A Macro-Cell Global Router Based on Two Genetic Algorithms

Henrik Esbensen

Computer Science Department
Aarhus University
DK-8000 Aarhus C, Denmark
E-mail: hesbensen@daimi.aau.dk

Abstract

This paper presents a novel approach to global routing of macro-cell layouts. A genetic algorithm generates several short routes for each net. Another genetic algorithm then selects a route for each net while minimizing area and secondarily interconnect length. Exact channel densities are used for area estimation. The layout quality obtained on MCNC benchmarks compares favourably to that of TimberWolfMC.

1 Introduction

A well-known strategy for global routing of macro-cell layouts consists of two phases [10]. In the first phase, a number of alternative routes are generated for each net. The nets are treated independently one at a time, and the objective is to minimize the length of each net. In the second phase, a specific route is selected for each net, subject to channel capacity constraints, and so that some overall criterion, typically area or total interconnect length, is minimized. A main advantage of this routing strategy is its independence of a net ordering.

Mercury [7] and TimberWolfMC [8] are state of the art global routers for macro-cell layouts, and both are based on the two-phase strategy. For nets with a small number of terminals, these routers generate up to $10-20$ alternative routes for each net. However, due to the time complexity of the applied algorithms, only a single route is generated for nets having more than $5-11$ terminals. As noted in [8] this limits the overall quality obtainable.

In this paper a new global router is presented which minimizes area and secondarily, total interconnect length. While also being based on the two-phase strategy, this router differs significantly from previous approaches in two ways:

1) Each phase is based on a genetic algorithm (GA). The GA used in phase one provides several high-quality routes for each net independently of its number of terminals. In the second phase another GA minimizes the dual optimization criterion by appropriately selecting a specific route for each net.

2) The estimates of area and total interconnect length used throughout the optimization process are calculated very accurately. The area estimate is based on computation of channel densities and the wirelength estimate is based on exact pin locations.

Experimental results shows that the layout quality obtained by the router compares favourably to that of TimberWolfMC.

2 Phase One of the Router

Before the global routing process itself is initiated a rectilinear *routing graph* is extracted from the given placement. Routing is then performed in terms of this graph, i.e., computing a global route for a net is done by computing a corresponding path in the routing graph.

A quite detailed description of how to generate the routing graph for a given placement is given in [7]. Roughly speaking, each edge of the graph corresponds to a routing channel and each vertex corresponds to the intersection of two channels. An example is shown in Fig. 1.

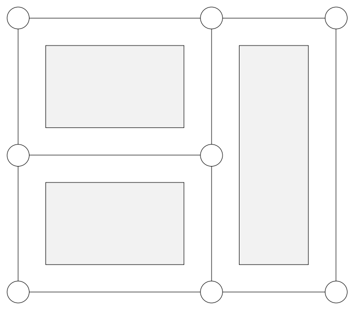

Figure 1: *A placement and the corresponding routing graph.*

Before finding routes for a given net, vertices representing the terminals of the net are added to the routing graph at appropriate locations. Finding the shortest route for the net is then equivalent of finding a minimum cost subtree in the graph which spans all of the added terminal vertices, assuming that the cost of an edge is defined as its length. This problem is known as the Steiner Problem in a Graph (SPG). When a net has been treated, its terminal vertices are removed from the routing graph before considering the next net, thereby significantly reducing the size of the SPG instances to be solved.

For each terminal the location of the corresponding terminal vertex is determined by a perpendicular projection of the terminal onto the edge representing the appropriate routing channel, as illustrated in Fig. 2. This is in contrast to the strategy used in e.g. [7]. Here vertices are added only at the center of routing channels and each terminal is then assigned to the closest vertex. This scheme may result in some nets having identical sets of terminal vertices, in which case some computations can be avoided. On the other hand, our scheme provides a more accurate estimate of the wirelength and also allows a more accurate area estimate as discussed in Section 3.1.

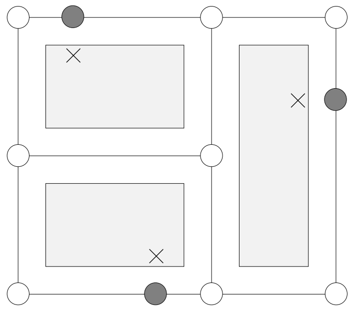

Figure 2: *Addition of terminal vertices (shaded) for a net with three terminals (marked with crosses).*

Fig. 3 outlines phase one. A net is *trivial* if all its terminals are projected onto the same edge of the routing graph. Although several routes can still be generated for a trivial net, it will rarely be advantageous. Hence, global routing is skipped for such nets.

```
generate routing graph
for each non-trivial net do :
   add vertices to graph
   if 2-terminal net :
      apply Lawlers algorithm
   else
      apply a GA for SPG
   end
   remove vertices from graph
end
```

Figure 3: *Outline of phase one.*

The SPG is in general NP-complete. However, if only two vertices are to be connected, SPG reduces to a shortest path problem, which is handled by an algorithm of Lawler discussed in Section 2.1. Nets with more than two terminals are handled by a GA discussed in Section 2.2.

2.1 Two-terminal nets

For each net with two terminals, an algorithm due to Lawler [6] is used to compute the shortest, second-shortest, third-shortest, etc. route until a maximum of R routes are found or no more routes exists. Lawlers algorithm is exact but also quite expensive, requiring time $O(Rn^3)$ for one net, where n is the number of vertices in the routing graph.

An earlier algorithm by Dreyfuss [1] may at first seem more attractive. It generates the R shortest routes from a designated vertex to each of the other vertices in time $O(Rn \log n)$. However, loops are allowed in a path, as opposed to Lawlers algorithm, and if two paths do not visit the same vertices in the same order they are considered distinct. One could then simply generate routes until R loopless routes were obtained, which were also distinct in the sense that their sets of edges are distinct. However, experiments have shown that this strategy is not feasible in practice due to the number of routes then required.

2.2 Nets with at least three terminals

At most R distinct routes are generated for each net having three or more terminals using a GA for the SPG. For a detailed description of that algorithm the reader is referred to [2, 3]. There are two main advantages of using that algorithm in this context. Firstly, it generates high-quality solutions. In [2] the GA is tested on graphs with up to 2,500 vertices and is found to be within 1 % from the global optimum solution in more than 92 % of all runs. The routing graph of a macro-cell placement with C cells will have less than $3C$ vertices. It is therefore most likely that the GA will find the shortest existing route for every net in any reasonably sized macro-cell layout. The second advantage of the GA is that it provides a number of distinct solutions in a single run. The problem of Mercury and TimberWolfMC that only one route is generated for nets with many terminals is thus eliminated.

For nets with few terminals, say 6-7 or less, exhaustive search for the shortest route will often be feasible. Using an algorithm by Sullivan [9] optimum can be found by exhausting a search space consisting of

$$\sum_{i=0}^{k} \binom{n}{i}$$

points, where $k = \min(t-2, n)$ and t is the number of terminals of the net. However, experiments has revealed that Sullivans algorithm often considers fewer distinct solutions and is slower than the GA. Therefore, the GA is used for every net with more than two terminals [1].

3 Phase Two of the Router

The area estimate is of course crucial to the phase two algorithm, and is discussed in Section 3.1. A detailed description of the GA performing the optimization then follows in Section 3.2.

[1]To obtain as many distinct solutions as possible, the GA does not use the reduction of the search space described in [2, 3].

3.1 Area Estimation

As in [7, 10] the area estimation is based on the formation of polar graphs as illustrated in Fig 4. For a given placement and routing graph, two polar graphs are constructed, a horizontal (HPG) and a vertical (VPG). Let us start by considering HPG. The vertices of HPG consists of a vertex for each cell plus two additional vertices, a source and a sink. Each edge in HPG corresponds to a vertical edge in the routing graph and is directed from the source towards the sink.

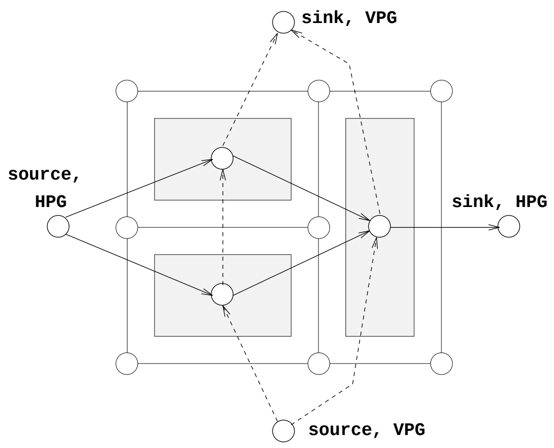

Figure 4: *Polar graphs for area estimation.*

Assume that each edge (v, w) has a cost which corresponds to the spacing needed between cells v and w to perform the routing. Furthermore, assign to each path from source to sink a fixed cost which is the sum of the horizontal length of all cells visited on the path. The total cost of the longest path in HPG then estimates the horizontal length of the layout. By constructing VPG in a similar way, the area is estimated as the product of the longest path in HPG times the longest path in VPG.

In [7] the cost of an edge in the polar graphs is a rather simple function of the number of nets present in the corresponding routing channel. However, if m nets are present in a channel, the channel density can be any number between 0 and m, assuming that two metal layers are available for routing and that each layer is used exclusively for routing in a specific direction. Therefore, to obtain a more accurate area estimate, we compute the exact channel density for each edge in the routing graph. This is possible since the routing in phase one was performed using accurate positions for the terminals of each net, cf. Section 2. The cost of an edge in the polar graphs is then proportional to the density of the corresponding channel.

Several factors affects the accuracy of the area estimate. The two most important has to do with the subsequent compaction/spacing of the layout:

1) If the compactor alters the placement to the extent where the topology of the routing graph is changed, the polar graphs are also changed. Hence, the quality of the area estimate decreases significantly or may even become meaningless. In other words, a good initial placement is required so that the compactor will only perform minor adjustments of the cell positions. This situation reflects the well-known strong mutual dependency of the placement and global routing tasks.

2) It is implicitly assumed that the compactor generates a layout in which no routing channel on a longest path of a polar graph is wider than needed. Otherwise, the area will be underestimated.

The practical consequences of these assumptions are addressed in Section 4.3.

3.2 Area and Wirelength Optimization

The concept of genetic algorithms, introduced by John Holland [5], utilizes the notion of the natural evolution process. In nature, the individuals constituting a population adapt to the environment in which they live. The fittest individuals have the highest probability of survival and tend to increase in numbers, while the less fit individuals tend to die out. This *survival-of-the-fittest* Darwinian principle is the basic idea behind the GA.

The algorithm maintains a *population* of *individuals*, each of which corresponds to a specific solution. A measure of *fitness* defines the quality of an individual. Starting with some set of individuals, a process of evolution is simulated. The main components of this process are *crossover*, which mimics propagation, and *mutation*, which mimics the random changes occurring in nature. After a number of *generations*, highly fit individuals will emerge corresponding to good solutions to the given optimization problem. A good introduction to GAs is given in [4].

```
generate($P_C$);
evaluate($P_C$);
$s$ = bestOf($P_C$);
repeat until stopCriteria():
  $P_N = \emptyset$;
  repeat $M/2$ times:
    select $p_1 \in P_C$, $p_2 \in P_C$;
    crossover($p_1, p_2, c_1, c_2$);
    $P_N = P_N \cup \{c_1, c_2\}$;
  end;
  evaluate($P_C \cup P_N$);
  $P_C$ = reduce($P_C \cup P_N$);
  $\forall\, p \in P_C$ : possibly mutate($p$);
  $\forall\, p \in P_C$ : possibly invert($p$);
  evaluate($P_C$);
  $s$ = bestOf($P_C \cup \{s\}$);
end;
optimize($s$);
output $s$;
```

Figure 5: *Outline of phase two.*

Fig. 5 outlines the phase two algorithm. Initially, the current population P_C of size $M = |P_C|$ consists of $M-1$ randomly generated individuals and a single individual consisting of the shortest route found for each net. Seeding the population with this relatively good solution does not lead to better final results, but merely speeds up the search process. Routine *evaluate* described in Section 3.2.2 computes the fitness of each of the given individuals, while *bestOf* finds the individual with the highest fitness. One execution of the outer "repeat" loop corresponds to the simulation of one generation. Throughout the simulation, M is kept constant. We keep track of the best individual s ever seen. Routine *stopCriteria* terminates the simulation when no improvement has been observed for S consecutive generations. Each generation is initiated by the formation of a set of offspring P_N of size M. The two mates p_1 and p_2 are selected independently of each other, and each mate is selected with a probability proportional to its fitness. The *crossover* routine described in Section 3.2.3 generates two offspring c_1 and c_2. Routine *reduce* returns the M fittest of the given individuals, thereby keeping the population size constant. The genetic operators for mutation and inversion are discussed in Section 3.2.4. Routine *optimize*(s) performs simple hill-climbing by executing a sequence of mutations on s, each of which improves the fitness of s. The output of the algorithm is then the solution s.

3.2.1 Representation

A global routing solution is represented by specifying for each net which of its possible routes is used. More specifically, assume a fixed numbering $0, 1, \ldots, N-1$ of the nets, let $\pi : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$ be a bijection and denote by $r_k \leq R$ the number of routes generated in phase one for the k'th net. An individual is then a set of N tuples:

$$\{(\pi(0), q_{\pi(0)}), (\pi(1), q_{\pi(1)}), \ldots, (\pi(N-1), q_{\pi(N-1)})\}$$

where $1 \leq q_k \leq r_k$ for all $k = 0, 1, \ldots, N-1$. For example, the tuple (3,7) specifies that the 3rd net uses its 7'th route. The mapping π defines an ordering of the nets, the purpose of which is explained in Section 3.2.4. Note that the routing solution specified by an individual is independent of π.

3.2.2 Definition of Fitness

Given a population P, the routine *evaluate* of Fig. 5 computes the fitness of each individual as follows. For each individual $p \in P$, its estimated area is computed as described in Section 3.1 and its estimated total wirelength is computed as the sum of the length of the routes specified by p.

The population $P = \{p_0, p_1, \ldots, p_{M-1}\}$ is then sorted lexicographically using area as most significant criterion and wirelength as a secondary criterion. Assume that P is sorted in decreasing order with respect to this ordering. The fitness F of p_i is then computed as $F(p_i) = 2i/(M-1)$ for $i = 0, 1, \ldots, M-1$. This scheme, called *ranking*, assures constant variance of fitness throughout the optimization process. Ranking is a good approach for controlling the speed of convergence, including the avoidance of premature convergence.

3.2.3 Crossover Operator

Given two parent individuals α and β, the crossover operator generates two offspring, ϕ and ψ. The parent individuals are not altered by the operator. In the following, a (second) subscript specifies which individual the marked property is a part of. Crossover consists of two steps:

1) One of the parents, say β, is chosen at random, and a copy γ of β is made. γ is then reordered so that it becomes homologous to α, that is, $\pi_\gamma = \pi_\alpha$.

2) The offspring are given the same ordering as their parents: $\pi_\phi = \pi_\psi = \pi_\alpha$. Standard 1-point crossover is then performed [5]: A crossover-point x is selected at random in $\{0, 1, \ldots, N-2\}$. The selected routes of ϕ is then defined by $q_{\pi(k),\phi} = q_{\pi(k),\alpha}$ if $k \leq x$ and $q_{\pi(k),\phi} = q_{\pi(k),\gamma}$ otherwise, where $\pi = \pi_\alpha$. Similarly, the selected routes of ψ is defined by $q_{\pi(k),\psi} = q_{\pi(k),\gamma}$ if $k \leq x$ and $q_{\pi(k),\psi} = q_{\pi(k),\alpha}$ otherwise.

3.2.4 Mutation and Inversion Operators

The mutation operator is very simple. It goes through the N tuples of the given individual and randomly selects another route for the k'th net with probability $p_{mut}(r_k - 1)$, where p_{mut} is a small userdefined probability. This scheme is called pointwise mutation.

As mentioned in Section 3.2.1 a given global routing solution can be represented by several equivalent individuals because of the independence of the ordering π. However, the fitness of offspring produced by crossover depends on the specific orderings of the given parent individuals. The purpose of inversion is to optimize the performance of the crossover operator. With a given probability p_{inv}, the inversion operator alters the ordering π of a given individual. To obtain a uniform probability of movement of all tuples, we consider the set of tuples to form a ring. A part of the ring is then selected at random and reversed. More specifically, two points $x, y \in \{0, 1, \ldots, N-1\}, x \neq y$, are selected at random. The operator then defines the new ordering π' as[2] $\pi'((x+i) \bmod N) = \pi((y-i) \bmod N)$ if $0 \leq i \leq (y-x) \bmod N$ and $\pi'((x+i) \bmod N) = \pi((x+i) \bmod N)$ otherwise, $i = 0, 1, \ldots, N-1$.

4 Experimental Results

The router has been implemented in the C programming language, and all experiments are performed on a Sun Sparc IPX workstation. The router is interfaced with the macro-cell layout system Mosaico, which is a

[2]The definition of π' relies on the mathematical definition of modulo, in which the remainder is always non-negative.

part of the Octtools CAD framework developed at University of California, Berkeley. This integration allows for comparison of the routers performance to that of TimberWolfMC [8], a state of the art global router also interfaced to Mosaico.

4.1 Test Examples

Three of the MCNC macro-cell benchmarks, xerox, ami33 and ami49, were used for the experiments. However, due to a purely technical problem, it became necessary to remove all pads from these examples before using them [3]. The modified benchmarks are referenced using a '-M' suffix.

Problem	#cells	#nets	#terms.
xerox-M	10	203	696
ami33-M	33	85	442
ami33-2-M	33	85	442
ami49-M	49	390	913
ami49-2-M	49	390	913

Table 1: *Problem characteristics.*

Table 1 lists the main characteristics of the test examples. The number of nets and the number of terminals listed are totals, i.e., they include the few trivial nets. xerox-M, ami33-M and ami49-M are placed by Puppy, a placement tool based on simulated annealing, also included in Octtools. ami33-2-M and ami49-2-M are other placements of ami33-M and ami49-M, respectively. The generation of these placements are described in Section 4.3.

4.2 Method

Two factors makes it difficult to device a sequence of experiments providing an absolute fair performance comparison of the two global routers. Firstly, global routing is just one of a sequence of heavily interacting steps needed to generate a complete layout. Hence, when considering a specific result, it may be influenced by a pattern of interactions with other tools, which accidentally favours one of the routers. Secondly, the optimization strategies used by the two routers are not identical. As described earlier, the GA-based router explicitly attempts to minimize area and secondarily wirelength. While TimberWolfMC also generates the shortest possible routes in phase one, area is not an explicit component of the optimization criterion used in the second phase. Instead, TimberWolfMC selects the shortest possible routes subject to channel capacity constraints.

The chosen strategy for experiments are as follows: For each of the placed examples listed in Table 1, Mosaico was executed to generate a complete layout, using either TimberWolfMC or the GA-based router for the global routing task. Hence, all other steps of the layout process are performed by the same tools.

Mosaico was executed five times for each example using the GA-based global router in order to capture the variations caused by the stochastic nature of the applied algorithms. The same set of parameters are used for all program executions, i.e., no problem specific tuning is performed. For each net, at most $R = 30$ alternative routes are generated. The parameters of the GA used in phase one are as given in [3]. The phase two GA is executed with population size $M = 40$, stop criteria $S = 100$, mutation probability $p_{mut} = 2.5 \times 10^{-4}$ and inversion probability $p_{inv} = 0.1$. There is no variation of results when applying TimberWolfMC.

4.3 Layout Quality

Table 2 summarizes the impact on the completed layouts of using the GA-based router instead of TimberWolfMC. A_{tot} denotes total area, A_{route} denotes routing area, i.e., the part of the total area not occupied by cells and WL denotes total wirelength. Each entry is computed as 100(GA-result/TW-result − 1). Hence, a negative value indicates a reduction in percent obtained by the GA-based router, while a positive value indicates a percentage overhead as compared to TimberWolfMC. Despite the inherent problems of this kind of comparison discussed in Section 4.2, it is clear that in general the GA-based global router obtains the best layout quality for the problem instances considered.

Problem	Solution	A_{tot}	A_{route}	WL
xerox-M	best	−1.9	−4.7	+0.0
	avg	−1.4	−3.5	+0.8
ami33-M	best	−3.2	−5.1	−3.2
	avg	+1.6	+2.5	−0.2
ami33-2-M	best	−3.0	−4.7	−1.5
	avg	−1.1	−1.7	−0.2
ami49-M	best	−1.9	−3.3	−1.5
	avg	−0.5	−0.8	+0.3
ami49-2-M	best	−4.2	−7.3	−4.0
	avg	−3.7	−6.3	−2.9

Table 2: *Relative improvements obtained by the GA-based router. best and avg. is best and average of the five runs performed.*

Inspection of the generated layouts reveals interesting information regarding the two major assumptions underlying the area estimation, discussed in Section 3.1. The placement of xerox-M is adjusted only slightly during compaction, and the routing graph topology is unaltered. For this example, the GA-based router obtains an average reduction of 3.5 % of the routing area which comes at the price of a 0.8 % increase in total wirelength. However, for ami33-M, the GA-based router on average obtains larger layouts than TimberWolfMC. In this case the placement, and hence the routing graph topology, is significantly modified by the compactor. As a consequence, the function minimized by the GA-based router in its second phase correlates very poorly to the actual layout generated, which inevitably leads

[3] In our version of Octtools (5.2) the channel definition program Atlas can not handle the pad placement generated by Padplace.

to a poor result. To counteract this phenomenon, a new placement ami33-2-M was produced by ripping up all routing in the completed layout of ami33-M generated using TimberWolfMC. Since the placement thus obtained is the result of compaction and completion of all routing, it will probably only be subjected to minor adjustments when used itself as input to Mosaico. Experiments confirmed this assumption. The topology of the routing graph of ami33-2-M is unaltered throughout the process and the performance of the GA-based router is now superior to that of TimberWolfMC. Very similar results are observed for ami49-M: The routing graph topology is significantly altered during the layout process. The placement of ami49-2-M is obtained the same way as ami33-2-M, and the performance of the GA-based router improves significantly on this example.

The significant routing graph alterations for some problems are a consequence of rather poor initial placements. It is not clear how better placements would affect the relative performance of the two routers. As placement quality increases, the relative effect of eliminating a wire from the longest path in a polar graph increases, indicating a potential advantage for the GA-based router. On the other hand, a good placement contains less routing, suggesting that the performance gap would be narrowed.

For the test examples considered here, most routing channels on the longest paths are compacted to their minimum widths by the compactor, cf. the second assumption discussed in Section 3.1. However, in most cases at least one channel on the longest paths are still wider than necessary. Hence, the area estimation performed tends to underestimate the final area. However, this assumption appears to be fairly reasonable.

4.4 Runtime

On average the router requires about 22, 12 and 130 minutes to route examples based on xerox, ami33 and ami49, respectively. TimberWolfMC spends about 30 seconds for examples based on xerox and ami33, and about 5 minutes for ami49-based examples. Hence, the GA-based router is clearly inferior to TimberWolfMC with respect to runtime. The total layout generation process performed by Mosaico (i.e. excluding placement) requires about 15 minutes for examples based on xerox and ami33, and about an hour for ami49-based examples, when TimberWolfMC is used. Hence, the use of the GA-based router increases the layout generation time by a factor of two or three.

However, the runtime of the current implementation can be improved significantly in a number of ways. The vast majority of the runtime is spend computing channel densities. When estimating the area of a solution, all densities are recomputed whether the routing in a channel is actually changed or not. Keeping track of the need to recompute channel densities and updating channel densities dynamically would hence reduce runtime significantly. Another approach is to implement a parallel version of the router. Due to the inherent parallelism of any GA, a high speedup can be expected on any MIMD architecture [4].

5 Conclusion and Future Work

In this paper a novel approach to global routing of macro-cell layouts based on genetic algorithms has been presented. The performance of the router is compared to that of TimberWolfMC on MCNC benchmarks. Experimental results shows that the quality of completed layouts improves when using the GA-based router instead of TimberWolfMC, assuming that the quality of the given placement is sufficiently high. The router is inferior to TimberWolfMC with respect to runtime, but major improvements are possible. Since the work presented here is a first approach to global routing based on genetic algorithms, future improvements of the layout quality obtainable are also very likely. We conclude that the genetic algorithm is well suited as the basic algorithm of a global router.

References

[1] Stuart E. Dreyfuss, "An Appraisal of Some Shortest-Path Algorithms," *Journal of the Operations Research Society of America*, Vol. 17, pp. 395-412, 1969.

[2] Henrik Esbensen, "Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm," Technical Report, Daimi PB-468, Aarhus University, Feb. 1994.

[3] Henrik Esbensen, Pinaki Mazumder, "A Genetic Algorithm for the Steiner Problem in a Graph," *Proceedings of the European Design and Test Conference*, pp. 402-406, 1994.

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, 1989.

[5] John H. Holland, *Adaption in Natural and Artificial Systems,* University of Michigan Press, Ann Arbor, MI., 1975.

[6] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart and Winston, New York, 1976.

[7] Y. Nishizaki, M. Igusa, A. Sangiovanni-Vincentelli, "Mercury: A New Approach to Macro-cell Global Routing," *Proceedings of the IFIP 10/WG 10.5 International Conference on VLSI*, Munich, 1989.

[8] Carl Sechen, *VLSI Placement and Global Routing Using Simulated Annealing,* Kluwer Academic Publishers, Boston, 1988.

[9] G. F. Sullivan, "Approximation Algorithms for Steiner Tree Problems," Technical Report 249, Dept. of Computer Science, Yale University, 1982.

[10] R. Venkateswaran, P. Mazumder, "Routing Algorithms in Semiconductor Circuit Design," In preparation.