IEICE TRANS. FUNDAMENTALS, VOL. E80-A, NO. 10 OCTOBER 1997

PAPER Special Section on VLSI Design and CAD Algorithms

Efficient Routability Checking for Global Wires in Planar Layouts**

Naoyuki ISO†, Yasushi KAWAGUCHI†*, and Tomio HIRATA†, Members

SUMMARY In VLSI and printed wiring board design, routing process usually consists of two stages: the global routing and the detailed routing. The routability checking is to decide whether the global wires can be transformed into the detailed ones or not. In this paper, we propose two graphs, the capacity checking graph and the initial flow graph, for efficient routability checking in planar layouts.

key words: *layout design, routability, routing*

1. Introduction

In VLSI and printed wiring board design, most of existing routers use a maze algorithm or a line search algorithm for finding a path in the routing area for each wire. Both algorithms find paths one by one. A major difficulty of such routing algorithms is that already routed wires are treated as fixed obstacles and thus routings for the follwing wires may be blocked by them. To overcome this difficulty, the global routing process has been proposed. There are several methods [1]–[3] for deciding whether every global wire can be transformed into a detailed wire or not. If the transfomation is possible, we can start the detailed routing process. Otherwise, we modify some global wires and check the routability again. Therefore, a fast routability checking is important.

In the case of planar layouts, we can check the routability of wires by comparing *capacity* with *flow* for every *cut*. Cut is a line segment connecting between two visible vertices in the routing area. Capacity of a cut is the maximum number of wires which can cross the cut. Flow of a cut is the number of global wires passing across the cut. Cole and Siegel [1] showed that if the flow do not exceed the capacity for every cut then every global wire can be transformed into a detailed one. They presented a checking algorithm which runs in $O(n \log n)$ time, where n is the number of obstacles (terminals and modules) in the routing area. Their algorithm is too complicated and its implementation seems to be difficult. Leiserson and Maley [3] presented a simpler algorithm and it runs in $O(n^2 \log n)$ time.

In this paper, we propose the capacity checking graph (CCG) for an efficient routability checking. Using the CCG, we can test the routability efficienty. Furthermore we propose the initial flow graph (IFG) for finding flows which are necessary for the use of the CCG. The CCG and IFG were originally proposed in [4] for a routing model with no modules. This paper extends the model so that modules are allowed in the routing area.

The remainder of this paper is organized as follows. In Sect. 2, we define our routing model. In Sect. 3, we define the capacity checking graph for the routing model and present an algorithm for constructing it. We define the initial flow graph in Sect. 4. In Sect. 5, we describe the routing scheme using the CCG and IFG. Some concluding remarks are given in Sect. 6.

2. Routing Model

In this section, we describe our routing model. A routing area consists of a rectangle on a single layer and there is a rectilinear lattice on the rectangle. In the routing area, there are terminals, rectangular modules and wires. The wires interconnect two terminals without crossing each other and do not pass through the other terminals and modules. A terminal connects to the only one wire. Every terminal and every vertex of a module is placed at a lattice point. An obstacle area is the union of the boundary and interior of a module. Outside of the routing area is also an obstacle area. Lattice points on the boundary of an obstacle are called boundary points. We denote the set of boundary points by $\mathbf{V}_b$. $\mathbf{V}_t$ is the set of terminals in the routing area. Let $\mathbf{V}_{t+b}$ denote the union of $\mathbf{V}_t$ and $\mathbf{V}_b$. We denote by $\mathbf{V}_m (\subset \mathbf{V}_b)$ the set of corner points of modules and they are called module points. Let $\mathbf{V}_{t+m}$ denote the union of $\mathbf{V}_t$ and $\mathbf{V}_m$. Let $\mathbf{V}_{b-m}$ be the set of boundary points except the module points. In global routing, every wire is represented by its topology. In detailed routing, it is embedded on the lattice (See Fig. 1). In the following, a "point" means a lattice point. Let a, b be two points in the routing area. When a line segment $\overline{ab}$ do not cross modules or terminals, we say that a and b are visible each other. We also say that the two points are visible or the pair (a, b) is visible.

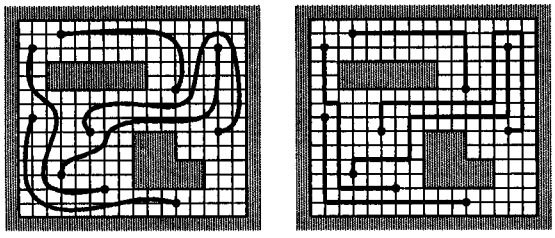
We define a cut as a line segment connecting two visible points in $\mathbf{V}_{t+b}$. Let $p = (x_p, y_p)$, $q = (x_q, y_q)$

Manuscript received February 21, 1997.

†The authors are with the Faculty of Engineering, Nagoya University, Nagoya-shi, 464-01 Japan.

*Presently, with Fujitsu Limited.

**The preliminary version of this paper was presented at ASP-DAC '97.



(a) Global wires. (b) Detailed wires.

Fig. 1 Routing model with rectangular modules.

be two terminals of a cut $\overline{pq}$. The capacity $cap(p,q)$ of the cut $\overline{pq}$ is the maximum number of wires which can pass across the cut $\overline{pq}$, that is, $cap(p,q) = \max(|x_p - x_q|, |y_p - y_q|) - 1$. Given the global routing, we define the flow of cut $\overline{pq}$, denoted by $flow(p,q)$, as the number of the global wires which pass across the cut $\overline{pq}$. The wire which do not cross $\overline{pq}$ topologically does not contribute to the flow. We call the constraint that $cap(p,q) \geqq flow(p,q)$ the capacity constraint for cut $\overline{pq}$.

The following theorem was given by Cole and Siegel [1].

Theorem 1: Given global routing, the global wires can be transformed into detailed ones if and only if the capacity constraint is satisfied for every cut.

In [1], cut is defined as a line segment interconnecting two visible lattice points on the boundaries of distinct objects.

Let l_a^+ (l_a^-) be the line passing through a point a in the routing area with angle 45(−45) degrees. Let b be a point not on l_a^+ or l_a^-. We denote by $rect(a,b)$ the rectangle formed by the four lines l_a^+, l_a^-, l_b^+ and l_b^-.

Consider a point $a = (x_a, y_a)$ and two lines l_a^+, l_a^-. We define an *upper* area of a as the intersection of the upper areas of the two lines. In the same manner, we define *left, right* and *lower* areas of a. When we rotate the coordinate axes to −45 degrees, the new coordinates of a is $(\mu_a, \nu_a) = (\frac{x_a - y_a}{\sqrt{2}}, \frac{x_a + y_a}{\sqrt{2}})$. For $a=(\mu_a, \nu_a)$ and $b=(\mu_b, \nu_b)$, if $\mu_a > \mu_b$, we say that a is larger than b with respect to the (−45)-degree axis. Similary, if $\nu_a > \nu_b$, a is larger than b with respect to the 45-degree axis.

3. Capacity Checking Graph

According to Theorem 1, the routability checking can be done by testing the capacity constraint for every cut. In practice, it is enough to check some particular cuts. For instance, consider a triangle abc which does not include a point of $\mathbf{V}_{t+b}$ in its interior or on the boundary. Suppose that the capacity for cut (a,b) is greater than the sum of the capacities of cut (a,c) and (b,c). Then, even if wires entering the triangle across (a,c) or (b,c) leave it across (a,b), violation of the capacity constraint does not occur at cut (a,b). That is, if the capacity constraint for (a,c) and (b,c) are satisfied, then the checking for (a,b) is redundant. In this section, we define the capacity checking graph whose edges correspond to all the non-redundant cuts.

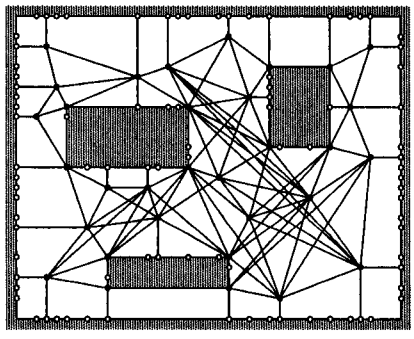


Fig. 2 Capacity checking graph.

3.1 Definition of the CCG

Let a be a terminal or a module point. A projection point of a is a boundary point which is visible from a and is at the intersection of the horizontal or vertical line passing through a and the boundary of an obstacle. Let $\mathbf{V}_p$ be the set of projection points. A terminal has at most four projection points, and a module point has at most two projection points.

For $a \in \mathbf{V}_{t+b}$ and $b \in \mathbf{V}_{b-m}$, $tri(a,b)$ is defined to be the triangle area surrounded by 45 and (−45)-degree lines passing through a and a line which supports the boundary containing b. When b is one of the four corner points of the routing area, $tri(a,b)$ is not unique. We can select any of them.

We define a visibility graph $\mathbf{G}_v = (\mathbf{V}_{t+b}, \mathbf{E}_v)$ as follows. $\mathbf{E}_v$ is the set of visible pairs of points in $\mathbf{V}_{t+b}$. According to Theorem 1, the global wires can be transformed into detailed ones if and only if the capacity constraint for every edge of $\mathbf{G}_v$ is satisfied.

The *capacity checking graph* $\mathbf{G}_c = (\mathbf{V}_c, \mathbf{E}_c)$ is a subgraph of $\mathbf{G}_v$, where $\mathbf{V}_c$ is the union of $\mathbf{V}_{t+m}$ and $\mathbf{V}_p$. $\mathbf{E}_c$ is defined as follows. (i) For $a, b \in \mathbf{V}_{t+m}$, if there is no point $c(\in \mathbf{V}_{t+m})$ visible from a or b in $rect(a,b)$, then $(a,b) \in \mathbf{E}_c$. (ii) For $a \in \mathbf{V}_{t+m}$ and $b \in \mathbf{V}_p$, if b is the projection point of a and there is no point $c(\in \mathbf{V}_{t+m})$ in $tri(a,b)$, then $(a,b) \in \mathbf{E}_c$. (iii) For the remaining cases, (a,b) is not included in $\mathbf{E}_c$.

Figure 2 illustrates an example of the CCG. We have the following theorem. The proof is in [5].

Theorem 2: Given the global wires, if $cap(a,b) \geqq flow(a,b)$ for every edge (a,b) of $\mathbf{G}_c$, then $cap(p,q) \geqq flow(p,q)$ for every edge (p,q) of $\mathbf{G}_v$.

According to Theorem 2, if the capacity constraint for the edges of the CCG is tested, then we can decide the routability of the wires.

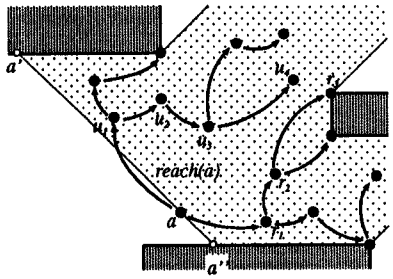


Fig. 3 $reach(a)$ (dotted area) and $tree(a)$.

3.2 Constructing the CCG

For the routing model which does not include modules, it was shown that the CCG can be constructed in $O(n \log n)$ time if n terminals are placed randomly in the routing area[4]. In the rest of this section, we present an efficient algorithm for constructing the CCG for the model which includes modules.

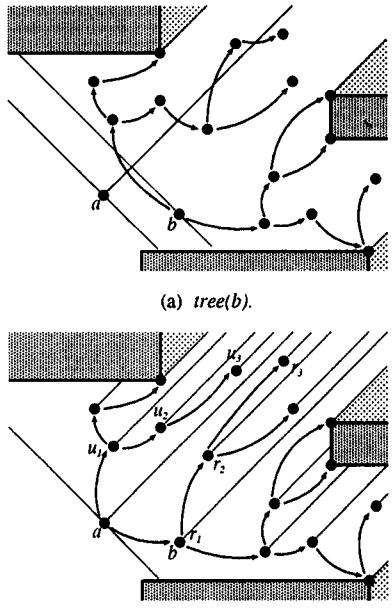
The algorithm consists of two phases. In the first phase, we find the edges of the CCG each having both endpoints in $\mathbf{V}_{t+m}$. In the second phase, we find the edges of the CCG each having one endpoint in $\mathbf{V}_{t+m}$ and the other in $\mathbf{V}_p$. We consider the first phase. Let $a(\in \mathbf{V}_{t+m})$ be a point in the routing area. we draw a (-45)-degree line passing through a. Let a' and a'' be the farthest points on the line visible from a. We draw upper right (45-degree) half lines in the upper area of the scan line which start at every module point. We denote by $reach(a)$ the area in the upper side of the scan line where we can reach from a without crossing a module boundary or a half line starting at a module point (See Fig. 3).

Lemma 1: Let a, b be points in the routing area. We assume that they are visible each other and $\nu_a \leqq \nu_b$. If $(a, b) \in \mathbf{E}_c$, then b is in $reach(a)$.

According to Lemma 1, it is enough for us to search in $reach(a)$ for finding the edges of $\mathbf{E}_c$ which are incident to a and exist in the upper or right area of a. (We call these edges the upper right edges of a.) We define a tree and denote it by $tree(a)$. Its root is a and every point $(\in \mathbf{V}_{t+m})$ in $reach(a)$ is its vertex. A vertex of the tree has at most two children. One child is called an upper child and the other a right child. An upper child and its descendants exist in the upper area of its parent. A right child and its descendants exist in the right area of its parent (See Fig. 3). Actually, $tree(a)$ is a heap search tree used in Computational Geometry.

Lemma 2: Let u_1, r_1 be the upper and right child of a, respectively. Let u_{i+1} be a right child of u_i and r_{j+1} be an upper child of r_j. For each $r_i(u_i)$, the edge connecting a with $r_i(u_i)$ is included in $\mathbf{G}_c$. For other vertex p of $tree(a)$, (a, p) is not included in $\mathbf{G}_c$.

According to Lemma 2, we can find the upper right edges of a by searching in $tree(a)$. The search time is bounded by the number of the edges of the CCG.



(a) *tree(b).*

(b) *tree(a).*

Fig. 4 Updating a tree at a terminal point.

The implementation of the first phase uses the sweep technique. We draw a (-45)-degree sweep line in the routing area and scan it from the upper right corner to the lower left corner. The event points are points of $\mathbf{V}_{t+m}$. During the sweep, we maintain a set of $tree$'s.

Assume that the sweep line stops at a terminal point a. Let $tree(b)$ be the tree in $reach(a)$. We remove the edges of $tree(b)$ which intersect an upper right half line starting at a. Then, we connect between proper vertices so that $tree(a)$ is legal for its definition (Fig. 4). Next, we search into $tree(a)$ and output the upper right edges of a. Note that the time spent for construction $tree(a)$ is proportional to the number of the upper right edges of a.

If a is a module point, we will update $tree$'s as follows. If a is an upper right vertex of a module (Fig. 5 (a)), we first construct $tree(a)$ and search for edges of the CCG just like the above procedure for a terminal point. Next, introducing a copy of a we divide $tree(a)$ into the upper and right sub trees, because $reach(a)$ is divided into two areas by upper right half line starting at a. Assume that a is a lower right vertex of a module (Fig. 5 (b)). We first construct $tree(a)$ and search for edges, and we remove the upper child of a and its descendants from $tree(a)$, since every point in the upper area of a is not included in $reach(a)$ hereafter. Similarly, if a is an upper left vertex of a module, we remove the right child of a and its descendants from $tree(a)$. If a is a lower left vertex of a module (Fig. 5 (c)), the roots of $tree$'s in the upper and right areas of a are connected to a as the upper and right child,

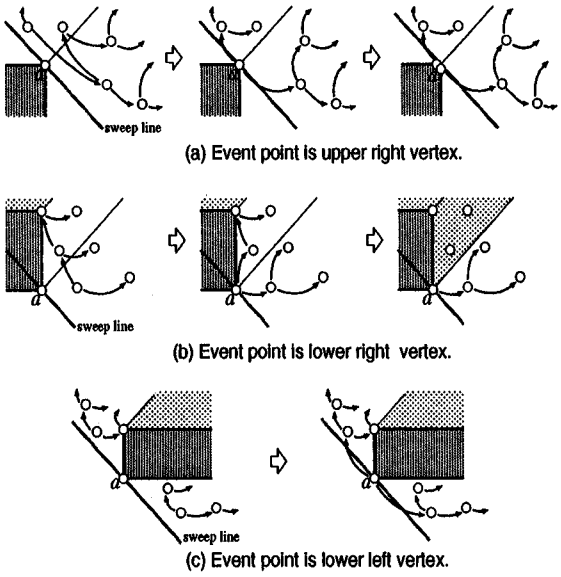


Fig. 5 Updating *tree*'s at a module point.

respectively. It means that the upper and right areas of a will be merged.

We consider the running time of the first phase. It is easy to see that the total time for updating *tree*'s is proportional to the number of edges in the CCG. Therefore, the total running time is $O(|\mathbf{V}_{t+m}| \log |\mathbf{V}_{t+m}| + |\mathbf{E}_c|)$.

Next, we describe the second phase. Let a be a point in $\mathbf{V}_{t+m}$ and let a' be the upper projection point of a. For every point a in $\mathbf{V}_{t+m}$, we check whether an upper edge of a in the CCG was found in the first step. If no edge was found, then we output (a, a') as an edge of $\mathbf{G}_c$. The case for other direction is similar. Therefore, the second phase can be done in $O(|\mathbf{V}_{t+m}| \log |\mathbf{V}_{t+m}|)$ time.

Theorem 3: The capacity checking graph $\mathbf{G}_c = (\mathbf{V}_c, \mathbf{E}_c)$ can be constructed in $O(n \log n + |\mathbf{E}_c|)$ time, where n is the number of terminals and modules.

4. Initial Flow Graph

The following lemma is due to [4].

Lemma 3: Let $\square v_1 v_2 v_3 v_4 (v_i \in \mathbf{V}_{t+b})$ be a convex quadrangle which has no other point of $\mathbf{V}_{t+b}$ in it. Then, $flow(v_1, v_3) + flow(v_2, v_4) = max(flow(v_1, v_2) + flow(v_3, v_4), flow(v_2, v_3) + flow(v_4, v_1))$.

According to Lemma 3, given flows of four edges and one diagonal of a convex quadrangle, we can compute the flow of the other diagonal. We call this operation *flipping*†. Figure 6 is an example of flippings.

The *initial flow graph* (IFG) is a subgraph $\mathbf{G}_f = (\mathbf{V}_c, \mathbf{E}_f)$ of the CCG. $\mathbf{E}_f$ is defined as follows. We assume that $\nu_a \leqq \nu_b$ for an edge (a, b) in $\mathbf{E}_c$. (i) For $a, b \in \mathbf{V}_{t+m}$, if there is no visible point ($\in \mathbf{V}_{t+m}$) from a or b in the intersection of the upper area of a and the left area of b, or in the intersection of the right area of

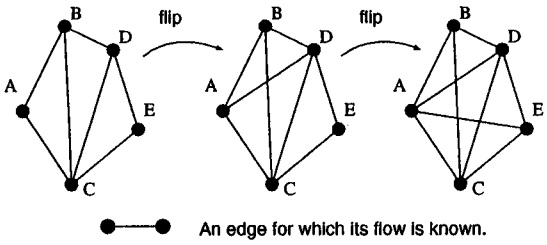


Fig. 6 Flippings. The first flip for $\square ABDC$ gives $flow(A, D)$. The next flip for $\square ADEC$ gives $flow(A, E)$.

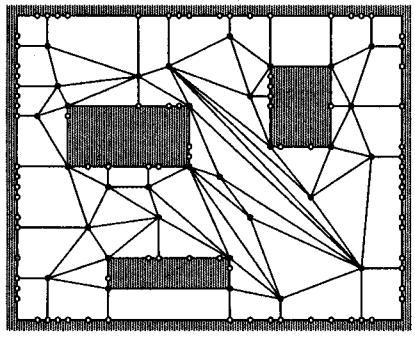


Fig. 7 Initial flow graph.

a and the lower area of b, then (a, b) is in $\mathbf{E}_f$. (ii) For $a \in \mathbf{V}_{t+m}$ and $b \in \mathbf{V}_p$, if b is a projected point of a, then (a, b) is included in $\mathbf{G}_f$.

Figure 7 is the IFG of the CCG shown in Fig. 2. Given a CCG, its IFG $\mathbf{G}_f = (\mathbf{V}_c, \mathbf{E}_f)$ can be constructed in $O(|\mathbf{E}_f|)$ time. It is easy to see that $\mathbf{G}_f$ is planar and the number of the edges $|\mathbf{E}_f|$ is $O(|\mathbf{V}_c|)$. If the flows are given to all edges of $\mathbf{G}_f$, then we can compute the flows of all edges of $\mathbf{G}_c$ using flippings [4]. This computation can be done in $O(|\mathbf{E}_c|)$ time [4].

5. Global Routing

Figure 8 is a flowchart of our global routing process using the CCG and IFG proposed in this paper. It takes $O(n \log n + |\mathbf{E}_c|)$ time to construct the CCG and $O(n)$ time to construct the IFG, where n is the number of terminals and modules. Then we do global routing and find the flows for all edges of the IFG. Next, we find the flows of the edges of the CCG and check the capacity constraints in $O(|\mathbf{E}_c|)$ time. If the terminals are placed randomly, then $|\mathbf{E}_c|$ is expected to be $O(n \log n)$[4] and all operations above are done in $O(n \log n)$ time. If a wire is found to be not routable, we modify some global wires and check the capacity constraints for the involved cuts.

In this routing process, the global routing must

† In Computational Geometry, a diagonal flipping means a triangle conversion. That is, the two diagonal of a convex quadrangle are exchanged. In this paper, a flipping is an operation that calculates the flow of a diagonal and does not remove the other diagonal.

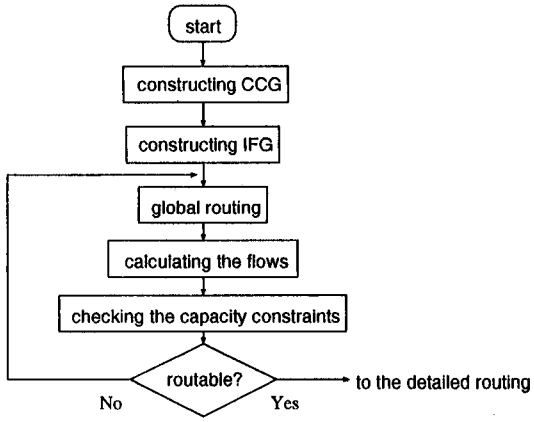


Fig. 8 Flowchart for the routing process.

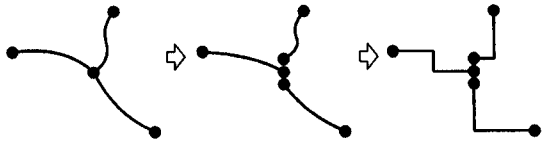


Fig. 9 Transformation of a multi-terminal net into two-terminal nets.

give the flows for the edges of the IFG. One possible way is that we triangulate the routing area using the edges of the IFG and search for the global wires using this triangulation. Then, we can find the flows for the edges of the triangulation. Some existing global routing methods use the Delaunay triangulation for routing the global wires. Repeating flippings to the Delaunay trianuglation, we can transform it to the triangulation constructed from the IFG. Thus, we find the flows for the edges of the IFG.

6. Conclusions

We have proposed two graphs, the capacity checking graph and the initial flow graph, for efficient routability checking.

The model in this paper assumes a one-layer routing. In the case of the multilayer routing, the planar global routing is done on each layer, after the assignment of the nets to layers. When wires of other layers are to be connected, we use through-holes which we can treat as terminals in the routing area.

We also assume that a terminal connects to the only one wire. In multi-terminal routing, we can treat a junction as a terminal connecting two or more wires. We replace it with some adjacent terminals and divide the multi-terminal net into some two-terminal ones. After finishing detailed routing, we connect these adjacent terminals (Fig. 9).

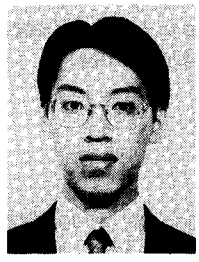
Acknowledgements

The authors wish to thank Makoto Ito of Chukyo University and Xuehou Tan of Tokai University for their helpful comments.

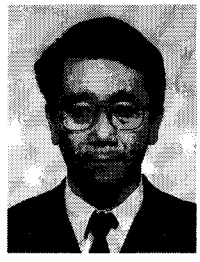
References

[1] R. Cole and A. Siegel, "River routing every which way, but loose," Proc. 25th annual Symposium on Foundations of Computer Science, pp.65–73, 1984.
[2] W.W. Dai, R. Kong, and M. Sato, "Routability of a rubber-band sketch," Proc. 28th ACM/IEEE Design Automation Conference, pp.45–48, 1991.
[3] C.E. Leiserson and F.M. Maley, "Algorithms for routing and testing routability of planar VLSI layouts," Proc. 17th Annual ACM Symposium on Theory of Computing, pp.69–78, 1985.
[4] Y. Kawaguchi, N. Iso, and T. Hirata, "Two graphs for efficient routability checking," IEICE Trans., vol.J80–I, pp.135–142, 1997.
[5] Y. Kawaguchi, N. Iso, and T. Hirata, "Efficient routability checking for routing model with rectangular obstacles," IPSJ DA Symposium '96, pp.189–194, 1996.

Naoyuki Iso was born in Yokohama, Japan on December 8, 1966. He received the B.S. and M.S. degrees from Yamanashi University, Japan in 1990 and 1992, respectively. He has been a Ph.D. candidate and a Research Associate at Graduate School of Engineering, Nagoya University, Japan from 1992 and 1996, respectively. His interests of research include placement and routing algorithm for VLSI and printed wiring board. He is a member of the Information Processing Society of Japan.



Yasushi Kawaguchi was born on December 29, 1971. He receied the B.S. and M.S. degrees from Nagoya University, Japan in 1994 and 1996, respectivery. He joined Fujitsu Limited in 1996. His research interests are in genetic algorighms and routing problem.



Tomio Hirata was born on October 13, 1949. He received the B.E., M.E. and Dr.Eng. degrees in Information Engineering from Tohoku University in 1976, 1978, and 1981, respectively. He joined Toyohashi University of Technology, Aichi, Japan in 1981. From 1986, he joined the Department of Information Engineering, Nagoya University, Japan. He is now a professor in the Department of Electronics. He is interested in graph algorithms and data structures. He is a member of the Information Processing Society of Japan.