A Genetic Approach to Standard Cell Placement Using Meta-Genetic Parameter Optimization

KHUSHRO SHAHOOKAR AND PINAKI MAZUMDER, MEMBER, IEEE

Abstract—**This paper describes the implementation of the Genetic Algorithm for Standard-cell Placement (GASP). Unlike the other placement algorithms that apply transformations on the physical layout, the genetic algorithm applies transformations on the chromosomal representation of the physical layout. The algorithm works on a set of configurations constituting a constant size population. The transformations are performed through crossover operators that generate a new configuration assimilating the characteristics of a pair of configurations existing in the current population (similar to biological reproduction). Mutation and inversion operators are also used to increase the diversity of the population, and avoid premature convergence at local optima. Due to the simultaneous optimization of a large population of configurations, there is a logical concurrency in the search of the solution space which makes the genetic algorithm an extremely efficient optimizer. Three efficient crossover techniques have been compared, and the algorithm parameters, namely mutation rate, crossover rate, and inversion rate have been optimized for the cell placement problem by using a meta-genetic process. The resulting algorithm was tested against TimberWolf 3.3 on five industrial circuits consisting of 100–800 cells. The results indicate that a placement comparable in quality can be obtained in about the same execution time as TimberWolf, but the genetic algorithm needs to explore 20–50 times less configurations compared to TimberWolf, which illustrates the efficiency of the search process.**

I. Introduction

As VLSI device integration is doubling every two to three years, layout complexity and chip turnaround time are becoming important design considerations. Standard cell design techniques are commonly used because they provide a semi-regular layout, which can be accomplished in a relatively short time. In such a layout, all cells have the same height, and their width varies with their functionality. The cells are placed in rows, and interconnects run in channels between the rows. One of the major objectives in standard cell layout is to minimize the overall interconnect length by placing the cells optimally with respect to each other.

Several deterministic algorithms have been proposed in the past, based on recursive partitioning [1], [8], [23], and physical laws which minimize the energy in the steady state such as Hooke's law [11] and power distributions in passive resistive networks [3], but the most successful methods so far are the ones that use a good general-purpose stochastic optimization algorithm, such as simulated annealing [5], [16], [18], [19], [21].

This paper describes the implementation and optimization of the Genetic Algorithm for Standard-cell Placement (GASP-1). As opposed to simulated annealing, which normally uses pairwise interchange for transforming the layout configuration, in a genetic algorithm, the *crossover operator* is used to combine two current configurations to generate a new configuration (similar to reproduction in biology). The traditional genetic crossover operator, as proposed by Holland [14], cannot be applied without modification to the cell placement problem, because it occasionally results in illegal placement. We have, therefore, directed a great deal of effort towards finding an efficient crossover operator for this problem domain.

We have implemented three powerful crossover operators and compared their performance in reducing the interconnect length. The results of this comparison were conclusively in favor of *Cycle crossover*. We have also analyzed the performance of these crossover operators, and demonstrated that these operators allow us to examine a large portion of the sample space. In this paper, the parameters of the genetic algorithm have been optimized using the meta-genetic technique, which itself is a genetic optimization process for tuning the parameters of the genetic algorithm. In order to benchmark the performance of GASP-1, we picked the best possible compromise between the parameters, and ran the algorithm to place five industrial circuits (obtained from IBM) consisting of 100–800 cells. The results were very encouraging. The total number of configurations examined by GASP-1 was 19–50 times less than that for TimberWolf 3.3 [21] and the runtime was marginally better. The percentage improvement in the wire length was better in three out of five circuits. The overall conclusion from this preliminary research is that adaptive search, similar to the one we used in this paper, can be *admissible* (i.e., it converges to near-global minimum) similar to the stochastic approaches (like the simulated annealing), but because it examines a large number of configurations concurrently, it searches far more efficiently than algorithms using pairwise exchange.

II. The Algorithm

The genetic algorithm has recently been applied to optimization problems in diverse fields, such as job scheduling [6], machine learning [9], the traveling salesman

Manuscript received April 4, 1989. This work was supported by URI-Program of the U.S. Army under Grant DAAL 03-87-K-0007 and by the Research Initiation Award of the National Science Foundation under Grant MIP-8808978. This paper was recommended by Associate Editor R. H. J. M. Otten.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 8934103.

problem [12], and pattern recognition [22], [24]. It is a powerful optimization algorithm, which starts with an initial set of random configurations and uses a process similar to biological evolution to improve upon them. The set of configurations the genetic algorithm operates upon is called the *population*. Each individual in the population is a string of symbols, usually but not necessarily, a binary bit string representing a solution to the optimization problem. We have used ordered triples of integers as symbols for the standard cell placement problem, as opposed to the bit map representation for macro-cell placement in [2]. The flowchart is given in Fig. 1. During each iteration, called a *generation*, the individuals in the current population are *evaluated*, using some measure of fitness. Based on this *fitness* value, individuals are selected from the population two at a time as parents. The fitter individuals have a higher probability of being selected. A number of genetic operators are applied to the parents to generate new individuals, called offspring, by combining the features of both parents. The three genetic operators commonly used by the genetic algorithm are crossover, mutation, and inversion, which are derived by analogy from the biological process of evolution. These operators are described in detail below. The offspring are next evaluated, and a new generation is formed by selecting some of the parents and offspring, and rejecting others so as to keep the population size constant.

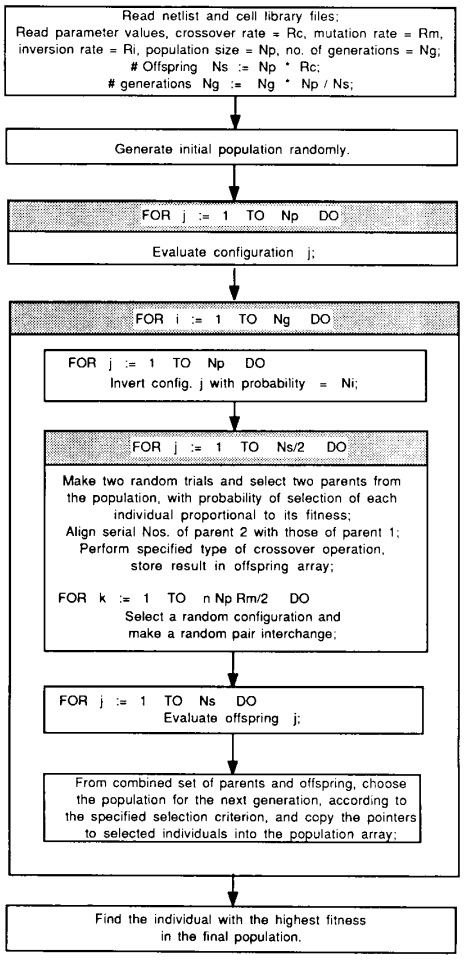


Fig. 1. The genetic algorithm.

2.1. Schemata

This section explains why genetic algorithms are so successful in complex optimization problems in terms of schemata and the effect of genetic operators on them. Informally, the symbols used in the solution strings are known as *genes*. They are the basic building blocks of a solution, and represent the properties which make one solution different from the other. For example, in the cell placement problem, the ordered triples consisting of the cells and their assigned coordinates can be considered as genes. A solution string, which is made up of genes, is called a *chromosome*. A *schema* is set of genes that make up a partial solution. An example would be a subplacement, consisting of any number of such triples, with "don't cares" for the rest of the cells. A schema with m defining elements, and "don't cares" in the rest of the $n - m$ positions (such as an m-cell subplacement in an n-cell placement problem) can be considered as an $(n - m)$-dimensional hyperplane in the solution space. All points on that hyperplane (i.e., all configurations that contain the given subplacement) are *instances* of the schema. Note here that the subplacement does not have to be physically contiguous, such as a rectangular patch of the chip area. For example, a good subplacement can consist of two densely connected cells in neighboring locations. Similarly, a good subplacement can also consist of a cell at the input end of the network and a cell at the output end which are currently placed at opposite ends of the chip. Both these subplacements will contribute to the high performance of the individual that inherits them. Thus a schema is a *logical* rather than physical grouping of cell-coordinate triples which have a particular relative orientation.

As mentioned above, the genetic operators create a new generation of configurations by combining the schemata (or subplacements) of parents selected from the current generation. Due to the stochastic selection process, the fitter parents, which are expected to contain some good subplacements are likely to produce more offspring, and the bad parents, which contain some bad subplacements are likely to produce less offspring. Thus in the next generation, the number of good subplacements (or high fitness schemata) tends to increase, and the number of bad subplacements (low fitness schemata) tends to decrease. Thus the fitness of the entire population improves. This is the basic mechanism of optimization by the genetic algorithm.

Each individual in the population is an instance of 2^n schemata, where n is the length of each individual string. Thus there are a very large number of schemata represented in a relatively small population. By trying out one new offspring, we get a rough estimate of the fitness of

all of its schemata or subplacements. Thus with each new configuration examined, the number of each of its 2^n schemata present in the population is adjusted according to its fitness. This effect is termed as the intrinsic parallelism of the genetic algorithm. As more configurations are tried out, the relative proportions of the various schemata in the population reflect their fitness more and more accurately. When a fitter schema is introduced in the population through one offspring, it is inherited by others in the succeeding generation, and thus its proportion in the population increases. It starts driving out the less fit schemata, and the average fitness of the population keeps improving.

2.2. *Genetic Operators*

The genetic operators and their significance can now be explained.

Crossover: Crossover is the main genetic operator. It operates on two individuals at a time, and generates an offspring by combining schemata from both parents. A simple way to achieve crossover would be to choose a random cut point, and generate the offspring by combining the segment of one parent to the left of the cut point with the segment of the other parent to the right of the cut point. This method works well with the bit string representation. Fig. 2 gives an example of crossover. In some applications, where the symbols in the solution string cannot be repeated, this method is not applicable without modification. Placement is a typical problem domain where such conflicts can occur [4]. For example, as shown in Fig. 2, cells *B* and *F* are repeated, and cells *H* and *I* are left out. Thus we need either a new crossover operator that works well for these problem domains, or a method to resolve such conflicts without causing significant degradation in the efficiency of the search process. The performance of the GA depends, to a great extent, on the performance of the crossover operator used. Three crossover operators are described in Section IV and their performance is evaluated for the cell placement problem.

When the algorithm has been running for some time, the individuals in the population are expected to be moderately good. Thus when the schemata from two such individuals come together, the resulting offspring can be even better, in which case they are accepted into the population. Besides, the fitter parents have a higher probability of generating offspring. This allows it to examine more configurations in a region of greater average fitness so that the optimum may be determined, and, at the same time, examine a few configurations in other regions of the configuration space so that other areas of high average performance may be discovered.

The amount of crossover is controlled by the *crossover rate*, which is defined as the ratio of the number of offspring produced in each generation to the population size. The crossover rate determines the ratio of the number of searches in regions of high average fitness to the number of searches in other regions. A higher crossover rate allows exploration of more of the solution space, and reduces the chances of settling for a false optimum; but if this rate is too high, it results in the wastage of a lot of computation time in exploring unpromising regions of the solution space.

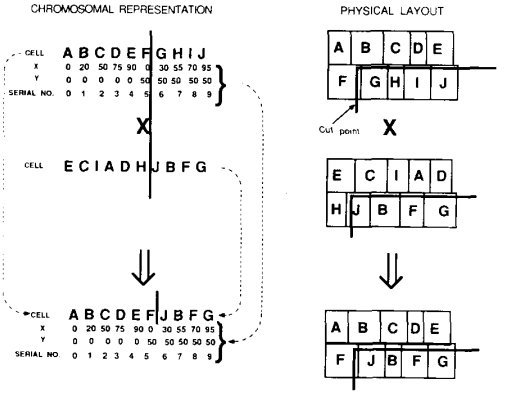


Fig. 2. The traditional method of crossover. A segment of cells is taken from each parent. The coordinate array is taken from the first parent. With this method, cells *B* and *F* are repeated, and cells *H* and *I* are left out.

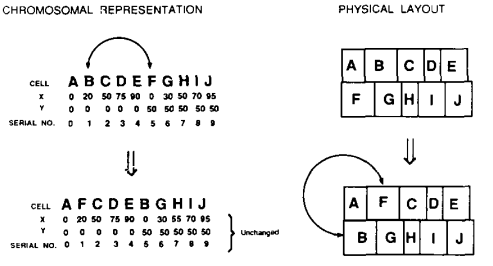


Fig. 3. Mutation.

Mutation: Mutation is a background operator, which produces spontaneous random changes in various individuals. The mechanism used in this implementation is pairwise interchange as shown in Fig. 3. This is not a mechanism for randomly examining new configurations as in other iterative improvement algorithms. In GAs, mutation serves the crucial role of replacing the genes lost from the population during the selection process so that they can be tried in a new context, or providing the genes that were not present in the initial population. In terms of the placement problem, a gene consisting of an ordered triple of a cell and its associated *ideal* coordinates may not be present in any of the individuals in the population. (That is, that particular cell may be associated with nonideal coordinates in all the individuals.) In that case crossover alone will not help, because it is only an inheritance mechanism for existing genes. The mutation operator generates new cell-coordinate triples. If the new triples perform well, the configurations containing them are retained, and these triples spread throughout the population by crossover.

The *mutation rate* is defined as the percentage of the total number of genes in the population, which are mu-

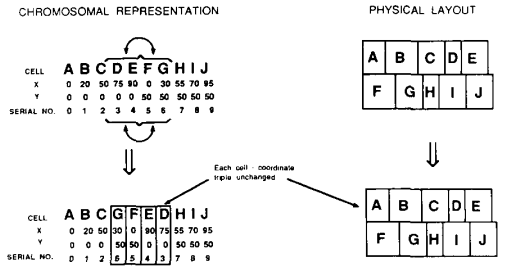


Fig. 4. Inversion.

tated in each generation. Thus for an n-cell placement problem, with a population size N_p, the total number of genes is nN_p, and $nN_pR_m/2$ pairwise interchanges are performed for a mutation rate R_m. The mutation rate controls the rate at which new genes are introduced into the population for trial. If it is too low, many genes that would have been useful are never tried out. If it is too high, there will be much random perturbation, the offspring will start losing their resemblance to the parents, and the algorithm will lose the ability to learn from the history of the search.

Inversion: This consists of taking a random segment in a solution string and flipping it (Fig. 4). This operation should be performed in a way such that it does not modify the solution represented by the string; instead, it only modifies the representation of the solution. Thus the symbols composing the string must have an interpretation independent of their position. This can be achieved by associating a serial number with each symbol in the string, and interpreting the string with respect to these serial numbers instead of the array index. When a symbol is moved in the array, its serial number is moved with it, and so the interpretation of the symbol remains unchanged. In the cell placement problem, the x- and y-coordinates stored with each cell perform this function. Thus wherever the cell-coordinate triple is located in the population array, it will have the same interpretation in terms of the physical layout.

The advantage of the inversion operator is the following. There are some groups of properties, or genes which would be advantageous for the offspring to inherit together from one parent. Such groups of genes, which interact to increase the fitness of the offspring that inherit them are said to be *co-adapted*. For example, if cells A and B are densely connected to each other, and parent 1 has the genes (A, x_1, y_1), and (B, x_2, y_2), where (x_1, y_1), and (x_2, y_2) are neighboring locations, then it would be advantageous for the offspring to inherit both these genes from one parent so that after crossover cells A and B remain in neighboring locations. If two genes are close to each other in the solution string, then they have a lesser probability of being split up when the crossover operator divides the string into two segments. Thus by shuffling the cells around in the solution string, inversion allows triples of cells which are already well-placed relative to each other, to be located close to each other in the string. This increases the probability that when the crossover operator splits parent configurations into segments to pass to the offspring, the subplacements consisting of such groups will be passed intact from one parent (or another). This process allows for the formation and survival of highly optimized subplacements long before the optimization of any complete placement is finished.

III. Implementation Details

The flowchart for GASP-1 is given in Fig. 1. First, an initial population is constructed *randomly*. Each individual is represented by a set of four integer arrays containing the cell number, the x- and y-coordinates, and a serial number. Internally assigned cell numbers are used instead of cell names for compact storage and faster processing. The serial number is used to keep track of the approximate slot in the physical layout area to which each cell is assigned. It is somewhat redundant when the x- and y-coordinates are given, but it greatly speeds up processing during crossover, and evaluation. For each configuration in the initial population, the coordinates of the cells are determined by placing them end-to-end in rows. The population size is provided by the user, and determines the tradeoff between processing time and result quality. From a lot of experimental data, we have observed that a constant population size of 24 gives the best possible solution quality for most of the circuits. Each individual is evaluated to determine its fitness. The fitness is the reciprocal of the bounded rectangle wire length. Since cell overlaps are removed before evaluation as described below, the traditional terms for row length control and overlap are unnecessary:

$$f = \frac{1}{\sum_{\text{nets}} x(i)\, W_H(i) + y(i)\, W_V(i)}$$

where f is the fitness, $x(i)$ and $y(i)$ are the horizontal and vertical spans of net i, and $W_H(i)$ and $W_V(i)$ are the horizontal and vertical weights.

At the beginning of each generation, *inversion* is performed on each individual, with a probability equal to the *inversion rate*. For this purpose, two cut points are determined randomly, and the segment between them in the cell array is flipped, along with the coordinates and the serial numbers (Fig. 3). This ensures that each cell is still associated with the same coordinates. The configuration is not changed; only its representation is changed in the population array. Then crossover takes place. Two individuals are selected from the population at random, with a probability proportional to their fitness. Before crossover, the serial numbers of the second parent are aligned in the same sequence as those of the first parent, so that cells in the same array locations correspond to approximately the same locations on the chip. Then segments are exchanged between parents as described in Section IV, so that for each location on the chip, the child inherits a cell from one parent or another. The coordinates and serial numbers are inherited entirely from one parent, which means that all the cells will not fit their slots and there

will be some overlap. This process is repeated until the desired number of offspring have been generated. The number of offspring per generation, N_s is determined by the *crossover rate*:

$$N_s = N_p R_c$$

where N_p is the population size, and R_c is the crossover rate. Since the number of configurations examined is kept constant in all runs for a particular circuit, the actual number of generations is increased as the crossover rate is reduced:

$$N_g = N_{g0} N_p / N_s$$

where N_p is the population size, and N_{g0} is the number of generations specified by the user. After crossover, each offspring is mutated with a probability equal to the *mutation rate*. For this purpose, a configuration is picked at random $nN_pR_m/2$ times as described above. Each time two cells are picked randomly and exchanged, leaving the coordinate arrays unchanged (Fig. 2). Thus mutation does result in a minor change in the physical configuration.

After crossover and mutation, the fitness of each offspring is evaluated. After a lot of experimenting, it was found beneficial to realign the cells in the offspring to remove cell overlap before evaluation. Unlike simulated annealing, where only two cells are moved at a time, and it is possible to calculate the wire length incrementally, in the genetic algorithm, as many as half the cells in the circuit are moved simultaneously, so the wire length must be computed exhaustively. Thus it is not necessary to tolerate overlap and preserve the coordinates of most of the cells as in simulated annealing. Besides, it was observed that determining the overlap takes about as much computation time as removing it. Further it was also observed that using overlap penalty and removing overlap at the end of the run resulted in a significant increase in wire length, as much as 10% in some cases, with the genetic algorithm receiving no further chance to work on the circuit. Removing the overlaps after every generation not only gives the algorithm a more accurate picture of the wire length, but also gives the algorithm repeated chances to optimize the circuit after it has been perturbed (from the state inherited from the parents) by overlap removal.

After evaluation, the population for the next generation is selected from the combined set of parents and offspring. Three selection methods have been tried—deterministic selection, in which the fittest individuals are selected, random selection, and random selection with the retention of the best individual. This completes the processing of one generation.

IV. Crossover Operators

As mentioned earlier, crossover is the primary method of optimization in the genetic algorithm, and, in the case of placement, works by combining subplacements from two different parent configurations to generate a new placement. In order to deal with the conflicts that can occur in traditional crossover, one must either find a way to combine two different configurations without conflicts, or use some method to resolve the conflicts that arise. We have compared the performance of three powerful crossover operators. Two of them, Order and PMX, differ in their conflict resolution methods, while Cycle crossover is a conflictless operator.

Order Crossover: The algorithm is as follows. Choose a cut point at random. Copy the array segment to the left of the cut point from one parent to the offspring. Fill the remaining (right) portion of the offspring array by going through the second parent, from the beginning to the end and taking those elements which were left out, in order. An example is shown in Fig. 5(a). This operator conveys a subplacement from the first parent without any changes, and then, to resolve conflicts, compresses the second parent by eliminating the cells conveyed by the first parent, and shifting the rest of the cells to the left, without changing their order [7]. It then copies this compressed second parent into the remaining part of the offspring array.

PMX: PMX [10] stands for "partially mapped crossover." It is implemented as follows. Choose a random cut point and consider the segments following the cut point in both parents as a partial mapping of the cells to be exchanged in the first parent to generate the offspring. Take corresponding cells from the segments of both parents, locate both these cells in the first parent and exchange them. Repeat this process for all cells in the segment. Thus a cell in the segment of the first parent and a cell in the same location in the second parent will define which cells in the first parent have to be exchanged to generate the offspring. An example is shown in Fig. 5(b).

Cycle Crossover: Cycle crossover [20] is an attempt to eliminate the cell conflicts that normally arise in crossover operators. In the offspring generated by Cycle crossover, every cell is in the same location as in one parent or the other. For Cycle crossover, we start with the cell in location 1 of parent 1 (or any other reference point), and copy it to location 1 of the offspring. Now consider what will happen to the cell in location 1 of parent 2. The offspring cannot inherit this cell from parent 2, since location 1 in the offspring has been filled. So this cell must be searched in parent 1 and passed on to the offspring from there. Supposing this cell is located in parent 1 at location x. Then it is passed to the offspring at location x. However, then the cell at location x in parent 2 cannot be passed to the offspring, so that cell is also passed from parent 1. This process continues until we complete a cycle and reach a cell that has already been passed. Then we choose a cell from parent 2 to pass to the offspring, and go through another cycle, passing cells from parent 2 to the offspring. Thus in alternate cycles, the offspring inherits cells from alternate parents, and the cells are placed in the same locations as they were in the parents from which they were inherited. An example is given in Fig. 5(c). One disadvantage of the Cycle operator is that the subsections passed from each parent are not contiguous in the cell array.

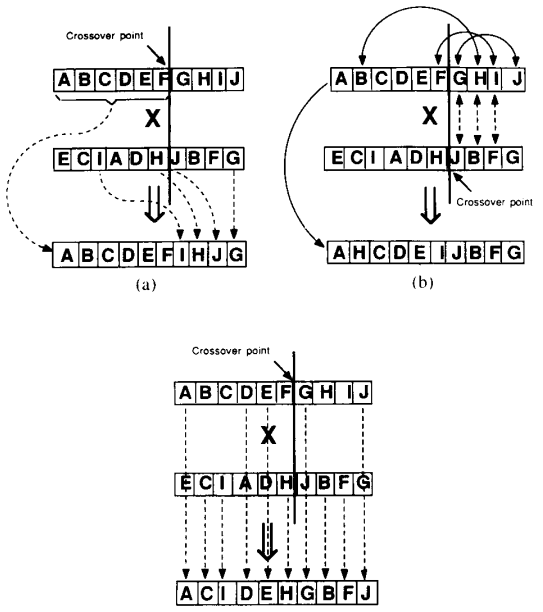


Fig. 5. Crossover operators. (a) *Order crossover:* Pass the left segment from parent 1. Construct the right segment by taking the remaining cells from parent 2 in the same order. (b) *PMX crossover:* The right segments of both parents act as a partial mapping of pairwise exchanges to be performed on parent 1. Since the pairs (G, J), (H, B), and (I, F) are situated at the same locations in both parents, exchange these cells in parent 1 to generate the offspring. (c) *Cycle crossover:* Start by passing A from parent 1 to the offspring. Since E is located at the same position in parent 2, it cannot be passed from there. D is located in the same position in parent 2, as E in parent 1. So proceed similarly with D. Now A is in the same location, but it has already been processed. This completes a cycle. Start another cycle from parent 2, by passing C to the offspring, and continue by passing B, H, F, and I from parent 2. The third cycle will again be from parent 1, and will pass G and J.

The survival probability for the schemata for each crossover operator is given in Appendix I, which indicates which operator is expected to be better under various situations.

V. Optimizing the Genetic Algorithm

We have used the meta-genetic algorithm [13] to optimize our genetic algorithm for cell placement. The three parameters optimized are the crossover rate, inversion rate, and mutation rate. The meta-genetic algorithm is itself a genetic optimization process, which runs the genetic algorithm to solve a placement problem, and manipulates its parameters to optimize its fitness (Fig. 6). The individuals in the population of the meta-genetic algorithm consist of three integers in the range [0, 20], representing the mutation rate, inversion rate, and crossover rate for the genetic algorithm. The mutation rate as given by this parameter can vary from 0 to 10% in steps of 0.5%, the inversion rate can vary from 0 to 100% in steps of 5%, and the crossover rate can vary from 20% to 100%, in steps of 4%. The fitness of an individual (a genetic algorithm with a certain parameter combination) is taken to be the fitness of the best placement that the genetic

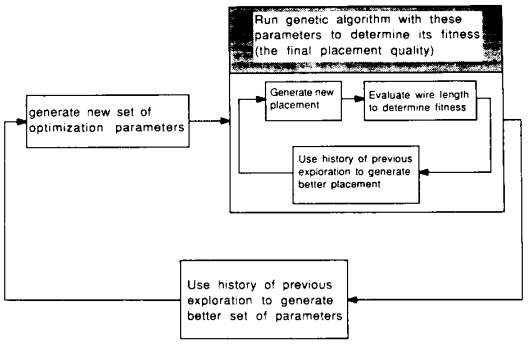


Fig. 6. The meta-genetic optimization process.

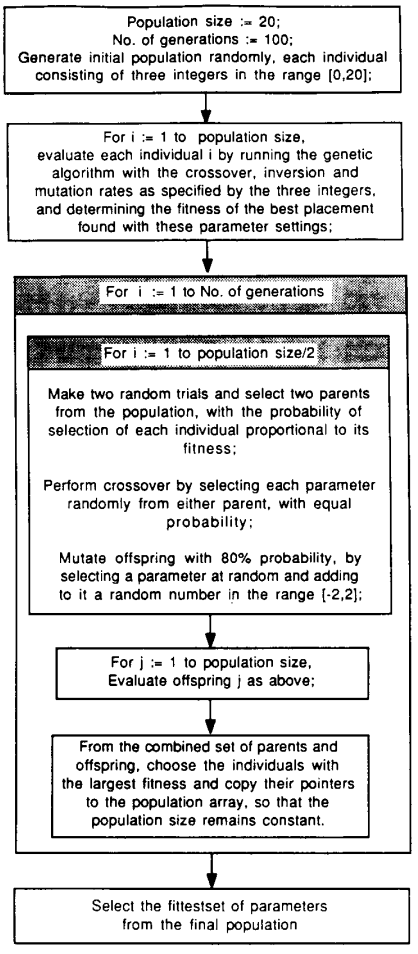


Fig. 7. The meta-genetic algorithm—flowchart.

algorithm can find in the entire run, using these parameters.

The flowchart is given in Fig. 7. For the meta-genetic algorithm, the population size was 20, and the algorithm was run for 100 generations. The crossover probability was 100%. Crossover consists of selecting each of the three parameters randomly from one parent or the other,

with equal probability of a parameter value being selected from either parent. Inversion is not used. This is because the length of each individual (three elements) is so small that there is little scope for group formation. After crossover, the offspring are mutated. Mutation consists of adding a random number in the range $[-2, 2]$ to any one parameter in the offspring. The probability of mutation is 20%. The high mutation rate and large number of generations are used so that the performance of the meta-genetic algorithm does not become a bottleneck in the search for the optimum parameters for the genetic algorithm.

The different crossover mechanisms were not included in the optimization. Instead, the optimization was performed separately for each crossover mechanism, and the best results for each were compared to determine an overall winner. This was done because of the following reason. Each crossover mechanism was expected to perform best with a slightly different combination of parameters. If the crossover type was included in the meta-genetic search, then the meta-genetic population would consist of individuals with different crossover types, each with its own optimized set of parameters. Unless the meta-genetic crossover operator was constrained to perform crossover between individuals of the same crossover type, it would tend to couple the wrong set of parameters with the wrong crossover type to generate offspring of poor quality. Thus we included only the three rate parameters in the meta-genetic optimization process and let them be optimized for a particular crossover type in each run.

VI. Comparison with Other Algorithms

6.1. Simulated Annealing

Both simulated annealing and the genetic algorithm are computation intensive. However, the genetic algorithm has some built-in features, which, if exploited properly, can result in significant improvement in runtime. One difference is that the simulated annealing algorithm operates on only one configuration at a time, while the genetic algorithm maintains a large population of configurations, which are optimized simultaneously. The population serves as a memory for experience gained in the search of the solution space. The genetic algorithm takes full advantage of this memory, and directs a more extensive search to areas of lower average cost, as determined by the search done so far. Since simulated annealing operates on only one configuration at a time, it has very little history to utilize, in order to be able to learn from past trials.

Both simulated annealing and the genetic algorithm have mechanisms for avoiding preconvergence to local optima. In simulated annealing, this is done by occasionally discarding a superior configuration and accepting an inferior one. The genetic algorithm also relies on inferior configurations as a means of avoiding false optima, but, since it has a whole population of configurations, it can keep and process inferior configurations without compromising the best ones. Besides, in the genetic algorithm, each new configuration is constructed from two previous configurations, which means that in a few iterations, all the configurations in the population have a chance of contributing their good features to form one super-configuration. In simulated annealing, each new configuration is formed from only one old configuration, which means that only incremental improvement is possible, and the good features of more than one radically different configurations never mix. A configuration is either accepted or thrown away as a whole, depending on its total cost.

On the negative side, the genetic algorithm requires more memory space compared to simulated annealing. For example, a 1000 cell placement problem would require up to 500 Kbytes to store a population of 24 configurations and 8 offspring (which includes the cell, coordinate and serial number arrays, but excludes the netlist and cell library information). For moderate size layout problems, this memory requirement may not pose any significant problem, because commercial workstations have 4M bytes or more of primary memory. For circuits of the order of 10 000 cells, the genetic algorithm is expected to have a small amount of extra paging overhead compared to simulated annealing, but it is still expected to speedup the optimization due to the efficiency of the search process.

6.2. Genie

Genie [4] is a genetic algorithm for standard cell placement, and follows the same fundamental techniques, although there are some important differences. Genie uses crossover and mutation operators which are different from those in GASP-1. Besides, the crossover operators in Genie pass on average only four to nine cells from one parent to the offspring, and the rest of the cells are passed from the other parent. Thus in placement problems of the order of 1000 cells, the crossover operators are unable to properly convey the features of both the parents to the offspring. In GASP-1, there is an equal probability of splitting the parents at any point along their length, which means that on average, both parents will contribute equally to the offspring.

Genie does not use the inversion operator, which is a very useful operator in genetic algorithms, as discussed in Section II. The scoring function in Genie considers the wire length, and wiring density. For the latter, a scheme is used, by which all channels with wiring density more than one standard deviation above the average are penalized. This is a very useful feature and would lead to higher quality layouts. However, we did not implement such a scheme because we believed it would be too expensive, computationally, and the benefits would not justify the added overhead. This issue is still open for further research.

6.3. ESP

Kling [17] has devised a novel evolution-based algorithm for placement. His approach is completely different from the traditional genetic algorithm as proposed by Holland [14]. ESP operates on only one configuration, like simulated annealing, and, naturally, generates offspring

from only one parent. There is some concurrency in the ESP search process, since several *badly placed* cells are removed from the current configuration and reallocated simultaneously. A number of heuristic measures used to determine the *goodness value* of placement of each cell. ESP has achieved impressive results compared to TimberWolf.

VII. Experimental Results

7.1. Meta-Genetic Parameter Optimization

Table I shows the statistics of the circuits used for the experiment. These circuits were obtained from IBM, and have been used by other researchers to evaluate their algorithms [15], [17]. A comprehensive experiment was performed to optimize the parameters of GASP-1, and determine which crossover and selection method performs the best. The experimental method was as follows. At first we ran the meta-genetic optimization algorithm on four test circuits consisting of 72, 100, and 183 cells, respectively. The results of this experiment are reported in Table II(a). This table shows the final optimized values of crossover rate, inversion rate, and mutation rate, the initial and final fitness of the corresponding genetic algorithm, and the convergence characteristics of the meta-genetic algorithm. Table II(b) shows the ranking of the various crossover and selection strategies according to the final fitness values. The overall winner in this comparison was Cycle crossover, with deterministic selection of the best configurations.

The optimum values of the genetic parameters given in Table II shows a large variation. However, these results provide a valuable insight into the acceptable range of parameter values. In all cases, the crossover rate was in the range 20–40%. The final population of the meta-genetic algorithm consisted of individuals with several different crossover rates, all with equal fitness. This indicated that the genetic algorithm is not very sensitive to slight variations in crossover rate, as long as a rate in the above range is used. For the mutation rate, in most cases, a very low value of 0.5–1.5% proved optimum, 1.5% being preferred only for the smallest circuit. The inversion rate fluctuated the most, but a value of 0–30% performed the best. We have adopted a compromise set of values that perform reasonably well with most of the circuits tested. These values are: crossover rate 33%, inversion rate 15%, and mutation rate 0.5%.

Table II also shows that the meta-genetic optimization runs converged in 20–30 generations in most cases. This is because of the small search space. There were only 8000 possible combinations of the three parameters to be searched.

Crossover: In most cases, either PMX or Cycle crossovers performed the best, and Order crossover performed the worst. Cycle crossover was found to be slightly better than PMX. As mentioned above, Cycle crossover does not suffer from any conflicts, and all the cells in the child are in the same locations as in one parent or another. This might be a factor in its better performance. Besides, PMX crossover results in less disruption of the placement conveyed from the parents as compared to Order crossover.

Selection: In all cases, deterministic selection of the best of the parents and offspring to be included in the next generation proved to be better than all other strategies. Fig. 8(a)–(e) show the plots of the lowest and average wiring cost in each generation as the optimization proceeds. Fig. 8(a)–(c) is a comparison of the crossover operators with deterministic selection, and Fig. 8(c)–(e) is a comparison of the selection methods used. The reason for the poor performance of the random selection methods can be clearly seen. Just as it is possible to combine the good features of two parents to form a better offspring, it is also possible to combine the bad features to form a far worse offspring. If these offspring are accepted on a random basis, the best and average cost in the population will oscillate as seen in Fig. 8(d). The losses involved in the random process far outweigh any advantage gained, and the algorithm takes a much longer time to converge. When we allow for the retention of the best solution along with random selection, the cost of the best solution is seen to decrease monotonically.

TABLE I
Test Circuits

CELLS	NETS	PINS
100	213	500
183	254	736
469	495	2189
750	1156	3062
800	843	2935

TABLE II
(a) Results of the Meta-Genetic Optimization Process (b) Ranking Based on Final Fitness

Cells	Selection procedure	Crossover operator	Crossover rate	Inversion rate	Mutation rate	Initial fitness	Final fitness	Convergence of Meta-genetic algorithm (gen)
72	Det	Order	20 - 44	10	1.0	1843	1877	18
	Det	PMX	20 - 44	70	1.5	1961	2048	24
	Det	Cycle	24 - 44	15	1.5	1951	2109	30
	R + B	Order	20 - 44	35	0.5	1652	1784	70
	R + B	PMX	24 - 44	15	0.0	1755	1779	22
	R + B	Cycle	20 - 44	65	0.5	1751	1880	57
100	Det	Order	18 - 56	85	0.5	1089	1089	49
	Det	PMX	20 - 28	30	1.0	1628	4770	42
	Det	Cycle	24 - 44	20	0.5	4632	4789	50
	Random	Order	32 - 44	20	1.0	1981	2086	29
	Random	PMX	32 - 48	30	1.0	1996	2100	32
	Random	Cycle	32 - 48	50	1.0	1995	2109	16
	R + B	Order	24 - 44	30	1.0	3490	3495	21
	R + B	PMX	20 - 28	25	0.5	3570	3711	25
	R + B	Cycle	32 - 44	55	0.5	3750	3882	28
183	Det	Order	32 - 40	0	0.5	917	957	23
	Det	PMX	24 - 44	0	0.5	958	1101	36
	Det	Cycle	44 - 48	20	0.5	1042	1137	35
	Random	Order	72 - 80	10	0.0	477	518	17
	Random	PMX	24 - 36	80	0.0	463	607	86
	Random	Cycle	68 - 80	5	0.5	695	761	16
	R + B	Order	20 - 40	10	0.0	827	839	19
	R + B	PMX	24 - 40	85	0.0	888	922	14
	R + B	Cycle	28 - 44	20	0.5	883	944	83

(a)

Rank	1	2	3	4	5	6
Cells						
72	Cycle, Det	PMX, Det	Cycle, R+B	Order, Det	PMX, R+B	Order, R+B
100	Cycle, Det	PMX, Det	Order, Det	Cycle, R+B	PMX, R+B	Order, R+B
183	Cycle, Det	PMX, Det	Order, Det	Cycle, R+B	PMX, R+B	Order, R+B

(b)

Det = Deterministic Selection
Random = Random selection
R+B = Random selection with retention of best string

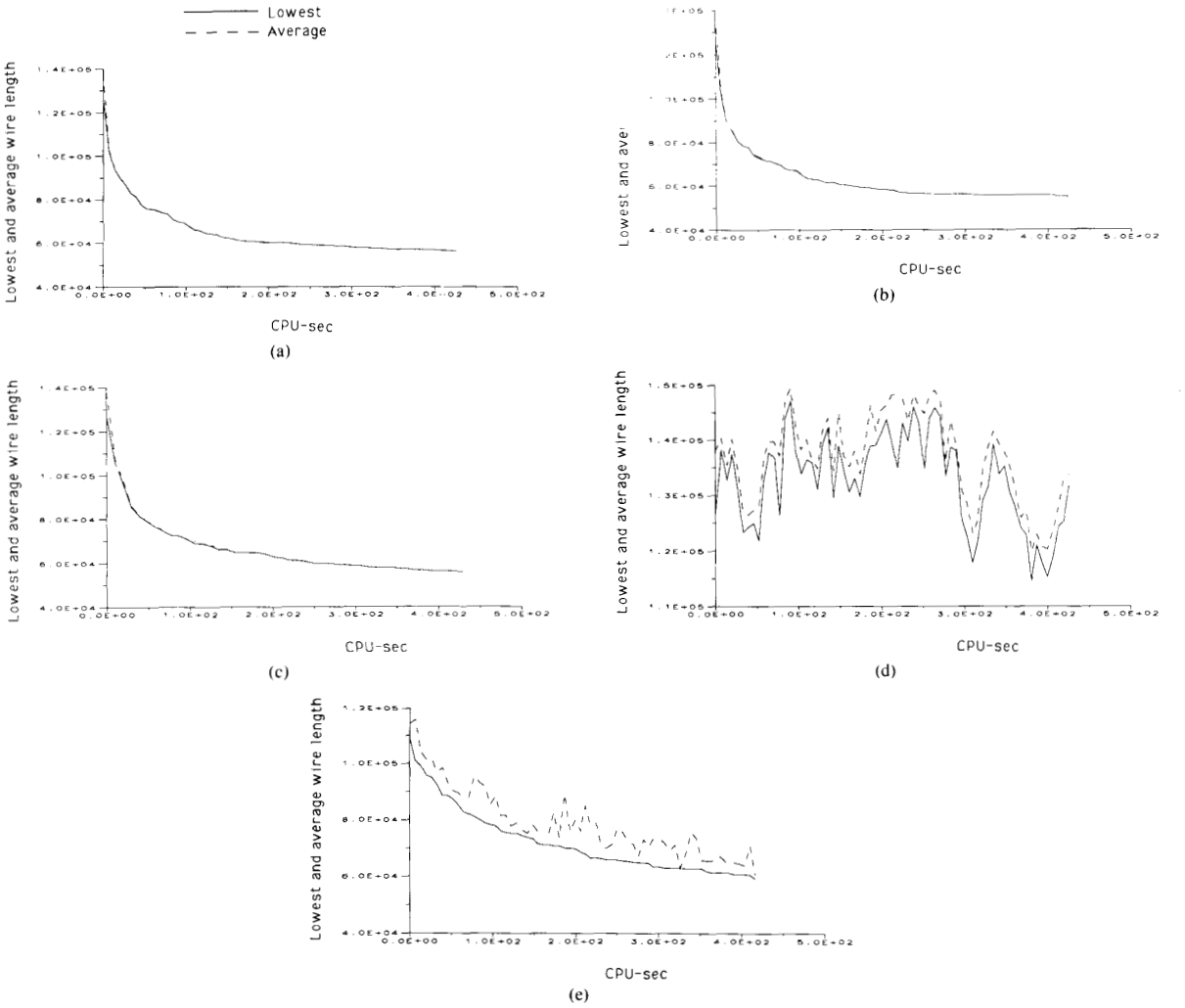


Fig. 8. The optimization characteristics of various crossover operators and selection functions. (a) Order crossover, deterministic selection. (b) PMX crossover, deterministic selection. (c) Cycle crossover, deterministic selection. (d) Cycle crossover, random selection. (e) Cycle crossover, random and best selection.

7.2. *Performance of GASP-1*

The performance of the algorithm was compared with TimberWolf 3.3 for five circuits ranging from 100 to 800 cells. The algorithm used for this purpose was Cycle crossover, with deterministic selection of the best configurations in each generation, and the crossover, mutation, and inversion rates given previously. TimberWolf was run with the attempts per cell parameter set to zero, which allows the algorithm to look up a table for the optimum number of iterations to perform. The genetic algorithm was run until no significant improvement was obtained. The results of this comparison are shown in Table III. The performance measure of the algorithm is taken as the percentage improvement in the wire length starting from a random initial configuration. For determining the percentage improvement in the genetic algorithm, the best individuals in the initial (random) and the final population are considered.

TABLE III
COMPARISON OF GASP-1 AND TIMBERWOLF 3.3

	TimberWolf 3.3				GASP-1					
Cells	Configs. examined	CPU sec	Δ	Total Memory	Generations	Configs. examined	CPU sec	Δ	M_p	Total Memory
100	1.17M	2102	61.3	384K	7900	62.2K	928	63.1	50K	212K
183	2.14M	2986	58.6	446K	14700	117K	2927	58.8	92K	427K
469	10.97M	40700	69.4	659K	24000	192K	39664	69.9	235K	623K
750	26.33M	44650	70.4	871K	55500	444K	40806	62.0	375K	890K
800	28.08M	49405	64.2	833K	64400	515K	44930	59.0	400K	911K

Δ = Percentage improvement in wire length,
M_p = Estimated memory required to store the population and offspring.

It can be observed from this table that GASP-1 examines 9–50 times less configurations compared to

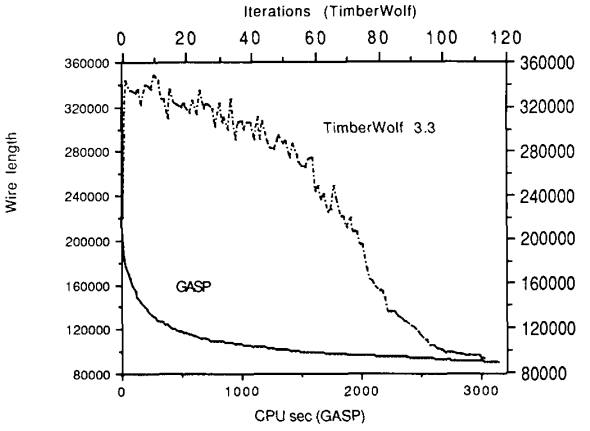


Fig. 9. The optimization characteristics of GASP compared to TimberWolf.

TABLE IV
DISTRIBUTION OF TIME IN CPU-SECONDS FOR GASP-1

Cells	Total	Crossover	Inversion	Mutation	Wire length evaluation	Overlap removal	Total evaluation	Selection
100	928	161	16	7	625	95	720	17
183	2927	649	37	16	1839	351	2190	32
469	39664	8349	372	128	26532	3050	30582	145
800	44930	10674	388	124	28570	5023	33593	80

TimberWolf for achieving the same or better percentage improvement of wire length. This illustrates the search efficiency of the genetic algorithm. The runtime, however, is only marginally less than TimberWolf in all cases. This is due to the increased overhead of wire length evaluation.

The result quality, is better in GASP-1 for three circuits, and in TimberWolf for two circuits. Thus we can conclude that the performance of this algorithm is comparable to TimberWolf 3.3 both in runtime as well as quality of placement. The memory requirements are also comparable, but this comparison is unfair for TimberWolf, which is a more well-developed package, and stores much more data on the physical structure of the layout. The column M_p of Table III gives the amount of memory required in GASP-1 to store the population and the offspring of one generation, or in other words, the overhead that the genetic algorithm has over simulated annealing.

Table IV shows the distribution of CPU time between the various functions of GASP-1. From this distribution it is seen that wire length evaluation is the bottleneck in the execution time of the algorithm, and takes 62–67% of the total time. In comparison, crossover takes only 17–24% of the total time. Thus by reducing the evaluation time, we will obtain a greater speedup. Overlap removal takes a very small amount of time as predicted in Section III.

Fig. 9 shows a comparison between the rate of optimization of GASP and TimberWolf. The 183 cell circuit, for which we obtained almost identical performance and runtime for both algorithms was used for the comparison. It was noted that in the first few high temperature iterations of TimberWolf, the wire length (and the penalties) *increased* significantly over the initial placement. This increase causes a greatly increased runtime, but is necessary for avoiding local optima. Besides, in almost the first half of the runtime, there is no significant improvement in wire length. In GASP-1, the wire length decreases monotonically, and the greatest reduction is at the beginning. Thus a moderate quality placement can be obtained very quickly. The same run can then be continued for a higher quality placement.

VIII. CONCLUSIONS

The genetic algorithm that was originally invented by Holland for adaptive searching in AI [14], has been employed in this research for the placement of standard cells. Unlike simulated annealing that uses pairwise exchange for evolving a new configuration from the current one, the genetic algorithm uses three powerful operators to guide its search through the solution space concurrently, by considering a set of configurations at a time. The placement problem is represented in the form of a genetic code, which is progressively refined, and improved by the operators. This is a major deviation from the conventional placement algorithms that directly apply the transformations to the physical layout. However, this feature of the genetic algorithm is also a potential problem, and unless clever representation is devised, the algorithm may prove inefficient. In this research, the standard cell placement problem has been represented as an effective genetic code, and three powerful crossover operators have been applied to generate new configurations. Two other genetic operators—inversion and mutation—have also been applied to obtain a highly efficient placement algorithm which requires fewer iterations to converge to a high quality placement. This has been made possible by applying a metagenetic process which optimizes the controlling parameters of the genetic algorithm. The algorithm has been compared with TimberWolf 3.3 for five different circuits with 100 to 800 cells, and it was observed that for similar quality of final placement the genetic algorithm described here needs to examine about 19–50 times less configurations as compared to TimberWolf 3.3. The runtimes of both algorithms are also comparable.

We are further optimizing the C code of GASP-1 to improve the runtime. We are also experimenting on ways to reduce the time required for the evaluation of new configurations, which is the bottleneck in GASP-1. At the same time we are also experimenting with ways to further improve the search efficiency of the genetic algorithm by improving the algorithm used for crossover, and by varying crossover and mutation rates during the optimization. With these improvements we expect to speedup the algorithm considerably.

We are also developing an efficient genetic representation of the standard cell routing problem. The overall objective will be to develop theoretical understanding of genetic algorithms for CAD applications and specifically to design a highly efficient tool for standard cell placement and routing.

Appendix I
Schema Survival Probabilities for Crossover Operators

The following analysis gives the survival probability of the schemata under the various crossover operators [20]. A higher survival probability implies a better transfer of desirable features from the parents to the offspring, and hence, a more efficient search process. Let:

O be the order of the schema (the number of defining elements),
D be the length of the schema (the sum of the number of defining elements and the don't cares),
K be the length of the cut (for Order and PMX crossover),
L be the length of the string (the number of cells in the placement problem),
N be the number of elements taken from parent 1 (for Cycle crossover).

These parameters are illustrated in Fig. 10.

Order Crossover: The probability of the schema surviving from parent 1 (from which an entire segment is taken) to the offspring is

$$P_1 = \frac{K - D + 1}{L}.$$

The probability of the schema surviving from parent 2 (from which the rest of the elements are taken in order) is the probability that none of the elements of the schema have been taken from parent 1. If the length of the schema is small compared to the length of the string, then

$$P_2 \approx \left(1 - \frac{K}{L}\right)^D.$$

Thus the probability of survival of the schema from either parent is

$$P_{OC}(S) \approx \frac{K - D + 1}{L} + \left(1 - \frac{K}{L}\right)^D.$$

PMX: The probability of the schema surviving from parent 1 (from which all elements in the cut segment are taken) is

$$P_1 = \frac{K - D + 1}{L}.$$

The probability of the schema surviving from outside the cut segment is

$$P_2 \approx \left(1 - \frac{K}{L}\right)^O$$

if the length of the schema is small compared to the length of the string. The probability of the schema being completely outside the cut segment is

$$P_3 = \frac{L - K - D + 1}{L}.$$

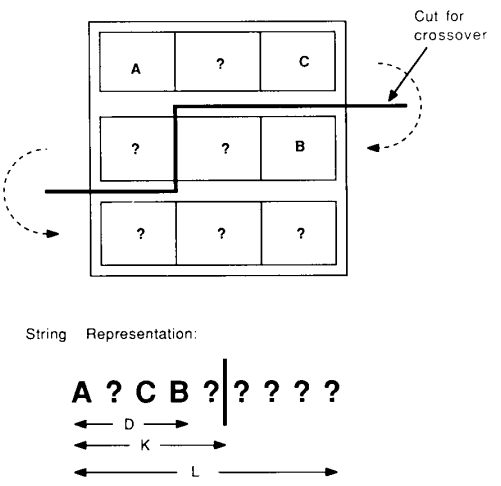


Fig. 10. The parameters used in the analysis of the crossover operators.

Thus the net probability of the schemata surviving in PMX is

$$P_{\mathrm{PMX}}(S) = P_1 + P_2P_3 \approx \frac{K - D + 1}{L} + \left(1 - \frac{K}{L}\right)^O \times \frac{L - K - K + 1}{L}.$$

Cycle Crossover: For large L and small O, the probability of the schemata surviving from parent 1 is

$$P_1 \approx \left(\frac{N}{L}\right)^O.$$

The probability of the schema surviving from parent 2 is similarly

$$P_2 \approx \left(\frac{L - N}{L}\right)^O.$$

Thus the overall probability of survival is

$$P_{CC}(S) \approx \left(\frac{N}{L}\right)^O + \left(\frac{L - N}{L}\right)^O.$$

These results indicate that the survival probability for schemata from parent 1 is the same for Order crossover and PMX. The survival probability of very small schemata from parent 2 is larger in Order crossover than in PMX, but it decreases exponentially in Order crossover as the length D of the schema increases, while in PMX the dependence on the length of the schema is linear, as shown in Fig. 11. Thus the overall survival probability is greater for order crossover in case of problems where compact schemata are important. Similarly the survival probability is greater for the PMX operator in problems where compact schemata are less important. In the case of Cycle crossover, the probability of survival of schemata is independent of the length of the schemata, and depends only on the order as shown in Fig. 12.

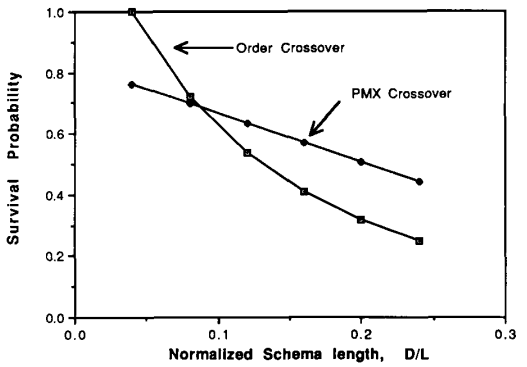


Fig. 11. Survival probability of schemata for Order and PMX crossover.

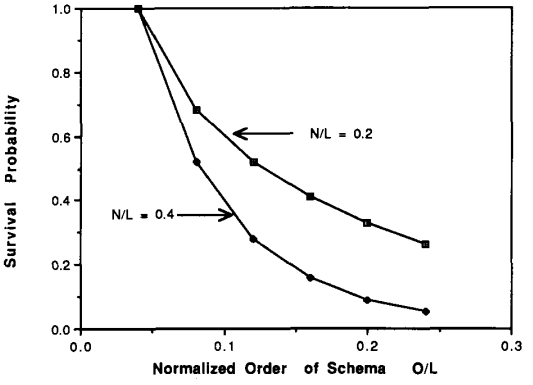


Fig. 12. Survival probability of schemata for cycle crossover.

Acknowledgment

The authors would like to thank Prof. Carl Sechen of Yale University for providing TimberWolf, Prof. B. Bannerjee of the University of Illinois, for providing the netlists of industrial circuits used in the evaluation of GASP-1, and Mr. Lorenzetti of MCNC for providing the benchmark netlists.

References

[1] M. A. Breuer "Min-cut placement" *J. Design Automat. Fault Tolerant Comput.*, vol. 1, pp. 343–382, Oct. 1977.

[2] H. Chang and P. Mazumder "Bit-map crossover based genetic algorithm for macro-cell placement" Tech. Rep., CRL-TR-10-89, Dep. Elect. Eng. Comput. Sci., Univ. of Michigan, 1989.

[3] C. Cheng and E. Kuh "Module placement based on resistive network optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 218–225, July 1984.

[4] J. P. Cohoon, W. D. Paris "Genie technical report," Dep. Comput. Sci., Univ. of Virginia, 1986.

[5] F. Darema, S. Kirkpatrick, and V. A. Norton "Parallel techniques for chip placement by simulated annealing on shared memory systems," in *Proc. IEEE Int. Conf. on Computer Design*, pp. 87–90, 1987.

[6] L. Davis "Job shop scheduling with genetic algorithms," in *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, pp. 136–140, 1985.

[7] —— "Applying adaptive algorithms to epistatic domains," in *Proc. Int. Joint Conf. on Artificial Intelligence*, 1985.

[8] A. E. Dunlop and B. W. Kernighan "A procedure for placement of standard-cell VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 92–98, Jan. 1985.

[9] A. C. Englander "Machine learning of visual recognition using genetic algorithms," in *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, pp. 197–201, 1985.

[10] D. E. Goldberg and R. Lingle "Alleles, loci, and the traveling salesman problem," in *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, 1985.

[11] S. Goto "An efficient algorithm for the two-dimensional placement problem in electrical circuit layout," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 12–18, Jan. 1981.

[12] J. J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht "Genetic algorithms for the traveling salesman problem," in *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, pp. 160–168, 1985.

[13] J. J. Grefenstette "Optimization of control parameters for genetic algorithms," *IEEE Trans. Systems, Man, Cybernet.*, vol. SMC-16, Jan./Feb. 1986.

[14] J. H. Holland *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan, 1972.

[15] M. Jones and P. Bannerjee "Performance of a parallel algorithm for standard cell placement on the intel hypercube," in *Proc. Design Automation Conf.*, 1987.

[16] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi "Optimization by simulated annealing," *Sci.*, vol. 220, no. 4598, pp. 671–680, May 13, 1983.

[17] R. M. Kling "Placement by simulated evolution," M.S. thesis, Coordinated Science Lab, College of Engr., Univ. of Illinois at Urbana-Champaign, June 1987.

[18] S. A. Kravitz and R. A. Rutenbar "Placement by simulated annealing on a multiprocessor," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 534–549, June 1987.

[19] S. Nahar, S. Sahni and E. Shragowitz "Experiments with simulated annealing," in *Proc. 22nd Design Automation Conf.*, pp. 748–752, 1985.

[20] I. M. Oliver, D. J. Smith, and J. R. C. Holland "A study of permutation crossover operators on the traveling salesman problem," in *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, pp. 224–230, 1985.

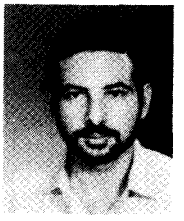
[21] C. Sechen and A. Sangiovanni-Vincentelli "The TimberWolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 510–522, Apr. 1985.

[22] I. Stadnyk "Schema recombination in pattern recognition problems," in *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, pp. 27–35, 1987.

[23] P. Suaris and G. Kedem "Quadrisection: A new approach to standard cell layout," in *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 474–477, 1987.

[24] S. W. Wilson "Adaptive 'cortical' pattern recognition," in *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, pp. 188–196, 1985.

*



Khushro Shahookar received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan in 1986, and the M.S. degree in electrical engineering from the University of Michigan, Ann Arbor in 1989, where he is currently working towards the Ph.D. degree.

His main interests are VLSI fabrication technology and automated VLSI layout algorithms.

*

Pinaki Mazumder (S'84-M'87) received the B.S.E.E. degree from the Indian Institute of Science in 1976, the M.Sc. degree in computer science from the University of Alberta, Canada, in 1985, and the Ph.D. degree in electrical and computer engineering from the University of Illinois in 1987.

Presently he is working as an Assistant Professor at the Department of Electrical Engineering and Computer Science of the University of Michigan, Ann Arbor. Prior to this he worked two years as a research assistant at the Coördinated Science Laboratory, University of Illinois, and over six years at the Bharat Electronics Ltd. (a collaborator of RCA) in the area of integrated circuit design and applications. During the summers of 1985 and 1986, he worked as a Member of Technical Staff at the AT&T Bell Laboratories, Indian Hill, Naperville, in the area of hardware synthesis from system level behavioral description. His research interests include VLSI testing, computer-aided design, parallel architecture, and neural networks.

Dr. Mazumder is a member of Phi Kappa Phi and SIGDA. He is the recipient of the Bell Northern Laboratory Faculty Award.