Genetic beam search for gate matrix layout

K. Shahookar
W. Khamisani
P. Mazumder
S.M. Reddy

Indexing terms: CMOS, Gate matrix layout, Genetic algorithm

Abstract: The paper presents a novel implementation of the genetic algorithm in combination with beam search for gate matrix layout. This is a permutation problem, for which the traditional genetic crossover operator results in repetition of gates, and therefore the GA is not applicable without modification. However, the GA is a very efficient stochastic optimisation technique, and is easily parallelisable. To adapt it to the gate matrix layout problem, the principles of beam search have been used. The gates are ranked according to their connectivity with each other, and the ranks of the gates to be placed next to each other are picked by the GA. A beam value is used to restrict the ranks of the gates placed next to each other to a low value. This reduces the search space to be explored, and thus results in a more efficient search. The algorithm produces better results compared to a graph-theoretic approach on published netlists.

1 Introduction

The gate matrix layout style was introduced by Lopez and Law [1], and can be considered as an extension of the Weinberger style [2] for CMOS layouts. It is characterised by simplicity of design and quick turnaround time. However, it has been found that straightforward implementation leads to very inefficient usage of chip area. The gate matrix is composed of intersecting rows of diffusion and columns of polysilicon. All transistors having a common input are placed on a common polysilicon column. Connections between source and drain terminals of transistors connected in series/parallel, as well as connections to the polysilicon gates, are made by nets which run in rows of metal interconnect. At each intersection of a row and column, either a transistor can be fabricated, with two nets connecting the source and drain, or the net can form a contact with the polysilicon.

Fig. 1 shows an example circuit, consisting of multiple CMOS complex gates, its netlist, connectivity matrix, layout, and the symbolic representation of the layout, suitable for gate matrix design. Power and ground routing is not shown in the layout; it is assumed to be on a separate layer. The set of gates consists of one column for each set of transistors that have their gates connected together, plus one column for each output of the entire gate matrix circuit.

The netlist consists of the set of gates connected to each net. The connectivity matrix specifies how many nets connect each pair of gates.

Several graph algorithms as well as a stochastic optimisation technique and simulated annealing have been used for gate matrix optimisation. Well known graph algorithms are proposed in References 3–8. These give moderately good results and are quite fast. Ohtsuki *et al.* [3] gave the first graph-theoretic model for this problem using interval graphs. Simulated annealing was proposed for gate matrix layout in References 8 and 9. It gives good results, but is very slow and hard to parallelise efficiently.

2 Genetic algorithm

The genetic algorithm (GA) [10] is a stochastic optimisation algorithm that simulates the process of natural evolution (optimisation) in living creatures. It uses two basic processes from evolution: inheritance, or the passing of useful features from one generation to the next, and competition, or survival of the fittest, which results in weeding out the bad features from the individuals in the population.

The main advantages of the GA are

(i) It is adaptive, and learns from experience.

(ii) It has intrinsic parallelism.

(iii) It is efficient for complex problems.

(iv) It is easy to parallelise, especially on a loosely coupled workstation network, without much communication overhead [11].

The GA starts with several alternative solutions to the optimisation problem, which are considered as individuals in a population. Often these solutions are coded as a binary string, called a chromosome. In the gate matrix problem, these consist of a representation of the gate permutation, encoded as described in the following Section. The initial population is usually constructed randomly. The GA then uses these individuals to produce a new generation of hopefully better solutions as follows.

First it does an evaluation using a user-provided fitness function. The next step is selection and repro-

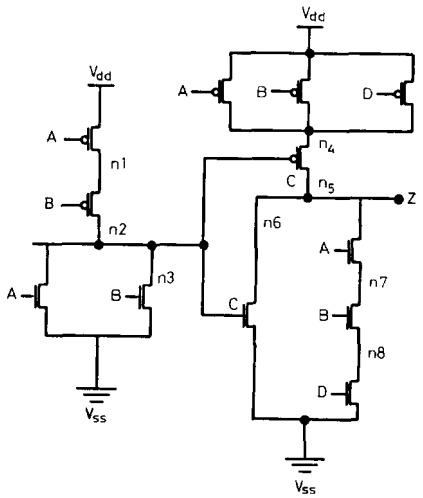
Paper 9879E (E10, E3), first received 18th March and in revised form 30th September 1993

K. Shahookar, W. Khamisani and P. Mazumder are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA

S.M. Reddy is with the Department of Electrical and Computer Engineering, University of Iowa, Iowa City, Iowa, USA

This research was partially supported by NSF grant 9013192 and the University of Michigan faculty grant.

duction. For each individual, a number of copies are made, proportional to its fitness, while keeping the population size constant. The least fit individuals are deleted. Therefore, a fitter individual, hopefully containing some useful features, has a higher probability of multiplying. This is the survival of the fittest part of the GA.

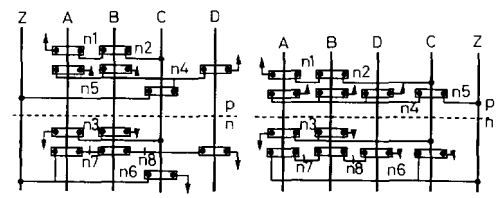


a

$n1 = \{A, B\}$
$n2 = \{B, C\}$
$n3 = \{A, B, C\}$
$n4 = \{A, B, C, D\}$
$n5 = \{C, Z\}$
$n6 = \{A, C, Z\}$
$n7 = \{A, B\}$
$n8 = \{B, D\}$

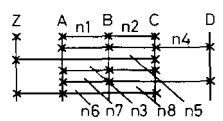
	A	*B*	*C*	*D*	*Z*
A	0	4	3	1	1
B	4	0	3	2	0
C	3	3	0	1	2
D	1	2	1	0	0
Z	1	0	2	0	0

b

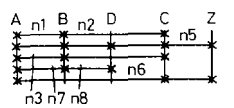


c

d



e



f

The next step is crossover, where individuals are chosen two at a time, at random, as parents. They are converted into two new individuals, called offspring, by exchanging parts of their structure. Thus, each offspring inherits a combination of features from both parents. This enables the GA to try out various features in different combinations, and see whether or not they still retain their fitness.

The next step is mutation. An incremental change is made to each member of the population, with a small probability. After mutation is performed on an individual, it no longer has just the combination of features inherited from its two parents, but also incorporates the additional change caused by mutation. This ensures that the GA can explore new features that may not yet be in the population. It makes the entire search space reachable despite the finite population size.

This completes the production of a new generation. This process is repeated for several generations, and the fittest gate permutation seen in the entire run is output at the end.

3 GA and beam search

The problem of determining the gate permutation is NP-complete, and is very similar to the travelling salesman problem. These permutation problems result in conflicts in the implementation of the traditional GA, as discussed in detail in Reference 12. For example, Fig. 2 shows one

	Gate sequence											
Parent 1:	A	B	C	D	E	F	G	H	I	J	K
Parent 2:	C	F	I	B	K	G	A	D	E	H	J
Offspring 1:	A	B	C	D	E	G	A	D	E	H	J

Fig. 2 *Simple crossover and resulting conflicts*

possible nonbinary representation of the gate matrix problem, which consists just of the sequence of the gates.

The offspring inherits the gate sequence to the left of the crossover point from parent 1, and the sequence to the right from parent 2. Crossover between two different gate sequences would result in duplicated gates, as well as missing gates. Thus, simple crossover cannot be used to allow the offspring to inherit parts of the gate sequence from two different parents. A method of resolving these conflicts is needed.

Three new genetic crossover methods have been tried in Reference 12, which were adapted from the travelling salesman problem in Reference 13. They gave moderately good results. Other implementations of the GA are described in References 14 and 15 for macro cells, and in Reference 16 for standard cells.

Fig. 1 *Gate matrix layout model*

a Circuit
b Netlist and connectivity matrix
c Unoptimised layout
——— metal
——— poly
□ diffusion
d Optimised layout
↑ V_{dd}
↓ V_{ss}
• contact
p/n well
e Unoptimised symbolic representation
f Optimised symbolic representation

This paper describes a novel implementation of the GA for permutation problems, using beam search to reduce the search space. For each gate i, all other gates j connected to it are ranked in descending order of the number of connections between gates i and j. The GA picks one of the K best-ranked gates to place next to each gate. The chromosome coding consists of the ranks associated with the starting gate (the gate to be placed at the extreme left) and with each other gate.

In beam search, we consider only a small subset of the solution space. Only those permutations are considered which can be obtained by placing one of the K best-ranked gates next to each gate. As we have K choices of gates for each column (except at the end when fewer than K gates are left to assign), the subspace considered is approximately K^n instead of $n!$, and $K \ll n$, where n is the number of gates and K is the given beam value. In the following Sections, we will see what is an appropriate value for K.

3.1 Schemata

Holland [10] explains the working of the GA in terms of the schemata contained in a chromosome. Each bit in the chromosome (or each symbol, if the encoding is not binary) is called a gene, and conveys a certain property of the current chromosome, which effects its fitness function, and which can be inherited by other chromosomes. Groups of genes are called schemata. Thus a schema is a partially specified chromosome, with the rest of the gene values as don't cares. According to Holland's schema theorem, if a certain group of genes occurs in the members of the population with a high average fitness, it will be inherited by many other offspring, and will multiply. This is because the GA produces more offspring out of the fitter members of the population. At the same time, crossover will enable the mixing of such schemata, which will be tried out in the presence of other schemata in the same string, resulting in the discovery of larger and larger good schemata, giving, in this case, more complete gate matrix permutations.

Note that we never manipulate partial gate matrix permutations separately. All members of the population are complete permutations. They contain these partial permutations, or schemata implicitly, and evaluating one entire permutation results in the refining of our estimate of the fitness of all the partial permutations it contains. As an n-bit string consists of 2^n substrings or schemata, this represents a large amount of implicit parallelism in the operation of the algorithm.

Contiguous substrings of a gate permutation, as shown in the left and right halves of Fig. 2, are meaningful schemata, and a good contiguous substring from each parent may result in a good offspring, although there might be some disruption at the cut. However, we have already seen that passing contiguous substrings of a permutation results in conflicts.

The approach chosen in this research is to combine rank information from the two parents, as the ranks provide meaningful information that can be inherited from the parents, and which controls the fitness of the offspring.

3.2 Algorithm

The GENESIS package [17] has been used as the GA for this research. The evaluation algorithm for constructing permutations from the chromosomes is listed in Fig. 3. It is illustrated in Fig. 4 for the five-gate circuit of Fig. 1. The explanation of the algorithm will be easier in terms of this example.

The connectivity matrix is given in Fig. 4, which specifies how many nets connect each pair of gates, as determined simply by counting from the netlist. As part of the initialisation procedure, a sorted matrix is constructed, containing, for each gate, all other gates sorted in descending order of their connectivity.

The GA provides the chromosomes that define a gate matrix placement. Each chromosome includes a starting rank. If the rank for the starting gate is i, the ith least connected gate is picked as the starting gate, considering the total connectivity of all gates. In the example, the starting rank is 1, so Z, with total connectivity 3 is picked. This is based on the heuristic that the least-connected gates should preferably be on the edges. Hence, the GA gets a chance to pick one of K least-connected gates as the starting gate, where K is the beam value.

In addition, for each gate G, the chromosome gives the rank i of the next gate to be placed to its right. This rank is to be looked up in the sorted matrix in the row of G, to pick the ith most-connected gate to G. If this gate has already been placed, we have to search in both directions for the closest free rank in the row of G.

After selecting Z as the starting gate, the next step is to select the rank associated with Z in the chromosome, and pick the next gate to the right. In the example, the rank is 1, and from the sorted matrix, C is the most-connected gate to Z. The rank associated with C in the chromosome is three. From the sorted matrix, the third most-connected gate to C is Z. This is already placed. To resolve this conflict, the rank is approximated to the nearest available one. In this case, D (rank 4) is placed next to C. The rank associated with D in the chromosome is 1, so from the sorted matrix, B is the next choice. The rank associated with B in the chromosome is 2, so A is the closest available choice. Note that as an extra rank is specified in the chromosome for the starting gate, the rank for the ending gate is redundant, and is ignored. In this example, the rank specified for A was never used.

The probability of any requested gate being available is proportional to the ratio of the number of gates not yet placed to the total number of gates. At the beginning of the permutation construction, all gates are available, so there is a high probability that the ranks requested by the GA are available. This probability decreases as the permutation is contructed. On average, the probability of a requested gate being available is 50% throughout the permutation construction. The probability that the requested gate, or the next or previous ranked gate, being available is 87.5%, and there is only a 12.5% probability that we will have to search further up and down the sorted matrix.

Thus, the permutation is constructed semi-greedily by considering the individual gate-to-gate connectivities, and using the instructions from the GA. The objective of the GA is to obtain feedback from the fitness function for the entire permutation, and optimise the ranks so as to attain a near-global optimum in the fitness function.

The GA works by passing some meaningful information from the parents to the offspring in the form of schemata. The fitness of the offspring depends on this information. In this algorithm, the ranks provide this meaningful information that can be inherited from the parents. That is, if, after placing a gate A, placing the kth best-ranked gate next to A results in a high fitness in the

parent, passing that information to the child can result in a high fitness for the child.

After a permutation is constructed, it is evaluated. The span of each net is determined. The track count for all the gates included in the span of the net is incremented by one. The maximum track count at any gate gives the number of tracks required. The total netlength is the sum of these net spans. The fitness of the gate matrix layout is given by

$$f = 1/(\text{tracks} * \text{weight} + \text{netlength})$$

The default value of *weight* is the number of gates. This gives the number of tracks an appropriately high weightage, and finds a layout with the minimum number of tracks first, and then tries to minimise the netlength. Note that the number (*tracks* * *gates*) will always be slightly larger than the netlength. The weight can be changed to reflect the designer's preference.

A special mutation function was also added to the GENESIS package. The GA starts with random ranks for all gates, and reduces them in an attempt to place the most-connected gates next to each other. In general, it finished with small values for most of the ranks in the chromosome. It was discovered that the algorithm was performing poorly because the traditional mutation operator changed some of the ranks back to a random value (with a small probability). Therefore, a new mutation operator was provided, which randomly increased or decreased the ranks by 1. This is a more incremental change, and explores the space close to the current value of the ranks, while still maintaining the reachability of all rank values.

4 Results

The algorithm was tested on six netlists, ranging from 15 to 100 gates. The program was written in C and was run on a DECstation 3100.

The first experiment was conducted on a 60-gate circuit to optimise the parameters of the GA. The results

```
Procedure INIT
Input: Problem data
Output: Connectivity matrix, sorted matrix
Begin
  Read netlist;
  Construct a connectivity matrix, C[i][j] = No. of nets connecting gates i, j;
  For i = 1 to num_gates
  Begin
    Fill row i of sorted_matrix with all gates except gate i;
    Quick sort row i of sorted_matrix on descending order of connectivities from gate i;
  End
End.

Procedure BEAM_SEARCH
Input: Chromosome, connectivity matrix, sorted matrix
Output: Gate permutation, fitness
Begin
  Read the rank j for the starting gate from chromosome;
  Considering the total connectivity of each gate, choose the jth least-connected gate as the starting gate;
  current_gate = starting_gate;
  permutation[0] = starting_gate;

For i = 1 to num_gates - 1
Begin
  Mark current_gate as used;
  Read the rank j associated with the current_gate from the chromosome;
  Determine next_gate = jth-best gate from sorted_matrix;
  If next_gate is already marked used then
    Alternatively scan up and down the sorted matrix to determine the nearest available rank;
  permutation[i] = next_gate;
  current_gate = next_gate;
End
/* Evaluate Gate Permutation: */
netlength = 0;
weight = num_gates;   /* default */
For each net i
Begin
  Find the position of the left-most gate connected to net i;
  Find the position of the right-most gate connected to net i;
  Increment the track count for each gate from the left-most gate to the right-most gate;
  netlength = netlength + (distance between left-most and right-most gate);
End
num_tracks = max. of track counts of all gates;
fitness = 1/(num_tracks * weight + netlength);
End.

Procedure MUTATION
Input: Chromosome
Output: Mutated chromosome
Begin
  For all genes
    If rand(0, 1) < mutation_probability
    Begin
      gene = gene + rand_integer(-1, +1);
      If gene exceeds beam limits then reset gene value to limit;
    End
End.
```

Fig. 3 *Genetic beam search algorithm*

Connectivity matrix

	A	B	C	D	Z	total
A	0	4	3	1	1	9
B	4	0	3	2	0	9
C	3	3	0	1	2	9
D	1	2	1	0	0	4
Z	1	0	2	0	0	3

Sorted matrix

rank:	1	2	3	4
A:	B	C	D	Z
B:	A	C	D	Z
C:	A	B	Z	D
D:	B	A	C	Z
Z:	C	A	B	D

Chromosome

start	1
A	1
B	2
C	3
D	1
Z	1

Steps in decoding the chromosome

Step	Permutation	Gates used				
		A	B	C	D	Z
1	Z					•
2	Z C			•		•
3	Z C D			•	•	•
4	Z C D B		•	•	•	•
5	Z C D B A	•	•	•	•	•

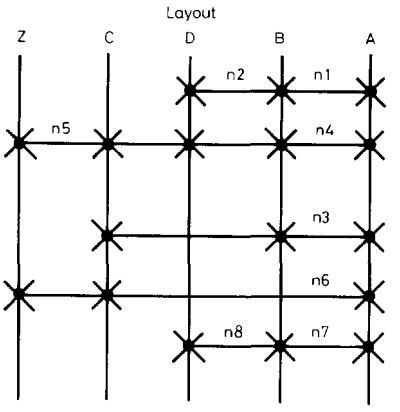


Fig. 4 *Example of chromosome decoding for the circuit of Fig. 1*

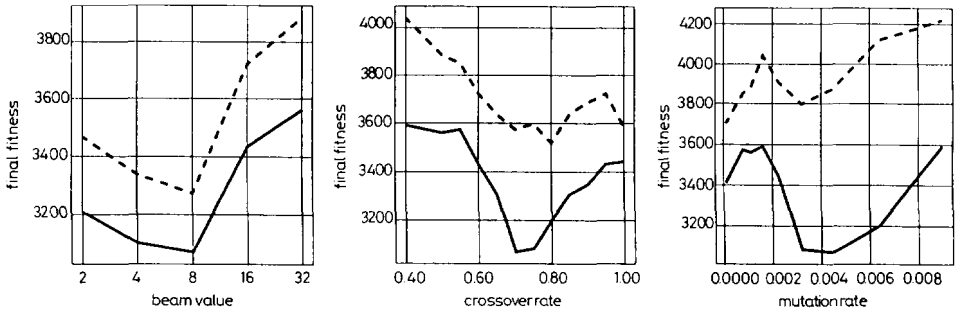


Fig. 5 *Effect of genetic parameters and beam value*
——— minimum - - - - average

are plotted in Fig. 5. The graphs show the final fitness for the 60-gate layout. The two curves in each graph are the minimum population, and average population at the end of each run. The results obtained for various beam values for the 60-gate circuit are also shown in this Figure. Similar experiments were performed for other circuits of different sizes. Our initial conjecture was that a larger beam value would be required for larger problems, in order to escape local optima. However, experiment showed that a small beam value of 8 gave the best or near-best results for all problems. The reason for this is that each gate is directly connected to at most a few other gates. Thus, for small i, the connectivity of any gate to the ith-best gate becomes zero. A look at the connectivity matrix shows that in most cases i is only a little larger than 8. Thus, for each gate, only the first i gates can be meaningfully ranked in descending order of their connectivity.

Fig. 5 also gives the results of varying the crossover and mutation rates. The mutation rate 0.005 and crossover rate 0.7 which gave the best results were selected. It was observed that if the population size is above 75, it does not affect the results significantly, and in a predictable manner. Therefore, an arbitrary population size of 100 was selected. Other parameters were set by default. Table 1 gives the GA parameters used with the GENESIS package.

Next, the algorithm was run with these parameters on six netlists. The results are shown in Table 2. Three of the netlists were taken from Reference 8 in which Hong *et al.* provide results for a simulated annealing technique as well as for a heuristic technique. The number of rows and the netlength for Hong's layouts as given in Table 2 were determined from the layouts published in the Appendix of Reference 8. In one case, we obtained one less track, and in all three cases, we obtained a better netlength.

The Table shows that the evaluation function used about 90% of the CPU time, and crossover, mutation etc. are relatively fast. This is to be expected in an algorithm

Table 1: Genetic algorithm parameters for GENESIS

Population size	100
Crossover rate	0.7
Mutation rate	0.005
Generation gap	1.0
Sigma scaling	3.0
Chromosome coding	binary
Beam value	8

Table 2: Experimental results

Gates	Nets	Hong's [8]		Genetic beam search				
		Rows	Net-length	Rows	Net-length	Evalu-ations	Total CPU time (s)	Eval. (s)
15	18	7	71	7	67	10000	11.9	9.9
29	37	13	255	13	227	100000	289.6	260.3
36	24	—	—	9	203	120000	325.9	286.3
48	48	13	411	12	365	250000	1393.2	1289.3
60	70	—	—	26	1149	150000	1314.0	1236.6
100	100	—	—	49	2957	170000	3649.6	3503.7

that does conflict resolution as part of the evaluation procedure, to derive a valid gate sequence from a chromosome that is a bit-string. The evaluation time shown in the Table includes this conflict-resolution time.

Fig. 6 shows the gate matrix layout for a 48-gate problem.

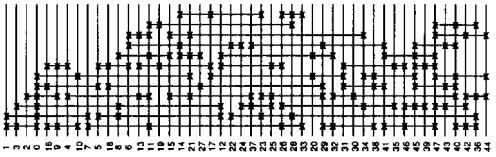


Fig. 6 *Layout for the 48-gate circuit of Hong et al.*

5 Conclusions

In this paper, we have presented a novel implementation of the GA for gate matrix layout. Gate matrix layout is an important tool for VLSI synthesis. This is a permutation problem, for which the traditional genetic crossover operator results in repetition of gates, and therefore the GA is not applicable without modification. However, the GA is a very efficient stochastic optimisation technique, which is adaptive, and easily parallelisable. To adapt it to the gate matrix layout problem, the principles of beam search have been used. The gates are sorted according to their connectivity to each other, and the ranks of the gates to be placed next to each other are given by the GA. A beam value is used to restrict the ranks of the gates placed next to each other to a low value. This reduces the search space to be explored, and thus results in a more efficient search.

The algorithm has been tried on six netlists, with up to 100 gates, including three benchmark circuits. The resulting track count and netlength were better than previously published results. The GENESIS package was used as the genetic algorithm of choice, along with appropriate evaluation functions to implement beam search. The values for the crossover rate, mutation rate and population size were fixed to give the best possible results with a wide variety of circuits.

6 References

1 LOPEZ, A.D., and LAW, H.F.S.: 'A dense gate matrix layout method for MOS VLSI', *IEEE Trans. Electron Devices*, 1980

2 WEINBERGER, A.: 'Large scale integration of MOS complex logic: a layout method', *IEEE J. Solid-State Circuits*, 1967

3 WING, O.: 'Interval-graph-based circuit layout'. Proceedings of the IEEE International Conference on Computer-Aided Design, 1983

4 WING, O., HUANG, S., and WANG, R.: 'Gate matrix layout', *IEEE Trans. Comput.-Aided Des., Integr. Circuits Syst.*, 1985, **CAD-4,** (3)

5 XU, D.M., CHEN, Y.K. *et al.*: 'A new algorithm for gate matrix layout'. Proceedings of the IEEE International Circuits and Systems Symposium, 1987, p. 288

6 HWANG, D.K., FUCHS, W.K., and KANG, S.M.: 'An efficient approach to gate matrix layout', *IEEE Trans. Comput.-Aided Des., Integr. Circuits Syst.*, 1987, **CAD-6,** (9)

7 HUANG, S., and WING, O.: 'Improved gate matrix layout', *IEEE Trans. Comput.-Aided Des., Integr. Circuits Syst.*, 1989, **CAD-8,** (8)

8 HONG, Y., PARK, K., and KIM, M.: 'A heuristic algorithm for ordering the columns in one-dimensional logic arrays', *IEEE Trans. Comput.-Aided Des., Integr. Circuits Syst.*, 1989, **CAD-8,** (5), pp. 547–562

9 LEONG, H.W.: 'A new algorithm for gate matrix layout'. Proceedings of the IEEE International Conference on Computer-Aided Design, 1986, pp. 316–319

10 HOLLAND, J.H.: 'Adaptation in natural and artificial systems' (University of Michigan Press, Ann Arbor, 1985)

11 MOHAN, S., and MAZUMDER, P.: 'A distributed genetic algorithm for standard cell placement on a network of workstations'. Technical Report, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, 1991

12 SHAHOOKAR, K., and MAZUMDER, P.: 'A genetic approach to standard cell placement using meta-genetic parameter optimization', *IEEE Trans. Comput.-Aided Des., Integr. Circuits Syst.*, 1990, **CAD-10,** (5)

13 OLIVER, I.M., SMITH, D.J., and HOLLAND, J.R.C.: 'A study of permutation crossover operators on the travelling salesman problem'. Proceedings of the International Conference on Genetic Algorithms and their Applications, 1985, pp. 224–230

14 CHAN, H., MAZUMDER, P., and SHAHOOKAR, K.: 'Macro-cell placement by genetic adaptive search with bitmap-represented chromosomes', *Integr. VLSI J.*, 1991, pp. 49–77

15 ESBENSEN, H.: 'A genetic algorithm for macro cell placement', *Proc. Euro-DAC*, 1992 (to appear)

16 COHOON, J.P., and PARIS, W.D.: 'Genetic placement'. Proceedings of the IEEE International Conference on Computer-Aided Design', 1986, pp. 422–425

17 GREFENSTETTE, J.J., and SCHRAUDOLPH, N.N.: 'A user's guide to GENESIS 1.2 ucsd'. CSE Department, University of California, San Diego, 1987

18 GOLDBERG, D.E.: 'Genetic algorithm in search, optimization and machine learning', 1989