

An Efficient Approach to Multi-layer Layer Assignment with Application to Via Minimization ¹

Chin-Chih Chang and Jason Cong
Department of Computer Science
University of California, Los Angeles, CA 90095
{cchang, cong}@cs.ucla.edu

Abstract

In this paper, we present an efficient heuristic algorithm for the layer assignment and via minimization problem for multi-layer gridless IC, PCB, and MCM layout. We introduce the notion of the extended conflict-continuation (ECC) graph to represent the multi-layer layer assignment problem. Our algorithm is based on a linear time optimal algorithm that solves a special case of the layer assignment problem when the ECC graph is a tree. For the general layer assignment problem where the ECC graph is not a tree, our algorithm constructs a sequence of induced subtrees in the ECC graph and applies our linear time optimal algorithm to each of the induced subtrees. We have applied this algorithm to the via minimization problem and get very encouraging results. We have achieved 13%-16% via reduction on the routing layout generated by V4R router[13], which is a router known to have low usage of vias. We successfully applied our algorithm to routing examples of over 30,000 wire segments and over 40,000 vias.

1 Introduction

As VLSI technology advances, interconnection and packaging technology have become bottlenecks in the system performance. For advanced IC design, three to five routing layers are commonly used to achieve high-performance, high-density design. Multi-chip module (MCM) technology was developed to increase packing densities, eliminate packaging level interconnections, and to provide more layers for routing. In both the multi-layer IC and MCM designs, the designer or automatic layout tools may use variable widths and spacings for different nets or different segments of the

same net to optimize the performance and density[15], which often results in gridless layout.

Because multi-layer gridless routing is a complex three-dimensional general area routing problem, it is not easy to incorporate different design objectives, such as delay minimization, wire length minimization, via minimization or crosstalk minimization all into a single router and expect it to meet every requirement. Therefore, it is important to perform certain post-layout optimizations (e.g., layer assignment) to help the router to meet the various design constraints and produce better routing solutions.

Traditionally, the post-layout layer assignment problem is discussed in the context of the via minimization problem for two-layer VLSI routing. There are two approaches for the two-layer via minimization problem: constrained via minimization in which only layer assignments can be changed (e.g., [1]-[7]) and unconstrained via minimization (or topological via minimization) in which both the routing topology and layer assignments can be changed (e.g., [8]-[9]).

Although two-layer constrained via minimization on the Manhattan plane is polynomial time solvable, multi-layer constrained via minimization is NP-hard [10][12]. Several heuristic algorithms had been developed (e.g., [10]-[12]).

In this paper, we introduce the notion of the extended conflict-continuation (ECC) graph to represent the multi-layer layer assignment problem. We propose a linear time optimal algorithm that solves the special case when the ECC graph is a tree. For the general layer assignment problem where the ECC graph is not a tree, our algorithm constructs a sequence of induced subtrees from the ECC graph and applies our linear time optimal algorithm to each induced subtree.

2 Problem Formulation

Given an existing valid layout, we first break the layout into wire segments and via candidates. A *wire segment* is a piece of wire assigned to a single layer during the layer assignment. A *via candidate* corresponding to a possible location in the layout where the wire segments of the same net may connect to each other by a via. We extend the concept of a conflict-continuation graph by Pinter[3] so that it can handle the multi-layer problem properly. In our *extended conflict-continuation (ECC) graph*, we have *wire vertices* and *via vertices* to represent the corresponding wire segments and via candidates, respectively.¹ In addition to layer assign-

¹This work is funded by DARPA/ETO under the contract DAAL01-96-K-3600, and a grant from Intel under the 1996-1997 California MICRO program.

¹Chang and Du[10] also introduced via vertices, but our representation differs from theirs in defining conflict edges and considering layer assignments for the via vertices.

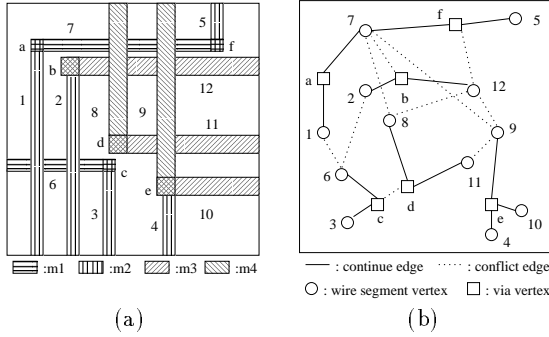


Figure 1: (a) A given layout. (b) The corresponding ECC graph.

ments for wire vertices, via vertices also have layer assignments. The layer assignment for a vertex is an integer m that encodes a pair of integers (l_l, l_u) , $l_l \leq l_u$, indicating that a possible via spans from layer l_l to layer l_u (if $l_l = l_u$, no via is used at this location). Note that we can easily represent vias connecting more than one layer (stacked vias) under this representation.

In our ECC graph, a *continuation edge* between two vertices exists if it connects a via vertex and a wire vertex of the same net which are adjacent to each other. A *conflict edge* exists if the two vertices it connects can not be assigned to certain layers at the same time. If we have conflict edges $e_{u,v}$ and $e_{w,v}$, continue edge $e_{u,w}$, and w as a via vertex, the edge $e_{u,v}$ is redundant and will be removed from the graph, because the constraints set by $e_{u,v}$ is implied by $e_{w,v}$.

Figure 1 shows an example of a layout and the corresponding ECC graph. To demonstrate the representation capability of ECC graph, we make a 3-way wire split and a stacked via in via vertex e . We also assume segments 7 and 12 are too close to be assigned in the same layer and via vertices c and d are too close to have vias at the same time. These explain the existence of edges $e_{7,b}, e_{f,12}$ and $e_{c,d}$. Note that we do not have edge $e_{7,12}$ which is a redundant edge.

For each vertex u , we use a vertex cost array $A_u[m]$ to specify the cost of assigning vertex u to layer m . For each edge $e_{u,v}$, we associate an edge cost matrix $M_{u,v}[m, n]$, which gives the cost for assigning u to layer m and v to layer n at the same time.

Given an ECC graph $G = (V, E)$ and a layer assignment σ that assigns each vertex v into layer $\sigma(v)$, we compute the cost of σ as follows.

$$COST(\sigma) = \sum_{u \in V} A_u[\sigma(u)] + \sum_{e_{u,v} \in E} M_{u,v}[\sigma(u), \sigma(v)]$$

Our goal for the layer assignment problem is to find a layer assignment σ^* with the minimum cost, i.e., $COST(\sigma^*) \leq COST(\sigma)$ for any σ .

For the via minimization problem, the cost comes from the via count and the penalty of layer assignment that causes the design rule violations. The vertex cost A_u for each via vertex u is the via count on that vertex. Any forbidden layer assignment of an edge or vertex will be assigned a cost MAX to the corresponding entry in the vertex or edge cost, where MAX is a number larger than the total cost of any feasible layer assignment.

3 Optimal Algorithm for Trees

It turns out that when the ECC graph is a tree, we can solve the layer assignment problem optimally in linear time. It enables us to optimize a large portion (an induced subtree) of the ECC graph when it is not a tree (as presented in the next section).

We define $ch(u)$ to be the set of child vertices of u and $pa(u)$ to be the parent of u in the tree. For a layer assignment σ on a tree, we define the cost for a layer assignment σ on a subtree rooted at u as $COST_u(\sigma) = A_u[\sigma(u)] + \sum_{v \in ch(u)} (M_{u,v}[\sigma(u), \sigma(v)] + COST_v(\sigma))$. This cost contains three parts: the layer assignment cost for u , the edge costs of edges between u and its children, and the costs of subtrees rooted at the children of u . We define $MC_u[i] = \min_{\sigma} \{COST_u(\sigma) \mid \sigma(u) = i\}$, i.e., it gives the minimum cost for the subtree rooted at u under the condition that u is assigned to layer i . Note that $MC_u[i]$ for a leaf vertex u is simply $A_u[i]$. For a non-leaf vertex u , when the layer of vertex u is fixed to i , we can optimize each subtree rooted at u 's child independently. Therefore, $MC_v[i]$ can be computed recursively by

$$MC_u[i] = A_u[i] + \sum_{v \in ch(u)} (\min_j \{M_{u,v}[i, j] + MC_v[j]\}).$$

Based on this formula, our algorithm consists of a bottom up cost computation and a top down layer assignment. For the bottom up cost computation, we make sure that when MC_u is computed, MC_v has been computed for each child v of u . When computing $MC_u[i]$, for each child v , if j^* is the layer gives $\min_j \{M_{u,v}[i, j] + MC_v[j]\}$, we assign $b_v[i] = j^*$. Therefore, when the parent of vertex v is assigned to layer i , the best assignment for v is layer $b_v[i]$.

After the cost array MC_r is computed for the root r , we know the minimum cost for the whole tree and the best layer assignment of r by checking the minimum cost in array MC_r . We then perform a top down layer assignment according to the b_v array of each vertex v .

We summarize our optimal layer assignment algorithm for trees as follows:

ALGORITHM: K-LAT(K-layer Layer Assignment for Trees)

1. Bottom up computation of MC_v and b_v for each vertex v .
2. For root r , find the assignment with the minimum MC_r .
3. Top down layer assignment: $\sigma(v) = b_v[\sigma(pa(v))]$ for each vertex v .

Theorem 1 *Given an ECC graph $G = (V, E)$ which is a tree, the K-LAT algorithm finds an optimal K-layer assignment in $O(K^4 \mid V \mid)$. It runs in $O(K^2 \mid V \mid)$ if stacked vias are not allowed.*

Details of the algorithm and the proof of Theorem 1 can be founded in [16].

Since K is a constant in our algorithm, we have $O(\mid V \mid)$ linear time algorithm for optimal K-layer layer assignment for trees. In practice, $K = 2$ to 5 for modern ICs, and usually less than 10 for most PCB/MCM designs.

4 An Efficient Heuristic for General Graphs

In general, the ECC graph of a layer assignment problem may not be a tree, but we can build a heuristic algorithm by optimizing a sequence of subproblems in it. If we choose a subset S of the vertices in the ECC graph G and fix the layer assignments for the vertices outside of S , the original layer

assignment problem is reduced to the layer assignment for S . Since we need to consider all the edges among vertices in S which are originally in G , we need to consider the subgraph induced by S . The definition for induced subgraph is shown below.

Definition 1 A subgraph $G' = (V', E')$ is an induced subgraph of $G = (V, E)$ if and only if $V' \subseteq V$ and $E' = \{e_{u,v} \mid u \in V', v \in V' \text{ and } e_{u,v} \in E\}$. When an induced subgraph of G is a tree, it is called an induced subtree.

To extend our K-LAT algorithm to an induced subtree, we simply add the costs of the edges between a tree vertex v and the adjacent non-tree vertices to the layer cost A_v for v , so that the layer cost for each tree vertex will also include the costs due to its relations with its non-tree neighbors. Then we can apply K-LAT algorithm directly on the induced subtree. The resulting algorithm is called K-LATI (K-LAT for Induced subtrees).

Because K-LAT is optimal, K-LATI is also optimal. We have the following corollary:

Corollary 1 Given an ECC graph G , a feasible assignment σ , and an induced subtree $T = (V', E')$ of G , K-LATI algorithm finds a minimum cost layer assignment of T in $O(|V'| + |CUT(V', V - V')|)$ time, where the $CUT(V', V - V')$ is the set of edges connecting V' and $V - V'$.

Proof: Please see [16].

In order to take advantage of K-LATI algorithm, we need to find a sequence of large induced subtrees so that the whole graph is covered by these subtrees.

Our induced subtree finding algorithm starts with an arbitrary vertex r and finds a maximal induced subtree rooted at r . For vertex $v \neq r$, we use $label(v)$ to keep track the number of adjacent vertices of v which is currently in the subtree. We first assign $label(r) = 1$ and $label(v) = 0$ for $v \neq r$ in the graph. Each time our algorithm will add one vertex v with $label(v) = 1$ into the subtree and increase the labels of all the adjacent vertices to v by one. The algorithm stops when there is no more vertex v where $label(v) = 1$. The correctness of the algorithm follows by the fact that since each time we add only one vertex v where $label(v) = 1$, the resulting induced subgraph remains a tree.

We summarize our algorithm for finding maximal induced subtrees as follows:

ALGORITHM: FMIST(Finding Maximal Induced SubTree)

1. $Label(r) = 1$. For each $v \neq r$, $label(v) = 0$.
2. Find a vertex v with label 1, put it into the tree. Increase the labels by one for all its neighbors.
3. Repeat 2. until no vertex with label 1.

The runtime of this algorithm is $O(|V'| + |CUT(V', V - V')|)$, where V' is the maximal induced subtree found by FMIST.

Given an initial feasible layer assignment solution σ with the ECC graph G , our heuristic algorithm contains several passes. In each pass, we cover G with a set of maximal induced subtrees computed by the FMIST algorithm, and apply K-LATI algorithm to get the optimal layer assignment for these maximal induced subtrees one by one. If there is cost reduction in the current pass, we start another pass. Otherwise, we stop the program. In practice, we observe that the FMIST program is able to find very large induced subtrees (11-38% of the entire graph). Therefore, our algorithm is able to optimize a large portion of the layout each time. Our heuristic algorithm is summarized as below.

ALGORITHM: K-LAG(K-layer Layer Assignment for ECC Graph)

1. Use a scheduler find an unmarked vertex r .
2. Run FMIST algorithm with r to get a maximal induced subtree T . Mark all vertices in T .
3. Apply K-LATI algorithm to T .
4. If there exists an unmarked vertex, goto step 1.
5. If there is improvement, unmark all vertices, goto step 1.
6. Stop.

In each pass, each vertex will appear in at least one (and possibly several) induced subtree.

The run time of K-LAG is $O(P \times \sum_{V' \in VP} (|V'| + |CUT(V', V - V')|))$, where P is number of passes and VP is the set of subtrees found in each pass. In practice, we observed that P is usually two or three.

The choice of scheduler can be very flexible and we may use different heuristics to choose the next unmarked vertex. In our implementation, we simply choose the first unmarked vertex in the vertex list.

5 Experimental Results

We have implemented K-LAG algorithm for the K-layer constrained via minimization problem using C++. We use the routing solutions generated by the V4R router [13] to test our algorithms. Stacked vias are not allowed except for the distribution via which bring the terminals to their proper routing layers. Test cases mcc1 and mcc2 are industrial MCM designs provided by MCC. In particular, mcc2 is a supercomputer with 37 VHSIC gate arrays. For further information on these test examples, please refer to [13].

The experimental results reported here were done on an HP9000 workstation with 160M of memory. Table 1 shows the results obtained by K-LAG algorithm.² For all examples, our algorithm consistently finds induced subtrees of large portions of the whole graph. The average subtree sizes are between 11% and 38% of the whole graph.

We can see that for all test cases the via reductions are above 12%. This is a significant reduction: V4R has very low usage of vias in general, as each two terminal net is routed using no more than four vias in V4R. In fact, it was shown in [13] that the average number of vias used by V4R is only 2.5 vias per two-terminal nets.

Table 2 shows the accumulated reductions of K-LAG. We can find that most of the reductions were done in the first few runs of K-LATI. This shows that the convergence of K-LAG is fast.

We have also used an industrial gridless maze based channel router to obtain a 3-layer gridless routing solution of Deutsch's difficult example [17] and applied K-LAG to it under the restriction that each terminal must originate from the middle layer and the vias at the terminal locations need to be counted as well. We reduced 13 out of 325 vias. The reduction is modest since the layout is very compact and we only allow via candidate points on the intersections of horizontal and vertical wire segments.³

²The via numbers for V4R shown here are different from those reported in [13]. In V4R, the distribution vias can be stacked to bring the I/O pins on the surface to any routing layer. Each distribution via is counted as one, no matter whether it is stacked or not. Also, due to an oversight, the distribution vias in a multi-terminal net are over-counted [14](as each multi-terminal net was broken as a set of independent two-terminal-nets). In our via counting routing, we count each distribution via by the number of layers it spans excluding the surface layer. Moreover, we corrected the via over counting problem for multi-terminal nets. The same via counting procedure is used to report the via numbers in columns 6 and 7.

Example	# of layers	# of vertices	# of subtrees	average tree size(%)	# of vias by V4R	# of vias reduced	% of vias reduced	CPU time (sec)	memory (M bytes)
test1	4	4328	148	1647(38.0)	1965	257	13.1	64.5	1.2
test2	4	8606	519	2844(33.0)	4888	704	14.4	579.2	2.6
test3	4	11204	346	3594(32.1)	6298	796	12.6	538.2	3.5
mcc1	4	11649	360	1276(11.0)	5733	771	13.4	171.4	3.2
mcc2	6	67555	300	15375(22.8)	40267	6051	16.1	3552.3	30.6

Table 1: Experimental results by K-LAG algorithm.

example	after 10th tree		after pass 1		after pass 2		after pass 3		Final	
	# vias reduced	CPU time (sec)	# vias reduced	CPU time (sec)	# vias reduced	CPU time (sec)	# vias reduced	CPU time (sec)	# vias reduced	CPU time (sec)
test1	241	3.47	257	32.57	257	64.55	-	-	257	64.55
test2	578	11.18	700	199.52	704	391.67	704	579.23	704	579.23
test3	644	11.41	796	273.53	796	538.42	-	-	796	538.42
mcc1	624	12.64	771	87.3	771	171.39	-	-	771	171.39
mcc2	4490	132.64	6033	1189.32	6051	2370.06	6051	3522.27	6051	3522.27

Table 2: Accumulated numbers of via reduction after the first 10 trees and each pass.(K-LAG algorithm converges for test1, test3, and mcc1 in the first two passes)

6 Conclusion

We have introduced the notation of an ECC graph to represent the layer assignment problem in multi-layer gridless layout. We showed how to use the ECC graph to represent the layer assignment problem for via minimization. We have developed linear time optimal layer assignment algorithms K-LAT and K-LATI for a tree or an induced subtree of the ECC graph. Our experimental results show that K-LAG algorithm consistently finds many large induced subtrees in the ECC graph, and achieves significant via reduction compared to the results of the V4R router, which is known to have low usage of vias. Moreover, our K-LAG algorithm scales well to handle very large multi-layer gridless layout. Currently, we are extending our work to delay and cross-talk minimization.

7 Acknowledgments

We thank Kei-Yong Khoo for providing V4R routing solutions and sharing the information of the V4R via counting procedure. We thank Randall Helzerman for proofreading our early manuscripts for submission.

REFERENCES

- [1] Kajitani, Y., On Via Hole Minimization of Routing on A 2-layer Board, Proc. of the IEEE International Conference on Circuits and Computers ICC 80, 1980, pp. 295-298.
- [2] Chen, R.-W.; Kajitani, Y.; Chan, S.-P., A graph-theoretic via minimization algorithm for two-layer printed circuit boards, IEEE Transactions on Circuits and Systems, May 1983, vol.CAS-30, (no.5):284-299.
- [3] Pinter, R.Y., Optimal Layer Assignment for Interconnect, Journal of VLSI and Computer Systems, vol. 1, no. 2, 1984, pp. 123-137.
- [4] Naclerio, N.J.; Masuda, S.; Nakajima, K., Via minimization for gridless layouts, 24th ACM/IEEE Design Automation Conference Proceedings 1987, pp. 159-165.
- [5] Chang, K.C.; Du, D.H., Efficient algorithms for layer assignment problem, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jan. 1987, vol.CAD-6, (no.1):67-78.
- [6] Kuo, Y.S.; Chern, T.C.; Shih, W.-K., Fast algorithm for optimal layer assignment, 25th ACM/IEEE Design Automation Conference. Proceedings 1988, pp. 554-559.
- [7] Naclerio, N.J.; Masuda, S.; Nakajima, K., The via minimization problem is NP-complete, IEEE Transactions on Computers, Nov. 1989, vol.38, (no.11):1604-1608.
- [8] Hsu, C.-P., Minimum-via topological routing, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Oct. 1983, vol.CAD-2, (no.4):235-246.
- [9] Marek-Sadowska, M., An unconstrained topological via minimization problem for two-layer routing, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, July 1984, vol.CAD-3, (no.3):184-190.
- [10] Chang, K.C.; Du, H.C., Layer Assignment Problem for Three-Layer Routing, IEEE Transactions on Computers, May 1988, vol.37, (no.5):625-632.
- [11] Fang, S.-C.; Chang, K.-E.; Feng, W.-S.; Chen, S.-J., Constrained via minimization with practical considerations for multi-layer VLSI/PCB routing problems, 28th ACM/IEEE Design Automation Conference. Proceedings 1991, pp. 60-65.
- [12] Ahn, K.; Sahni, S., Constrained via minimization, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1993, vol.12, (no.2):273-282.
- [13] Khoo, K.Y. and Cong, J., An Efficient Multilayer MCM Router Based on Four-Via Routing, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.14, no. 10, Oct, 1995, pp. 1277-1290.
- [14] Khoo, K.Y., private communication, 1996.
- [15] Cong, J.; He, L., Optimal wiresizing for interconnects with multiple sources. 1995 IEEE/ACM International Conference on Computer-Aided Design, pp. 568-574.
- [16] Chang, C.C.; Cong, J., An Efficient Approach to Multi-layer Layer Assignment with Applications to Via Minimization, UCLA Computer Science Dept. Tech. report CSD TR-970009, 1997.
- [17] Deutsch, D.N., A 'Dogleg' channel router, Proceedings of the 13th Annual Design Automation Conference, 1976. pp. 425-433.

³ Many researchers assumed that the terminals can originate from any layer without increase the via counts (e.g. [11]). This will significantly increase the possibility of via reduction. Under this assumption, our K-LAG algorithm can reduce 69 vias.