Vol. 43 No. 5 IPSJ Journal May 2002

Regular Paper

A Genetic Algorithm for the Rectilinear Steiner Tree Problem in VLSI Interconnect Layout

Shin'ichi Wakabayashi[†]

This paper proposes a genetic algorithm for generating a set of rectilinear Steiner trees for the interconnect optimization problem in VLSI layout design. The algorithm produces a set of rectilinear Steiner trees, whose geometrical and timing characteristics are different each other. In the proposed genetic algorithm, each chromosome represents the topological structure of a Steiner tree. An evaluation function is given to map it into the layout of a Steiner tree. Steiner trees produced by the algorithm are Pareto-optimal with respect to the total wire length and the maximum propagation delay, and the user can choose any tree among those solutions as a final routing solution. Experimental results show that the algorithm efficiently produces a set of alternative routes in VLSI interconnect optimization.

1. Introduction

The routing problem in VLSI physical design is generally formulated as the problem of finding a rectilinear Steiner tree, which connects a given source with a set of sinks with vertical and horizontal wire segments. The minimum Steiner tree construction problem is an NP-hard problem, and thus many heuristic algorithms have been proposed[15)]. On the other hand, with the advent of sub-micron geometries in semiconductor technology, wire resistance becomes a significant contributor to signal delay, and thus routing should be performed under the timing constraints with an appropriate delay model[4)].

Most of previous work on Steiner tree construction considering timing constraints was aiming at producing one Steiner tree as an output for a given net[3)]. However, there often exist cases, in which more than one trees are required to generate. For example, the rip-up and reroute technique is common in LSI routing. A set of Steiner trees helps the router to find a better alternative route. Besides rip-up and rerouting, a set of Steiner trees often gives the good flexibility to the router. However, there have been few previous algorithms, which could effectively produce a set of Steiner trees with different geometrical as well as timing characteristics. Alpert *et al.* proposed a Steiner tree construction algorithm, which smoothly combine the minimum cost and the minimum radius objectives[1),2)]. Using this algorithm, a set of rectilinear Steiner trees could be obtained by varying the value of a control parameter of cost-radius tradeoff. However, this algorithm did not directly treat the propagation delay. Cong and Koh proposed a branch-and-bound-based Steiner tree construction algorithm with wire-sizing, which produced a set of required-arrival-time Steiner trees, providing a smooth trade-off among signal delay, wave form, and routing area[5)]. Although this algorithm could produce a set of good alternative routes, due to its branch-and-bound nature, the CPU time would become large when the number of sinks is large.

In this paper, we propose an algorithm to produce a set of Steiner trees. The algorithm produces a set of rectilinear Steiner trees, whose total wire length and signal propagation delay are different each other. The proposed algorithm is a genetic algorithm. Genetic algorithms (GAs)[8)] are known to be robust heuristic algorithms to solve optimization problems, and for VLSI design areas, a number of GA based algorithms have been presented[12),13)]. For the (rectilinear) Steiner construction problem, a few GAs have been also proposed[9),11)]. In those previous GAs, a chromosome directly represents the geometry of a Steiner tree. We have a different approach to representing a solution of the problem. In the proposed GA, each chromosome represents a topological structure of a Steiner tree. An evaluation function is given to map it into the layout of a Steiner tree. Furthermore, the proposed GA has another feature, which also discriminates the proposed GA from the previous ones. Steiner trees produced by the algorithm are *Pareto-optimal*[8)] with respect to the total wire length and the maximum propagation delay, and the user can choose any

[†] Hiroshima University

tree among those solutions as a final solution. As the interconnect delay model, we adopt the Elmore delay model[7]. To evaluate the proposed GA, we have implemented it on a workstation, and compared it with the Steiner tree construction algorithm proposed by Alpert, et al.[1],[2]. Experimental results show that the proposed algorithm efficiently produces a set of alternative routes in VLSI interconnect optimization.

This paper is organized as follows. In Section 2, the delay model is given and the problem is formulated. In Section 3, the proposed algorithm is presented. Section 4 shows experimental results to evaluate the proposed algorithm, and finally, Section 5 concludes with possible directions for future research.

2. Preliminaries

2.1 Delay Model

As in most previous work on interconnect layout optimization, we adopt the Elmore delay model[7] for interconnects. For wire e, let l_e, c_e and r_e denote length, capacitance, and resistance, respectively. Further, let e_v denote the wire entering node v from its parent. We use the following model for interconnects delay D_{wire}:

$$c_e = c_u \times l_e, r_e = r_u \times l_e$$

$$D_{wire}(e_v) = r_{e_v} \times (c_{e_v}/2 + c(T_v))$$

where c_u and r_u are capacitance and resistance for unit-length wire, respectively, T_v is the subtree rooted at v, and $c(T_v)$ is the total capacitance of T_v. If v is a sink, then $c(T_v) = c_b$, where c_b is the input load of the buffer. The Elmore delay from source s_0 to sink s_i is

$$t_{Elmore}(s_0, s_i) = \sum_{e_v \in path(s_0,s_i)} D_{wire}(e_v).$$

2.2 Problem Formulation

We use the total wire length and the maximum source-sink delay as our optimization objectives. Since there are two terms to be minimized, in a usual formulation, one of two terms would be handled as a secondary objective. Linear combination of two terms as one unified objective may be another common method for this case. In this paper, however, we treat the Steiner tree construction problem as a multi-objective problem, and introduce the concept of *Pareto-optimality*[8] to this problem.

Let P be a multi-objective optimization problem. For simplicity, we assume that two objective functions, f and g, are given to be minimized. Generalization to more than two objective functions is easy. Any solution of P, which satisfies P's all constraints, but does not necessarily minimize any of two objective functions, is called a feasible solution. Let s be a feasible solution of P, and the values of two objectives are represented as $(f(s), g(s))$. A feasible solution s is said to be a *noninferior* (or, *nondominated*) solution if there is no feasible solution s' such that $f(s') < f(s)$ and $g(s') < g(s)$. A set of noninferior solutions is said to be *Pareto-optimal*.

The Steiner tree problem in this paper is stated as follows:

Given a source s_0 and n sinks s_1, s_2, ..., s_n of a signal net S with given positions, find a Pareto-optimal set of rectilinear Steiner trees with respect to two objective functions, i.e., the total wire length of the tree and the maximum source-sink delay among all routes from the source to sinks.

Note that, in VLSI design, any net will be driven by a gate at the source, and due to the fanout limitation of a gate, the number of sinks of a net, denoted n, is bounded by some constant. This constant depends on the semiconductor technology, with which the VLSI design is performed, but we can safely assume that the number of sinks of any net is at most 30, and in most cases, it is less than 20.

3. The Algorithm

3.1 Outline of the Algorithm

The algorithm presented in the following is a genetic algorithm (GA). GA is known to be a robust heuristic algorithm to complex optimization problems[8]. For VLSI physical design, many GAs have been also proposed[12],[13].

The proposed GA is a generational GA, and maintains the population consisting of m chromosomes, each representing a rectilinear Steiner tree. Chromosomes in the current generation are recombined and mutated, and selection is performed to produce a new generation. Mapping from a chromosome to a tree is given, and the total wire length and the maximum source-sink delay of the tree are regarded as the fitness values of the chromosome. The proposed mapping is very effective to explore the search space efficiently. Based on the fitness values, tournament selection is performed to construct a new generation with elitist strategy. The algorithm repeats those procedures within the user specified number. An overview

```
generate($P_C$);
evaluate($P_C$);
repeat noOfGenerations times:
  $P_N := \emptyset$;
  repeat noOfOffspring times:
    select $p_1, p_2 \in P_C$;
      $P_N := P_N \cup$mutate(crossover($p_1$,
  $p_2$));
  end;
  evaluate($P_C \cup P_N$);
  $P_C :=$ select($P_C \cup P_N$);
end;
```

Fig. 1 Overview of the algorithm.

of the algorithm is shown in **Fig. 1**.

3.2 Genotype

In GA, a genotype is a coding of the information constituting a chromosome. In our problem, we should represent a rectilinear Steiner tree with an appropriate coding. There may be a number of possible representations of a Steiner tree. In general, for representing a Steiner tree, there are two types of information, that is, geometrical information and topological information. The former specifies how each wire segment in the tree is actually laid out, and the latter specifies the parent-children relations among nodes. In the previously proposed GAs for the Steiner tree problem, those two types of information were both coded in a chromosome. In Ref. 9), a GA for the Steiner tree problem was proposed, in which a chromosome was an assembly of the x, y-positions of a fixed number of Steiner points. Since the layout area of a Steiner tree in VLSI layout design is large in general, using this chromosome, it would be very difficult to realize the efficient search in the solution space of the problem. In Ref. 11), a GA for the rectilinear Steiner tree problem was proposed, in which a chromosome consists of $n-1$ binary symbols and $n-2$ symbols selected from an alphabet of n symbols, where n is the number of points to be connected with a Steiner tree. Due to the Cayley's Formula and Hanan's theorem, from this chromosome, a Steiner tree could be constructed on Hanan grids. However, Hou and Sapatnekar showed that a minimum-delay Steiner tree may require the use of non-Hanan points[10)]. Since, in this paper, we should take not only the wire length but also the signal propagation delay into account, this coding scheme may not be appropriate.

In this paper, a chromosome only specifies the

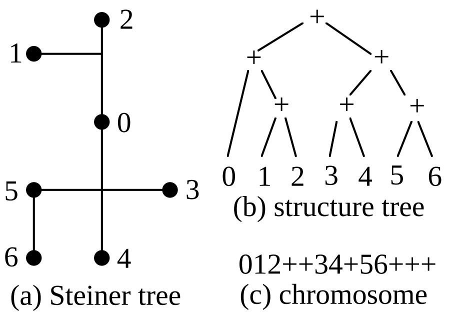

Fig. 2 Genotype.

topological information of a Steiner tree. Geometrical information of the tree will be determined during the fitness evaluation, that is, a Steiner tree will be constructed according to the topological information specified by the chromosome.

Topological information of a Steiner tree is coded as follows. The coding is defined recursively. Let T be a rectilinear Steiner tree, and T_u be a subtree of T. Assume that T_u consists of two subtrees T_v and T_w, each of which contains at least one source or sink. Let $code(T_v)$ and $code(T_w)$ be the coded strings of T_v and T_w, respectively. Then the coded string of T_u is defined as the concatenation of $code(T_v)$, $code(T_w)$, and the special symbol $+$, that is, $code(T_u) = code(T_v)code(T_w)+$. If T_u contains only one source or sink, then $code(T_u)$ is defined as s_i $(0 \leq i \leq n)$.

A chromosome defined here specifies how the tree is constructed. The coding can be interpreted as a tree, whose leaves are a source and sinks, and internal nodes are the special symbols $+$. We call this tree the *structure tree*. From the structure tree, its geometrical information is determined during the fitness evaluation. **Figure 2** shows an example of a Steiner tree and its corresponding structure tree as well as the chromosome.

3.3 Fitness

In a GA, to evaluate each chromosome, a fitness function is required. In our algorithm, each chromosome is evaluated with two objective functions, i.e., the total wire length and the maximum source-sink delay. Since each chromosome only specifies the topological information, to evaluate its fitness, the Steiner tree should actually be laid out according to the topological information given by the structure tree.

Steiner tree construction in the proposed algorithm is performed as follows. Let S_u be a subtree of the structure tree given by the chromosome, and S_v and S_w be the subtrees constituting S_u, where u, v, and w are roots of those subtrees, and v and w are the two children of u. Assume that S_v and S_w have been already laid out. Then, we determine a route from a node in S_v to a node in S_w to construct a Steiner subtree corresponding to S_u. Note that, in this case, nodes to be connected in the subtrees may be existing nodes or intermediate points in the existing edges.

This Steiner construction algorithm is similar to the Kruskal's algorithm for finding a minimum spanning tree of a given graph[6]. However, in the proposed algorithm, the ordering of matching nodes or edges is specified by the structure tree. To complete the description of the algorithm, we should specify how to choose the pair of two nodes or edges to connect two subtrees. One possible idea is to choose the node (edge) pair with the minimum distance. However, this idea has two drawbacks. First, this heuristic is so strong that it would cause the premature convergence into a local optimum. Second, since all node (edge) pairs are required to be checked, time complexity to construct a tree would become $O(n^2)$. Since the fitness evaluation is executed in many times in one GA execution, this is not feasible.

To reduce the computation time, and to avoid causing premature convergence, we restrict the range of searching in combining two subtrees. Let v and w be two roots of subtrees, T_v and T_w, to be combined. Let $E(v)$ and $E(w)$ be sets of edges in T_v and T_w, in which each edge resides within the distance *max_edge_level* from v and w, respectively. The parameter *max_edge_level* is specified by the user. When combining the two subtrees, only edges in $E(v)$ and $E(w)$ are examined so that the time complexity in constructing the whole tree is reduced from $O(n^2)$ to $O(n)$.

Once the Steiner tree is constructed from the chromosome, the total wire length as well as the maximum source-sink delay are calculated.

3.4 Crossover Operator

To generate a set of chromosomes belonging to the population in the next generation, two chromosomes are randomly selected and recombined with a crossover operator with probability p_c. The crossover operator adopted in the proposed algorithm is called the *subtree interchange*. In the following, we treat each chromosome as its corresponding structure tree. Let T and U be the structure tree to be recombined. First, in each structure tree, a + node is randomly selected from all + nodes except its root. Let T_v and U_w be two subtrees whose roots are selected + nodes. Second, two subtrees, T_v and U_w, are interchanged. Interchanging subtrees usually causes the inconsistency in other parts in T and U. To fix this inconsistency, duplicate leaves may be renamed or removed, and new leaves may be added if necessary. Due to the lack of space, the details are omitted, and an example of recombination is shown in **Fig. 3**.

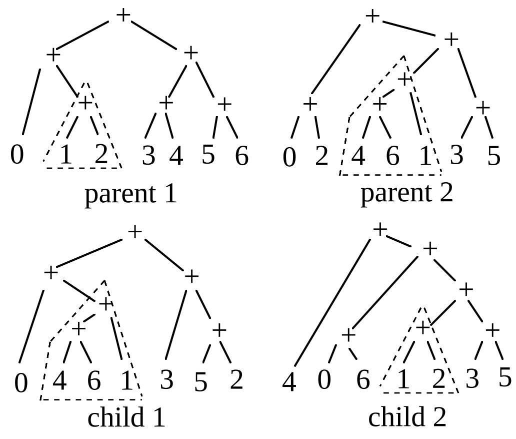

Fig. 3 Crossover.

3.5 Mutation Operator

After performing the crossover, each newly created chromosome is mutated with the mutation operator. The mutation operator adopted in the algorithm is as follows. We regard a chromosome to be mutated as a structure tree T. First, two distinct nodes in T are arbitrarily chosen, and let those nodes be u and v. Let T_u and T_v be subtrees whose roots are u and v, respectively. If there is no node, which is contained in both T_u and T_v, then two subtrees are actually interchanged. Otherwise, the mutation failed. The probability of applying mutation is denoted as p_m.

3.6 Selection

After applying mutation, the population of next generation is constructed with tournament selection with elitist strategy. Since our problem is a two-objective optimization problem, and we want to find a set of Pareto-optimal solutions, selection should also be tailored to be multi-objective. In this paper, we define the following three rank functions for each chromosome. Let $P = \{S_1, S_2, ..., S_m\}$ be the set of chromosomes in the current generation. Let $L(S_i)$ and $D(S_i)$ be the total wire length and the maximum source-sink delay of the Steiner tree given by S_i. Then, three rank functions are defined as follows:

$$\begin{aligned} rank_l(S_i) &= |\{S_j | L(S_j) \le L(S_i), S_j \in P\}| \\ rank_d(S_i) &= |\{S_j | D(S_j) \le D(S_i), S_j \in P\}| \\ rank_p(S_i) &= |\{S_j | L(S_j) \le L(S_i), \\ &\qquad D(S_j) \le D(S_i), S_j \in P\}| \end{aligned}$$

Tournament selection with size 2 is carried out in the algorithm as follows. Let S_i and S_j be chromosomes as inputs of selection. Then, S_i will survive as an element in the new generation if $rank_p(S_i) < rank_p(S_j)$, or if $rank_p(S_i) = rank_p(S_j)$ and $rank_l(S_i) + rank_d(S_i) < rank_l(S_j) + rank_d(S_j)$. S_j will survive if $rank_p(S_j) < rank_p(S_i)$, or if $rank_p(S_j) = rank_p(S_i)$ and $rank_l(S_j) + rank_d(S_j) < rank_l(S_i) + rank_d(S_i)$. Otherwise, one of two chromosomes is randomly chosen as a survivor.

In addition to the tournament selection, we also adopt the elitist strategy. For each chromosome, if it ranked first in any of three ranking functions, it survives as a member of new generation.

4. Experimental Results

4.1 Test Data and Parameter Values

We have implemented the proposed algorithm with the Free Pascal language (Version 1.0.4) on a personal computer (CPU: Pentium III (1 GHz), Memory: 512 MB, OS: Vine Linux 2.1.5). To evaluate the algorithm, we randomly generated two sets of test data, and applied the algorithm to those. For all test data, the chip area was assumed to be $10 \times 10mm^2$. For each test data in the first test data set, which consists of 6 test data, the source was located at the center of the chip area, and each sink was randomly generated in the chip area. The numbers of sinks of test data in the first test data set were 5, 10, 15, 20, 25, and 30, respectively. For each test data in the second test data set, which consists of 5 test data, the number of sinks was set to 20, and the source and 20 sinks were randomly generated in the chip area.

In the experiments, the delay of a Steiner tree was calculated with the Elmore delay model described in 2.1 with the following parameter values: $c_u = 0.15$fF$/\mu$m, $c_b = 50.0$fF, $r_u = 0.12\Omega/\mu$m. As the GA parameters, we set the population size to 21, crossover and mutation probabilities to 0.8 and 0.05, respectively, and the maximum number of generations to 200 except otherwise stated. We also set *max_edge_level* in the fitness evaluation phase to 2. Since the proposed algorithm was based on a GA, and a GA is a stochastic algorithm, for each run of the algorithm, a different solution will be obtained even if the proposed algorithm is applied to the same instance of the problem. Thus, for each test data, we executed the proposed algorithm in 20 times so that a stochastic nature of the algorithm was able to be properly evaluated.

We have also implemented the Steiner tree construction algorithm proposed by Alpert, et al. 1),2) as the previous algorithm☆. In Ref. 2), they showed that the min-cost and min-radius objectives can be addressed simultaneously via direct combinations of the Prim's minimum spanning tree and Dijkstra's shortest-path tree construction algorithms. In their algorithm, called the algorithm AHHKK in the following, there was a control parameter, denoted $c\,(0 \le c \le 1)$, to control the cost-radius tradeoff in the Steiner tree construction. We implemented the AHHKK algorithm with the Free Pascal language on the same computer environment used to implement the proposed algorithm. In the experiments, for each test data, we set c to $0.00025 \times i\,(0 \le i \le 4000)$, and executed the algorithm AHHKK, producing 4001 Steiner trees, so that the total CPU time of the algorithm AHHKK for each test data is roughly the same as the CPU time of the proposed algorithm.

4.2 Comparison with the Algorithm AHHKK

Table 1 shows the results of the algorithm AHHKK for the first test data set. In the table, $\#s$ means the number of sinks. L [mm] is the best value of the total wire length among generated Steiner trees, and D_L [nanoseconds] is the source-sink delay of a tree whose total wire length is L. D [nanoseconds] is the best of the maximum source-sink delay among gen-

☆ In Ref. 2), two algorithms, ALG1 and ALG2, were proposed. ALG1 was also proposed in Ref. 1). Since the authors of Ref. 2) concluded that ALG1 was superior than ALG2, we adopted ALG1. Note that the original ALG1 was the algorithm to produce an ordinary Steiner tree. Thus, to produce the rectilinear Steiner tree, we implemented and added a greedy edge-overlapping method to ALG1. The edge-overlapping method used here was the one given also in Ref. 2).

Table 1 Experimental results of the algorithm AHHKK.

	$c = 0.01$							$c = 0.00025$				
$\#s$	L	D_L	D	L_D	$\#d$	$\#ni$	CPU	D	L_D	$\#d$	$\#ni$	CPU
5	11.227	0.709	0.709	11.227	4	1	0.006	0.709	11.227	4	1	0.220
10	22.308	0.735	0.709	22.447	8	2	0.012	0.709	22.447	8	2	0.491
15	26.827	1.138	1.117	27.360	12	2	0.024	1.117	27.360	12	2	0.928
20	35.902	1.128	0.992	39.089	13	3	0.041	0.992	39.089	15	3	1.618
25	44.549	1.207	1.096	49.079	20	4	0.066	1.038	50.883	29	6	2.578
30	49.491	2.146	1.477	54.754	32	5	0.101	1.477	54.754	39	5	4.022

Table 2 Experimental results of the proposed algorithm.

$\#s$	$minL$	D_{minL}	$minD$	L_{minD}	$aveL$	$aveD$	$maxL$	$maxD$	$\#d$	$\#ni$	CPU
5	**11.088**	**0.617**	**0.617**	**11.088**	**11.088**	**0.617**	**11.088**	**0.617**	**7.9**	**7.7**	0.253
10	**21.801**	**0.589**	**0.589**	**21.801**	**21.801**	**0.658**	**21.801**	0.735	**8.1**	**7.3**	0.568
15	**26.551**	**0.683**	**0.683**	**26.551**	**26.588**	**0.906**	**26.816**	**1.086**	9.2	**8.4**	0.939
20	**33.888**	**0.796**	**0.742**	**34.639**	**34.753**	**0.889**	**35.190**	1.031	9.0	**8.1**	1.323
25	**42.800**	**1.034**	**0.910**	44.657	**44.539**	**1.027**	46.364	1.248	11.4	**9.7**	1.787
30	**48.362**	**1.310**	**1.187**	51.365	51.095	**1.468**	53.988	2.030	11.7	**7.8**	2.275
$30^{\dagger}$	**48.233**	**1.250**	**1.153**	**49.354**	50.636	**1.409**	53.397	1.838	10.4	**8.6**	5.720
$30^{\ddagger}$	**47.596**	**1.337**	**1.172**	51.505	49.927	**1.384**	52.319	1.929	13.8	**11.5**	8.161

$30^{\dagger}$: gen.=500, $30^{\ddagger}$: gen.=500, pop. size = 30.

erated Steiner trees, and L_D [mm] is the total wire length of a tree whose maximum source-sink delay is D. $\#d$ means the number of distinct trees, and $\#ni$ shows the number of distinct noninferior trees. Note that the algorithm AHHKK may produce the inferior solutions as well as the same solutions. CPU [seconds] shows the CPU time of the algorithm to produce 4001 Steiner trees. To evaluate the effect of the value of c to the quality of solutions and the CPU time, in the table, two sets of results were shown, each of which were respectively obtained when $c = 0.01$ and $c = 0.00025$. L and D_L remained unchanged when the value of c was changed, so those data were not shown for the case of $c = 0.00025$. From Table 1, we see that it is difficult to obtain a sufficient number of non-inferior distinct solutions using the algorithm AHHKK even when 4001 trees were generated. We also note that, for the test data with less than 20 sinks, $c = 0.01$ is enough to produce a good solution.

Table 2 shows the experimental results of the proposed algorithm for the first test data set. In the table, $\#s$ means the number of sinks. In this experiments, 20 runs were executed for each data, and for each run, the best of the total wire length, denoted L, and the best of the maximum source-sink delay, denoted D, were obtained. $minL$, $aveL$, and $maxL$ are the best, average, and worst values of all L's, and $minD$, $aveD$, and $maxD$ are the best, average, and worst values of all D's, respectively. D_{minL} shows D of a run, in which $minL$ was obtained, and L_{minD} shows L of a run, in which $minD$ was obtained. $\#d$ means the average number of distinct solutions in 20 runs, and $\#ni$ shows the average number of distinct noninferior solutions in 20 runs. Figures in Table 2 with bold fonts showed that the data was better than L, D, $\#d$, and $\#ni$ in Table 1. CPU shows the average CPU time to execute the algorithm with 200 generations.

From the experimental results with comparison of the results of the algorithm AHHKK, in general, we see that the proposed algorithm effectively produces a set of rectilinear Steiner trees with different topological as well as timing characteristics. The proposed algorithm successfully produces a number of non-inferior distinct solutions, and those solutions are better than ones produced by the algorithm AHHKK. Thus, those trees can be used as good candidates in various phases of routing.

Experimental result for 30-sink data seemed inferior to the ones of data having the smaller number of sinks. To improve the result, two additional experiments were performed. One was the case, in which the maximum number of generations was set to 500, and the other was the case, in which the maximum number of generations was set to 500 as well as the population size to 30. The results of those two cases were shown in Table 2 as $\#s = 30^\dagger$, and $\#s = 30^\ddagger$. Those results show that the proposed algorithm may further improve the solutions if the parameter setting was appropriately done.

To understand the effectiveness of the proposed algorithm more easily, we summarize how many runs the proposed algorithm produced better solutions than the algorithm AHHKK. The results were summarized in **Table 3**. In this table, N_L, N_D, N_{ni} mean the number of runs the proposed algorithm produced better solutions than the algorithm AHHKK in terms of L, D, $\#ni$. N_{all} shows how many runs the proposed algorithm produced better results than the algorithm AHHKK in terms of L, D, $\#ni$ simultaneously.

4.3 More Results for the Case of #s=20

To further confirm that the proposed algorithm was effective to produce the better results than the algorithm AHHKK, we performed another experiments. As noted before, we can assume that, in most cases, the number of sinks of a net is less than 20. Thus, in the following, we prepared 5 additional test data, which consists of one source and 20 sinks. In this second test data set, the source was not located at the center of the chip area, and its coordinates, denoted (x_0, y_0), were randomly specified. Experimental results for the second test data set were summarized in **Table 4**, **Table 5**, and **Table 6**. From those tables, we see that the proposed algorithm constantly produced better results than the algorithm AHHKK. Note that, since the proposed algorithm was a stochastic one, even if a solution produced by the proposed algorithm was not better than the one by the algorithm AHHKK, you will probably get better one from additional one or two more runs of the proposed algorithm.

From the experimental results, we also notice that the proposed algorithm can be also considered as a good algorithm for finding an ordinary minimum Steiner tree without performance consideration. The CPU time of the proposed algorithm was roughly the same or shorter than the algorithm AHHKK when c is set to 0.00025 and about 300 times larger than the algorithm AHHKK when c is set to 0.01. Thus, reduction of the CPU time of the algorithm is worth investigating in the future.

Table 3 Comparison of the proposed algorithm with the algorithm AHHKK.

$\#s$	N_L	N_D	N_{ni}	N_{all}
5	20	20	20	20
10	20	19	20	19
15	20	20	20	20
20	20	18	20	18
25	15	13	19	10
30	3	11	17	2
$30^\dagger$	4	13	18	3
$30^\ddagger$	9	17	20	6

$30^\dagger$: gen.=500,
$30^\ddagger$: gen.=500, pop. size = 30.

Table 4 Experimental results of the algorithm AHHKK for the case of $\#s = 20$ ($c = 0.00025$).

$\#s$	(x_0, y_0)	L	D_L	D	L_D	$\#d$	$\#ni$	CPU
20a	(5.811, 5.704)	32.161	1.285	1.285	32.161	22	1	1.593
20b	(5.519, 2.933)	39.647	2.539	1.561	42.057	15	3	1.581
20c	(9.378, 2.944)	35.928	6.417	2.650	45.798	18	7	1.561
20d	(8.752, 8.913)	35.139	7.805	3.639	38.765	22	7	1.575
20e	(6.779, 5.596)	44.731	1.848	1.841	46.752	28	2	1.602

Table 5 Experimental results of the proposed algorithm for the case of $\#s = 20$.

$\#s$	$minL$	D_{minL}	$minD$	L_{minD}	$aveL$	$aveD$	$maxL$	$maxD$	$\#d$	$\#ni$	CPU
20a	**30.991**	**0.885**	**0.819**	32.416	**31.882**	**0.967**	32.655	**1.201**	10.7	**9.8**	1.366
20b	**37.961**	**1.324**	**1.324**	**37.961**	**38.703**	**1.472**	41.155	1.855	8.4	**7.3**	1.338
20c	**34.466**	**1.806**	**1.753**	**34.760**	**35.065**	**2.260**	**35.843**	3.596	7.3	**6.8**	1.297
20d	**33.311**	**3.605**	**2.779**	**33.409**	**33.627**	**3.447**	**34.075**	4.457	9.0	**7.1**	1.268
20e	**42.163**	**1.398**	**1.398**	**42.163**	**43.006**	**1.573**	**44.041**	1.896	8.9	**7.6**	1.321

Table 6 Comparison of the proposed algorithm with the algorithm AHHKK for the case of $\#s = 20$.

$\#s$	N_L	N_D	N_{ni}	N_{all}
20a	16	20	20	16
20b	16	18	20	14
20c	20	15	13	11
20d	20	18	11	9
20e	20	19	20	19

5. Conclusion

In this paper, we proposed a genetic algorithm to produce a set of rectilinear Steiner trees in VLSI interconnect optimization. Trees produced by the algorithm were different in geometrical as well as timing characteristics, and can be used in various phases in VLSI routing. Future research includes the extension of the proposed algorithm to treat wire sizing and buffer insertion[14]. Since the framework of the proposed algorithm is so flexible that it is easy to change the delay model as well as the objective functions used in the algorithm.

References

1) Alpert, C.J., Hu, T.C., Huang, J.H. and Kahng, A.B.: A direct combination of the Prim and Dijkstra constructions for improved performance-driven global routing, *Proc. ISCAS'93*, pp.1869–1872 (1993).
2) Alpert, C.J., Hu, T.C., Huang, J.H., Kahng, A.B. and Karger, D.: Prim-Dijkstra tradeoffs for improved performance-driven routing tree design, *IEEE Trans. CAD*, Vol.14, No.7, pp.890–896 (1995).
3) Cong, J., He, L., Koh, C.-K. and Madden, P.H.: Performance optimization of VLSI interconnect layout, *Integration, the VLSI Journal*, Vol.21, No.1&2, pp.1–94 (1996).
4) Cong, J.: Modeling and layout optimization of VLSI devices and interconnects in deep submicron design, *Proc. ASP-DAC'97*, pp.121–126 (1997).
5) Cong, J. and Koh, C.-K.: Interconnect layout optimization under higher-order RLC model, *Proc. ICCAD'97*, pp.713–720 (1997).
6) Cormen, T.H., Leiserson, C.E. and Rivest, R.L.: *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts (1990).
7) Elmore, W.C.: The transient response of damped linear network with particular regard to wideband amplifier, *J. Applied Physics*, Vol.19, pp.55–63 (1948).
8) Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts (1989).
9) Hesser, J., Männer, R. and Stucky, O.: Optimization of Steiner trees using genetic algorithms, *Proc. 3rd ICGA*, pp.231–236 (1989).
10) Hou, H. and Sapatnekar, S.S.: Routing tree topology construction to meet interconnect timing constraints, *Proc. International Symposium on Physical Design*, pp.205–210 (1998).
11) Julstrom, B.A.: A genetic algorithm for the rectilinear Steiner problem, *Proc. 5th ICGA*, pp.474–480 (1993).
12) Lienig, J. and Cohoon, J.P.: Genetic algorithms applied to the physical design of VLSI circuits: A survey, *Proc. Parallel Problem Solving from Nature IV*, pp.839–848 (1996).
13) Mazumder, P. and Rudnick, E.M.: *Genetic Algorithms for VLSI Design, Layout & Test Automation*, Prentice-Hall PTR, Upper Saddle River, New Jersey (1999).
14) Okamoto, T. and Cong, J.: Buffered Steiner tree construction with wire sizing for interconnect layout optimization, *Proc. ICCAD'96*, pp.44–49 (1996).
15) Sarrafzadeh, M. and Wong, C.K.: *An Introduction to VLSI Physical Design*, McGraw-Hill, New York (1996).

(Received September 25, 2001)
(Accepted March 14, 2002)

Shin'ichi Wakabayashi received the B.E. degree in Electrical Engineering, M.E. and Ph.D. degrees in Systems Engineering from Hiroshima University in 1979, 1981, and 1984, respectively. He was a researcher at Tokyo Research Laboratory, IBM Japan Ltd., in 1984-1988. Since 1988 he has been an Associate Professor in Faculty of Engineering, Hiroshima University. His research interests include VLSI CAD, combinatorial optimization, and evolutionary computation. Dr. Wakabayashi is a member of the Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineers, the Association for Computing Machinery, and the Institute of Electrical and Electronics Engineers.