 4 Comparison between different filter classes; filtered image is Airm0s9l01 from the TUT database.

Filter	SNR	PSNR	MAE	MSE
none	9.013	21.587	10.335	451.134
Linear (average)	11.288	23.862	11.602	281.006
Median	13.626	26.200	8.308	160.341
Standard Morphological	12.759	25.333	8.780	204.538
Rank-order Morphological	15.240	27.820	6.960	107.681

Table 3 presents the results for filtering the images with a new class of nonlinear filters currently under development at the University of Strathclyde, known as rank-order morphological filters.[34,35]

Rank-order morphological operations are a class of nonlinear operations that contain as a subclass the class of basic structural morphological operations. The basis for rank-order morphological operations is that the maximum and minimum operators of the morphological operations of dilation and erosion, respectively, are replaced with a rank-order operator. If all the gray level values within the structuring element's region of support are zero, i.e., the structuring element is "flat," the rank-order morphological operation reduces to a simple rank-order operation. Also, when the rank-order operator is set to unity, the basic rank-order morphological operations of rank-order dilation and rank-order erosion reduce to the basic morphological operations of dilation and erosion, respectively.

The size of the structuring element is limited to a maximal size of 3×3 pixels. The optimization of the filter is done with the general parallel genetic algorithm, as described previously. The filters are optimized on a subset (size 100×100 pixels) of the entire image and finally applied on the complete image. The optimization is done on a combination of the mean absolute error (MAE) and mean square error (MSE) values. The left side of Table 3 contains the value for the corrupted image and the right side for the filtered one.

10.2 *Comparison Between Different Filter Classes*

To give an impression of the different performances of different filter classes in a particular task, a test image with a medium intensity of impulsive noise (Airm0s9l01) was filtered with different filters. Figure 9 shows the original and filtered results, and Table 4 contains the appropriate error values. Four filter types have been used: (1) a linear average filter, (2) a standard morphological filter, (3) a median filter filter, and (4) a rank-order morphological filter. For all filters the window (structuring element) size was limited to 3×3 pixels. The linear average filter has shown the poorest performance. This is because linear filters are optimal for Gaussian noise but cannot cope adequately with different types of noise. The median filter performed slightly better than the standard morphological filter and the rank-order morphological filter has clearly shown the best results. Both morphological filters have been designed with the GPGA.

11 Conclusions

In this paper an optimization method for a particular class of nonlinear filters, morphological filters, has been described. This method uses a genetic algorithm in the optimization process. The optimization is based on a search for the optimum filter within the space of all possible filters. Due to the complex shape and the huge size of the search space, all deterministic design methods tend to be computationally intractable. It has been shown that GAs provide a simple tool in complex search problems without making any particular assumptions about the correlation of the search space. The expense in computation time for the GA has been limited to an acceptable duration by developing a parallel genetic model, which allows the use of either high-

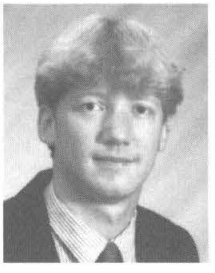
erformance massively parallel computers or larger workation clusters in an very efficient way. Test results to ove both the speed-up and the design quality have been own. Methods of optimizing further classes of nonlinear lters have already been developed using similar methods.

cknowledgments

arts of this work have been carried out at the University of echnology in Hamburg-Harburg, Germany. We would ke to thank Professor Burkhardt for making the parallel omputers available for this project and G. Schreiber and I. Nölle for assisting in various tasks.

eferences

1. J. Serra, *Image Analysis and Mathematical Morphology*, Vol. 1, Academic Press, San Diego, CA (1982).
2. R. P. Loce and E. R. Dougherty, "Using structuring element libraries to design suboptimal morphological filters," *Proc. SPIE* **1568**, 233–246 (1991).
3. R. Loce and E. Dougherty, "Facilitation of optimal morphological filter design via structuring element libraries and design constraints," *Opt. Eng.* **31**(5), 1008–1025 (1992).
4. E. R. Dougherty, "Optimal mean-square n-observation digital morphological filters-part i:optimal binary filters," *J. Comput. Vision, Graphics and Image Processing–Image Understanding (CVGIP)* **55**(1), 36–54 (January 1992).
5. E. R. Dougherty, "Optimal mean-square n-observation digital morphological filters-part ii:optimal gray-scale filters," *J. Comput. Vision, Graphics and Image Processing–Image Understanding (CVGIP)* **55**(1), 55–72 (January 1992).
6. H. J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, Cambridge (1992).
7. D. Macfarlane and I. East, "An investigation of several parallel genetic algorithms," in *Tools and Techniques for Transputer Applications OUG-12*, Stephen J. Turner, Ed., pp. 60–67, IOS Press, Amsterdam (1990).
8. P. Kraft, S. Marshall, J. J. Soraghan, and N. R. Harvey, "Parallel genetic algorithms for optimizing morphological filters," in *IEE Proc. 5th International Conf. on Image Processing and its Applications (IPA '95)*, **410**, 762–766 (1995).
9. J. Bala and H. Wechsler, "Shape analysis using genetic algorithms," *Patt. Recog. Lett.*, **14**, 965–973 (1993).
10. R. Ehrhardt, "Morphological filter design with genetic algorithms," *Proc. SPIE* **2300**, 2–12 (1994).
11. C. H. Chu, "A genetic algorithm approach to the configuration of stack filters," in *Proc. Third International Conf. on Genetic Algorithms (ICGA '89)*, pp. 219–224, June 4–7, 1989, San Mateo CA.
12. H. Huttunen, P. Kuosmanen, L. Koskinen, and J. Astola, "Optimization of soft morphological filters by genetic algorithms," *Proc. SPIE* **2300**, 13–24 (1994).
13. *Image Analysis and Mathematical Morphology, Vol. 2: Theoretical Advances*, J. Serra, Ed., Academic Press, New York (1988).
14. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA (1989).
15. R. R. Martin, D. Beasley, and D. R. Bull, "An overview of genetic algorithms: Part 1, fundamentals," Technical report, Dept. of Computing Mathematics, Univ. Cardiff, U.K. (1993).
16. R. R. Martin, D. Beasley, and D. R. Bull, "An overview of genetic algorithms: Part 2, research topics," Technical report, Dept. of Computing Mathematics, Univ. Cardiff, U.K. (1993).
17. D. E. Goldberg, "Zen and the art of genetic algorithms," in *Proc. Third International Conf. on Genetic Algorithms*, pp. 80–85, Fairfax, VA (June 1989).
18. *The Hitch-Hiker's Guide to Evolutionary Computation (FAQ in comp.ai.genetic)*, J. Heitkötter and D. Beasley, Eds., 2.4 ed. (Dec. 1994).
19. X. Zhuang, R. M. Haralick, and S. R. Sternberg, "Image analysis using mathematical morphology," *IEEE Trans. Patt. Anal. Mach. Intell.* **PAMI-9**(4), 533–550 (July 1987).
20. S. R. Sternberg, "Grayscale morphology," *Comput. Vision, Graphics and Image Process*, **35**, 333–355 (1986).
21. T. C. Belding, "The distributed genetic algorithm revisited," in *Proc. Sixth International Conf. on Genetic Algorithms*, pp. 114–121, Morgan Kaufmann (1995).
22. B. C. H. Turton, T. Arslan, and D. H. Horrocks, "A hardware architecture for a parallel genetic algorithm for image registration," in *IEE Electronics Division Colloquium on Genetic Algorithms in Image Processing and Vision* **193**, 11/1–6 (1994).
23. R. Bianchini and C. M. Brown, "Parallel genetic algorithms on distributed-memory architectures," S. Atkins and A. S. Wagner, Eds., *Transputer Research and Applications 6*, pp. 67–82, IOS Press, Amsterdam (1993).
24. G. D. McClurkin, R. A. Geary, and T. S. Durrani, "An investigation into the parallelising of genetic algorithms," in *Applications of Transputers 2*, D. J. Pritchard and C. J. Scott, Eds., pp. 581–587, IOS Press, Amsterdam (1996).
25. A. Bertoni and M. Dorigo, "Implicit parallelism in genetic algorithms," Technical Report TR-93-001, International Computer Science Institute, Berkeley, CA (January 1993).
26. H. Mühlenbein, "Parallel genetic algorithm, population dynamics and combinatorial optimization," in *Proc. Third International Conf. on Genetic Algorithms (ICGA '89)*, pp. 434–439 (1989).
27. C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, Murray, London (1859).
28. S. Wright, "The role of mutation, inbreeding, crossbreeding and selection in evolution," in *Proc. 6th International Congress on Genetics*, pp. 356–366 (1932).
29. C. Bierwirth, H. Kopfer, D. C. Mattfeld, and T. Utecht, "Parnet: Distributed realization of genetic algorithms in a workstation cluster," Technical report, FB. Wirtschaftswissenschaften, Univ. Bremen (1992).
30. T. Opaterny, "Ein virtuell paralleler genetischer algorithmus," Diplomarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institut für Mathematische Maschinen und Datenverarbeitung, July 1994.
31. G. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM 3.0 User's Guide and Reference Manual*, Oak Ridge National Laboratory, Tennessee, ORNL/TM-12187 (Sep 1994).
32. M. Nölle, G. Schreiber, and H. Schulz-Mirbach, "PIPS—a general purpose parallel image processing system," in *16 DAGM—Symposium Mustererkennung*, Wien, Austria, Sept. 1994, Reihe Informatik XPress, TU-Wien, ftp://www.til.tu-harburg.de/pub/papers/noe:schr:hsm:dagm94.ps.
33. ESPRIT BRA 7130 NAT Consortium, "Performance comparison of several nonlinear filter classes for image restoration," Technical report, Tampere Univ. Technology (1995).
34. N. R. Harvey and S. Marshall, "Rank-order morphological filters: A new class of filters," in *IEEE Workshop on Nonlinear Signal and Image Processing*, pp. 975–978, Halkidiki, Greece (June 1994).
35. N. R. Harvey and S. Marshall, "Design of different classes of morphological filter using genetic algorithms," in *IEE Proc. 5th International Conf. on Image Processing and its Applications (IPA '95)*, **410**, 227–231 (1995).



Peter Kraft received his Diploma in electrical engineering from the University of Technology, Hamburg-Harburg, Germany, in 1994, and his MPhil in parallel image processing from the University of Strathclyde, Scotland, in 1996. He is currently a Research Engineer with Femutec Gmbh, Germany, where he is involved in the field of 3-D graphic automated grid generation.

Neal R. Harvey received his BEng honors degree in mechanical engineering from Hatfield Polytechnic, England, in 1989, and his MSc in information technology systems from the University of Strathclyde, Scotland, in 1992. He is currently a Research Fellow in the Signal Processing Division of the electronic and electrical engineering department at the University of Strathclyde.

Stephen Marshall received his BSc first class honors in electrical and electronic engineering from the University of Nottingham, England, in 1979, and his PhD in image processing at the University of Strathclyde, Scotland, in 1989. He is now a Senior Lecturer in the Department of Electronic and Electrical Engineering, a member of the Institution of Electrical Engineers (IEE) Professional Group on Image Processing and Vision, and is a former director and chairman of the Scottish Chapter of the British Machine Vision Association.