Enhanced Heuristic Algorithms K-LAG-V and K-LAG-S for the Constrained Via Minimization Problem

Daisuke Takafuji, Toshimasa Watanabe and Yuji Suga
[1]Graduate School of Engineering, Hiroshima University
1-4-1, Kagamiyama, Higashi-Hiroshima, 739-8527 Japan
Email: {takafuji,watanabe,suga}@infonets.hiroshima-u.ac.jp

Abstract: **CVM** requires finding any layer assignment of wire-segments, whose topology has already been given, so that the total number of vias may be minimized. A given topology of wire-segments is called an *initial wiring layout*. Let k**CVM** denote **CVM** in which k layers are available for routing. In this paper, only rectilinear routing is considered. The subject of the paper is to propose heuristic algorithms *K-LAG-V* and *K-LAG-S* that are enhanced versions of *K-LAG*. Based on experimental results, it is shown that they are promising ones for solving k**CVM** with $k \in \{4, 12\}$.

1. Introduction

A *via* is a through-hole used in connecting wire-segments placed on different layers in layout design of VLSI chips or printed circuit boards. Vias may cause higher production cost, larger routing area, or deterioration of reliability because of increased resistance and delay, and, therefore, designing layouts with the smallest possible number of vias is required. The problem, which decides topology and the layer assignment of wire-segments so that the number of vias in the layout are minimized, is called "the *via minimization* problem".

There are two types of via minimization problems: the unconstrained via minimization problem (**UVM**) and the constrained via minimization problem (**CVM**). **UVM** asks for deciding both topology and the layer assignment of wire-segments at the same time, while **CVM** requires finding any layer assignment of wire-segments, whose topology has already been given. Topology of wire-segments is called a *wiring layout*, and initially given topology of wire-segments is called an *initial wiring layout*. Let k**CVM** denote **CVM** in which k layers are available for routing. In this paper, only rectilinear routing (that is, only horizontal or vertical wiring) is considered.

Some of known results on **CVM** are summarized. For 2**CVM**, polynomial time algorithms [3, 5, 6], or a heuristic one [4] has already been proposed. It is known in [2] that k**CVM** with $k \geq 3$ is NP-hard even if $k = 3$, and a heuristic algorithm is given in [1]. *K-LAG* [1] is known to have very high capability for solving general k**CVM**.

In this paper, we propose heuristic algorithms *K-LAG-V* and *K-LAG-S* for k**CVM**, which are enhanced versions of *K-LAG*, and evaluates their performance through computing experiments.

2. Definitions

The paper considers designing one printed circuit board consisting of several layers, each can be used for routing, and any module is to be placed only on the top or bottom layer. In k**CVM**, we assign each of k layers an identification (ID) number $1, \ldots, k$ from the top to the bottom.

Given a set of terminals, a net is a set of terminals such that they are acyclically connected by wires. A *via-candidate* is a location (to be represented as a point) such that either a via is assigned to it in a given initial wiring layout or a junction of a wire for some net, where a *junction* is a grid point at which the wiring bends. Let us call terminals and via-candidates as endpoints. A *wire-segment* is a portion of a wire between two endpoints such that no intermediate endpoints are included. Let $L(w_i)$ denote the ID number of the layer to which each wire-segment w_i is assigned. We define the number of vias at a via-candidate v_1, $\delta(v_1)$, by $\delta(v_1) = \max\{|L(w_1) - L(w_2)| \mid w_1$ and w_2 are a pair of wire-segments, sharing v_1, such that they are two portions of the same wire $\}$. The number of vias given by a wiring layout σ is defined by the total number of vias at all via-candidates, and is denoted by $n(\sigma)$.

Let $G = (V, E)$ be a graph. A vertex set $V' \subseteq V$ is a *tree component* (rooted at v) of G if the subgraph induced by V' of G is a tree (alternatively we say that V' is rooted at v). A tree component $V' \subseteq V$ is *maximal* if $V' \cup \{w\}$ is not a tree component for any $w \in V - V'$. For disjoint sets $V', W' \subseteq V$, we denote $K(V', W') = \{(v, w) \in E \mid v \in V', w \in W'\}$. And, for any set $V' \subseteq V$, let $Ad(V') = \{u \in V - V' \mid u$ is adjacent to some $v \in V'$ in $G\}$.

Because of space limitation, other definitions are omitted (see [1] for example).

3. A Known Heuristic Algorithm *K-LAG*

In this section, a heuristic algorithm *K-LAG* [1] for k**CVM** is explained.

Some additional definitions are given. An *ECC graph* [1] $G = (V, E)$ of a given wiring layout is defined as follows:

(a) $V \leftarrow V_C \cup V_W$, where V_C is the set of all via-candidates, and V_W is the set of vertices each representing a wire-segment.

(b) E consists of edges defined by (i) or (ii).

(i) A *continuation edge*: one endvertex is representing a wire-segment and the other is representing a via-candidate which is one of endpoints of this wire-segment.

(ii) A *conflict edge*: two endvertices are included in different layers, and they are placed at the same location in the layout projected onto one layer.

Let E_T or E_F denote the set of continuation edges or of conflict edges, respectively, where $E = E_T \cup E_F$.

K-LAG first constructs an ECC graph of a given initial wiring layout, and then repeats deciding layer assignment of wire-segments, represented as vertices of this ECC graph, so that the subgraph induced by these vertices may be a tree. The formal description is given as follows.

Algorithm *K-LAG*
Input: An initial wiring layout σ and a terminating condition

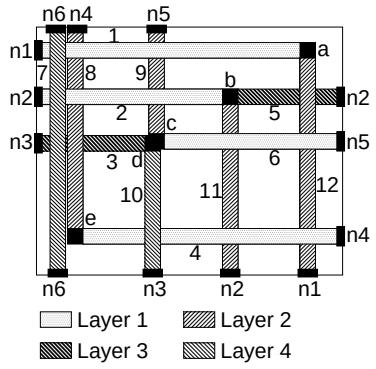

Figure 1. An example of an initial wiring layout σ_0 with $n(\sigma_0) = 6$ in 4-layers, where each ni, $1 \leq i \leq 6$, is a net, each integer i, $1 \leq i \leq 11$, denotes a wire-segment, and $V_C = \{a, b, c, d, e\}$ is a set of via-candidates such that $\delta(b) = 2$ and $\delta(v) = 1$ for any $v \in V_C - \{b\}$.

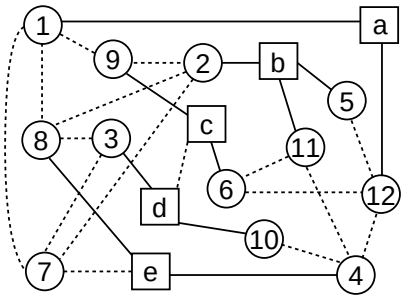

Figure 2. An ECC graph G constructed from the wiring layout σ_0 of Fig. 1. The vertices representing via-candidates or wire-segments are denoted by squares or open circles, respectively. And the continuation edges or the conflict edges are denoted by solid lines or dotted lines, respectively.

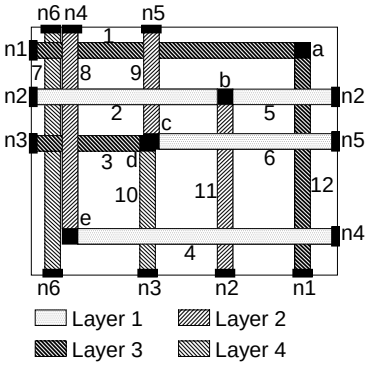

Figure 3. A layer assignment σ_1 obtained by *K-LAG* applied to the wiring layout of Fig. 1, and we have $n(\sigma_1) = 4$ in this case, where $\delta(a) = 0$ and $\delta(v) = 1$ for any $v \in V_C - \{a\}$.

t (a real number);
Output: A wiring layout σ^* for σ;

Step 1. Find all via-candidates in σ.

Step 2. $p \leftarrow 1$. Construct an ECC graph $G = (V, E)$ of σ.

Step 3. Set every vertex as "unvisited". If $p = 1$ then $\sigma^* \leftarrow \sigma$ and $\sigma' \leftarrow \sigma$.

Step 4. Execute (a) through (d) until all vertices of G are changed to "visited".

(a) Choose any "unvisited" vertex r. By applying Function *AS* to G and r, find a maximal tree component $V' \subseteq V$ rooted at r. Set every vertex $v \in V'$ as "visited".
/* Layer assignment of every $w' \in V - V'$ is kept similar to σ'. */

(b) For each $w \in V'$, decide all layers to which w can be assigned or to which w cannot be assigned.
/* This can be done by checking layer assignment of the via-candidate or the wire-segment represented by w' for each $w' \in V - V'$ such that w' is adjacent to w. */

(c) Select layer assignment that minimizes the total number of vias to be assigned to via-candidates represented by vertices in V'.

(d) Update σ^* by checking layer assignment for each $w \in V'$.

Step 5. Compute the *via reduction rate* $D'_v = \frac{n(\sigma') - n(\sigma^*)}{n(\sigma')} \times 100$. If $D'_v > t$ then $\sigma' \leftarrow \sigma^*$ and return to Step 3 else output σ^* and halt.

Function *AS* (arbitrary selecting)

1. $V' \leftarrow \{r\}$.
2. While $Ad(V') \neq \emptyset$, execute (a) and (b).
   (a) Choose any vertex $v \in Ad(V')$.
   (b) If $V' \cup \{v\}$ is a tree component of G, then $V' \leftarrow V' \cup \{v\}$ and update $Ad(V')$, else $Ad(V') \leftarrow Ad(V') - \{v\}$.

Example 1: Given an initial wiring layout in Fig. 1, the ECC graph G shown in Fig. 2 is constructed, and the layer assignment as in Fig. 3 is obtained from Fig. 1 and Fig. 2 by means of *K-LAG*. ■

4. Proposed Algorithms *K-LAG-V* and *K-LAG-S*

4.1 Introducing Functions *SV* and *SC*

The point of *K-LAG-V* or *K-LAG-S* is that we try to select a vertex $v \in Ad(V')$ for enlarging V' in Step 4(a) so that the resulting V' may satisfy the following condition.

(K-LAG-V) V' contains no less vertices representing via-candidates than any vertex set obtained by *K-LAG*.

(K-LAG-S) The tree induced by V' contains no less continuation edges than the tree induced by any vertex set obtained by *K-LAG*.

K-LAG-V or *K-LAG-S* is an enhanced version of *K-LAG* by using Function *SV* or *SC*, instead of Function *AS*, respectively. Functions *SV* and *SC* are described as follows. Their differences from *AS* exist in the following Step 2(a), and the rest is the same. Hence only Step 2(a) is shown.

Function *SV* (priority to selecting via-candidates)

Step 2(a) If $Ad(V') \cap V_C \neq \emptyset$, then select $v \in Ad(V') \cap V_C$ else select $v \in Ad(V')$.

Function *SC* (priority to selecting continuation edges)

Step 2(a) If $K(V', Ad(V')) \subseteq E$ contains at least one continuation edge, then (i) select any continuation edge and (ii) select the vertex $v \in Ad(V')$, which is an end-vertex of the selected edge, else select $v \in Ad(V')$.

4.2 Expected effects of *SV* and *SC*

Let x_1, x_2, y be any three vertices such that $x_1, x_2 \in V_W$, $y \in V_C$, $(y, x_1) \in E_T$ and $(y, x_2) \in E_T$. If both x_1 and x_2 are assigned to the same layer (that is, $L(x_1) = L(x_2)$), then no vias are assigned to y. Even if $L(x_1) \neq L(x_2)$, it is desirable to make $|L(x_1) - L(x_2)|$ as small as possible, since

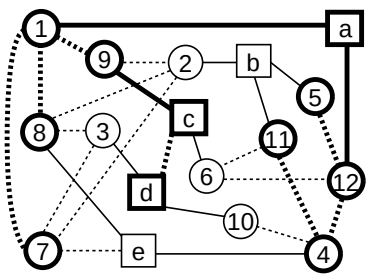

Figure 4. A tree (denoted by bold lines) induced by V' obtained by Function *AS* of *K-LAG*. The tree contains three via-candidates (squares) and four continuation edges (solid lines).

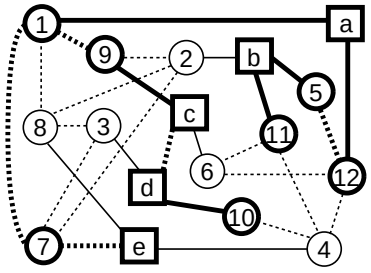

Figure 5. A tree (denoted by bold lines) induced by V' obtained by Function *SV* of *K-LAG-V*. The tree contains five vertices (squares) representing via-candidates, two more than that of Fig. 4.

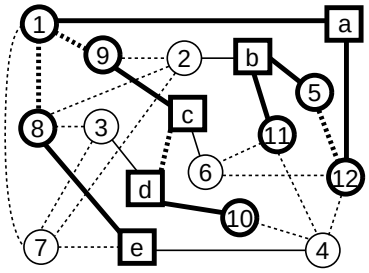

Figure 6. A tree (denoted by bold lines) induced by V' obtained by Function *SC* of *K-LAG-S*. The tree contains seven continuation edges (solid lines), three more than that of Fig. 4.

this may decrease $\delta(y)$. Hence simultaneously changing layer assignment of x_1 and x_2 has higher possibility of avoiding increase in $\delta(y)$. Layer assignment of each vertex $v \in V_C \cup V_W$ is changed only if it is included in V'. If one of x_1 and $x_2 \in V_W$ is not included in V', then $|L(x_1) - L(x_2)|$ may be increased, which may result in increase in $\delta(y)$, and this should be avoided.

Therefore we try to find a tree component V' containing as many elements in $V_C \cup E_T$ as possible, as in Functions *SV* and *SC*.

Example 2: Fig. 4, Fig. 5 or Fig. 6, respectively, shows a pair of a tree component V' rooted at a and the subgraph induced by V', which are obtained by Function *AS*, *SV* or *SC* for the ECC graph of Fig. 2. The tree of Fig. 5 contains more vertices representing via-candidates than that of Fig. 4, and the tree of Fig. 6 contains more continuation edges than that of Fig. 4. ■

It is shown through experiment results that Functions *SV* and *SC* improve capability of algorithms. Comments on another Function *SVC*, a hybrid version of *SV* and *SC*, is given in 5.3.

5. Experiment Results

The existing algorithm *K-LAG* and the proposed algorithms *K-LAG-V* and *K-LAG-S* have been implemented on a personal computer (CPU: Opteron 285, OS: FreeBSD6.2-RELEASE) with the C programming code. By using random numbers, we constructed input problems to k**CVM** such that sets of nets and feasible HVH k-layer wiring layouts ($k = 4, 12$) with 500 to 1,500 nets. In order to compare capability of these algorithms, we pay attention to the via reduction rate D_v (%) defined by:

$$D_v = \frac{n(\sigma) - n(\sigma^*)}{n(\sigma)} \times 100$$

such that the greater percentage, the better, where σ (σ^*, respectively) is a given initial layout (the layout obtained by each algorithm). Also compared are CPU time in second, sizes $|V'|$ and the numbers of via-candidates and continuation edges.

5.1 Input problems and experiment results

(Input problems) The total number of input problems is 300. The number of nets, $\sharp NET$, is $500 \leq \sharp\text{NET} \leq 1,500$, where each net is a 2-terminal net.

(Experimental results) Tables 1 through 3 show a part of experiment results, where each algorithm has a terminating condition $t = 0$.

Via reduction rates $D_v(\%)$ and CPU time (second) over 300 input problems are summarized in Table 1, where the minima, the averages and the maxima are denoted as $min/ave/max$ in the table.

Let $E(G[V'])$ denote the edge set of the subgraph induced by V' of G for any $V' \subseteq V$. We compare the averages of the three ratios $|V'|/|V|$, $|V' \cap V_C|/|V'|$ and $|E(G[V']) \cap E_T|/|E_T|$ over all tree components V' obtained in Step 4(a) of the algorithms *K-LAG*, *K-LAG-V* and *K-LAG-S* in Table 2. Let $\sharp LO$ be the number of iteration of Steps 3 through 5 in *K-LAG*, *K-LAG-V* or *K-LAG-S*, and $\sharp TR$ be the total number of tree components found during execution of each algorithm. Let $Ave(*)$ denote their averages. Table 3 shows the ratio $Ave(\sharp TR)/Ave(\sharp LO)$, which shows the average number of tree components found in each execution of Steps 3 through 5, in the period from the beginning until all vertices are set "visited" in Step 4.

5.2 Observations

The left half of Table 1 shows that the average via reduction rates of *K-LAG-V* and *K-LAG-S* are 0.48% and 2.32% higher than that of *K-LAG*, respectively. On the other hand, CPU time of *K-LAG-V* and *K-LAG-S* is 1.78 times and 1.99 times that of *K-LAG*, respectively, as shown in the right half of Table 1.

Table 2 shows averages of ratios $|V'|/|V|$, $|V' \cap V_C|/|V_C|$ and $|E(G[V']) \cap E_T|/|E_T|$ obtained by *K-LAG-V*, *K-LAG* and *K-LAG-S*, respectively. The results clearly show the effect of adopting Functions *SV* and *SC*. Comparing the results of Table 1 suggests that finding a tree component V' for which $|E(G[V']) \cap E_T|/|E_T|$ is as large as possible gives us highest via reduction rates.

Table 3 shows the ratios $Ave(\sharp TR)/Ave(\sharp LO)$ obtained by *K-LAG*, *K-LAG-V* and *K-LAG-S*, where *K-LAG-S* has the largest ratios. That is, *K-LAG-S* finds the largest number of

Table 1. Comparison of D_v (%) (left half) and CPU time (s) (right half), where minima, averages and maxima are shown as min/average/max, where ♯L is the number of available layers and ♯NET is the total number of nets.

♯L	♯NET	Via reduction rates D_v (%)		CPU time (s)	
		1000	1500	1000	1500
4	*K-LAG*	21.75/23.62/25.90	20.88/22.45/24.01	10.14/**16.55**/27.88	22.43/**49.47**/88.05
	K-LAG-V	21.49/24.01/26.59	21.25/22.76/24.22	8.88/24.47/42.07	35.49/72.31/119.53
	K-LAG-S	22.64/**24.82**/27.54	21.96/**23.58**/25.07	10.96/23.80/48.21	45.50/71.73/109.05
12	*K-LAG*	31.71/33.96/35.98	28.09/29.68/31.12	61.95/**116.86**/208.67	138.98/**288.01**/460.27
	K-LAG-V	32.41/34.19/36.67	28.49/29.85/31.19	138.27/271.65/406.60	261.27/677.64/1115.70
	K-LAG-S	35.07/**36.81**/39.04	29.85/**31.86**/33.52	177.72/317.34/529.05	339.05/753.56/1093.48

Table 2. Comparison of the average of $|V'|/|V|$, $|V' \cap V_C|/|V_C|$ and $|E(G[V']) \cap E_T|/|E_T|$.

♯L	♯NET	$\|V'\|/\|V\|$ (%)			$\|V' \cap V_C\|/\|V_C\|$ (%)			$\|E(G[V']) \cap E_T\|/\|E_T\|$ (%)		
		K-LAG	*K-LAG-V*	*K-LAG-S*	*K-LAG*	*K-LAG-V*	*K-LAG-S*	*K-LAG*	*K-LAG-V*	*K-LAG-S*
4	1000	**47.31**	46.37	44.95	53.40	**56.54**	50.56	34.90	35.24	**36.81**
	1500	**46.24**	45.27	43.85	52.41	**55.27**	49.51	34.20	34.45	**35.87**
12	1000	44.83	**46.01**	44.29	51.88	**62.86**	50.95	24.22	25.82	**28.55**
	1500	44.11	**44.96**	43.38	51.88	**61.93**	50.55	23.66	24.94	**27.69**

Table 3. Comparison of the ratios $Ave(\sharp TR)/Ave(\sharp LO)$.

♯L	♯NET	*K-LAG*	*K-LAG-V*	*K-LAG-S*
4	1000	74.51	78.06	115.19
	1500	104.98	108.06	166.42
12	1000	64.43	88.13	114.32
	1500	87.95	111.67	158.22

tree components in each iteration of Steps 3 through 5 in the period from the beginning until all vertices is set "visited" in Step 4. This may increase the number of execution of the operations (a) through (d) in Step 4, resulting in spending longer CPU time.

As a summary of experiment results, we conclude that *K-LAG-S* is practically useful, because CPU time is 1,093 seconds for input problems of **12CVM** with 1,500 nets.

5.3 Comments on Function *SVC*, a hybrid version of *SV* and *SC*

Comments on Function *SCV*, which is a hybrid version of Function *SV* and *SC*, are given.

Function *SCV* executes the following Step 2(a) instead of Step 2(a) in *SC*.

Function *SCV*

Step 2 (a) If $K(V', Ad(V')) \subseteq E$ contains at least one continuation edge, then select any continuation edge and the vertex $v \in Ad(V')$, which is an endvertex of this selected edge. Otherwise, if $Ad(V') \cap V_C \neq \emptyset$ then select $v \in Ad(V') \cap V_C$ else select $v \in Ad(V')$.

Let *K-LAG-SV* denote the algorithm that uses this function instead of Function *SC*. We have already obtained some results of preliminary experiment, and they are compared with the results by *K-LAG-S*, where both of them has $t = 0.5$. It is shown that both via reduction rates and CPU time of *K-LAG-SV* are worse than those of *K-LAG-S*, and $|E(G[V']) \cap E_T|/|E_T|$ of *K-LAG-SV* is also smaller than that of *K-LAG-S*. The reason may be as follows: selecting via-candidates in the case when $K(V', Ad(V')) \subseteq E$ contains no continuation edges restricts finding desirable tree components V'.

6. Concluding Remarks

Heuristic algorithms *K-LAG-V* and *K-LAG-S* for k**CVM**, which are enhanced versions of *K-LAG* [1], are proposed. Their performance is evaluated through computing experiments, and it is concluded that *K-LAG-S* is more useful in practical situations than the algorithms *K-LAG* and *K-LAG-V*.

Proposing another new algorithm and comparing capability of algorithms for larger input problems are left for future research.

Acknowledgments

The research is partly supported by the Grant-in-Aid for Scientific Research (C) (No. 20500015) of the Ministry of Education, Culture, Sports, Science and Technology.

References

[1] C. C. Chang and J. Cong, "An efficient approach to multilayer layer assignment with an application to via minimization," *IEEE Trans. on Computer-Aided Design of Intergrated Circuits and Systems*, Vol. 18, No. 5, pp. 608–620, May 1999.

[2] K. C. Chang and H. C. Du, "Layer assignment problem for three-layer routing," *IEEE Trans. on Computers*, Vol. 37, No. 5, pp. 608–620, 1988.

[3] R. W. Chen, Y. Kajitani, and S. P. Chan, "A graph-theoretic via minimization algorithm for two-layer printed circuit boards," *IEEE Trans. on Circuits and Systems*, Vol. CAS-30, No. 5, pp. 284–299, May 1983.

[4] Y. Miyata, N. Iso, and Tomio Hirata, "A heuristic via minimization algorithm for two-layer routing," Tech. Rep. VLD99-122, IEICE of Japan, pp. 41–48, Mar. 2000.

[5] R. Pinter, "Optimal layer assignment for interconnect," *J. of VLSI and Computer Systems*, Vol. 1, No. 2, pp. 123–137, 1984.

[6] C.-J Shi, "Solving constrained via minimization by compact linear programming," in *Proc. ASP-DAC '97*, pp. 635–640, 1997.