[16] Khoo, K.Y., private communication, 1996.

[17] Hanafusa, A.; Yamashita, Y.; Yasuda, M., Three-dimensional routing for multilayer ceramic printed circuit boards. 1990 IEEE International Conference on Computer-Aided Design, pp. 386–389.

[18] Garey, M.R.; Johnson, D.S., Computers and Intractability: A guide to the Theory of NP-Completeness, W.H. Freeman And Company, New York, 1979.

[19] Cong, J.; Wong, D.F., How to obtain more compactable channel routing solutions, 25th ACM/IEEE Design Automation Conference. Proceedings 1988, pp 663–666.

8 Acknowledgments

We would like to thank the authors of [15] for providing V4R routing solutions and sharing the information of the V4R via counting procedure. We would like to thank Randall Helzerman for proofreading our manuscripts.

References

[1] Hashimoto, A.; Stevens, J., Wire routing by optimizing channel assignment with large apertures, Proc. 8th Design Automation Workshop, June 1971, pp. 155–169.

[2] Kajitani, Y., On Via Hole Minimization of Routing on A 2-layer Board, Proceedings of the IEEE International Conference on Circuits and Computers ICCC 80, pp. 295–298.

[3] Ciesielski, M.J.; Kinnen, E., An optimum layer assignment for routing in ICs and PCBs, ACM IEEE Eighteenth Design Automation Conference Proceedings, 1981, pp. 733–737.

[4] 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.

[5] Pinter, R.Y., Optimal Layer Assignment for Interconnect, Journal of VLSI and Computer Systems, vol. 1, no. 2, 1984, pp. 123–137.

[6] Naclerio, N.J.; Masuda, S.; Nakajima, K., Via minimization for gridless layouts, 24th ACM/IEEE Design Automation Conference Proceedings 1987, pp. 159–165.

[7] 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.

[8] Kuo, Y.S.; Chern, T.C.; Shih, W.-K., Fast algorithm for optimal layer assignment, 25th ACM/IEEE Design Automation Conference. Proceedings 1988, pp. 554–549.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Sung-Chuan Fang; Kuo-En Chang; Wu-Shiung Feng; Sao-Jie Chen, Constrained via minimization with practical considerations for multi-layer VLSI/PCB routing problems, 28th ACM/IEEE Design Automation Conference. Proceedings 1991, pp. 60–65.

[14] 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.

[15] 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.

For the layer assignment problem in VLSI layout, we can always represent the penalties for conflict and discontinuity using a cost matrix for each edge. We assume we have the cost matrix $M_{u,v}[i,j]$ as defined before to represent such penalties. All we need is to add extra cost functions to model the problem we want to solve.

For the crosstalk minimization problem, we could add in the crosstalk penalty for wire segments pairs with crosstalk concern. Assuming that for each pair of wire segments (u,v) in which we need to consider crosstalk, there is a pre-allocated slack $CB_{u,v}$ on the amount of crosstalk allowed for that pair of wire segments. Let us denote $CSK_{u,v}[i,j]$ as the crosstalk between u and v when they are assigned to layer i and j respectively. We can define our cost for crosstalk as:

$$CM_{u,v}(i,j) = \begin{cases} \lambda_{u,v}(CSK_{u,v}[i,j] - CB_{u,v}) & \text{if } CSK_{u,v}[i,j] - CB_{u,v} \leq 0 \\ MAX & \text{otherwise} \end{cases}$$

where $\lambda_{u,v}$ is a pre-assigned weighting coefficient.

This function makes any assignment with crosstalk exceeding the allocated slack to have a large penalty cost. Otherwise, the cost is just the difference between the actual crosstalk and allocated slack times the corresponding coefficient constant. The weight coefficient $\lambda_{u,v}$ allows us to put priority on different pairs according to their importance.

The cost for vertices u and v to be assigned to layers i and j is $M_{u,v}[i,j] + CM_{u,v}(i,j)$. There is no vertex cost for this problem if only crosstalk minimization is considered. Our objective is to minimize the total cost on all the edges.

We can also apply such a layer assignment formulation to do delay optimization. In VLSI layout, wires routed in different layers may have different widths, spacings and RC constants. So the delay of a wire is different when it is assigned to different layers. Assume that we have a delay bound DB_u for each wire segment u and the delay for u be assigned to layer i is $d_u(i)$. We can define cost $DA_u(i)$ as the cost of each segment vertex u as follows.

$$DA_u(i) = \begin{cases} \lambda_u(d_u(i) - DB_u) & \text{if } d_u(i) - DB_u \leq 0 \\ MAX & \text{otherwise} \end{cases}$$

where λ_u is a pre-assigned weighting coefficient to adjust the priority of minimization delay on vertex u.

The cost function gives the maximum value if the delay exceeds the delay bound allocated, otherwise it is the difference between the delay and the delay bound times the corresponding coefficient constant.

The total cost is the sum of the costs over all vertices (DA arrays) and edges (M matrices). The objective is to minimization total cost.

Given these definitions of cost functions, we can apply the K-LAG algorithm to solve the multi-layer layer assignment problem for crosstalk and delay minimization.

7 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, crosstalk minimization and delay minimization. We have presented K-LAT, a linear time optimal layer assignment algorithms for an ECC graph which is a tree, and K-LATI for induced subtrees in an ECC graph. Our K-LAG algorithm utilizes the K-LATI algorithm as the underlying optimization engine and can handle very large designs efficiently, with very good solutions. Our experimental results show that the 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.

Example	number of chips	number of nets	number of pins	number of wires	size of substrate(mm^2)	pitch (um)	number of layers
test1	4	500	1000	1914	22.5 x 22.5	75	4
test2	4	956	1912	3825	30 x 30	75	4
test3	9	1254	2058	4975	37.5 x 37.5	75	4
mcc1	6	802	2495	5519	45 x 45	75	4
mcc2	37	7118	14659	30241	152.4 x 152.4	75	6

Table 1: Test Examples

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

Table 2: Experimental results by the K-LAG algorithm.

select the next unmarked vertex to grow in a clever way, it is possible to achieve further runtime reduction with the same quality of the results.

6 Layer Assignment for Delays and Crosstalk Minimization

As we mentioned in the previous section, the ECC graph is a very general model of the layer assignment problem. With proper definitions of cost functions, our algorithm can be used for the layer assignment problem for crosstalk minimization and delay minimization. In fact, our algorithm will work on any cost function on ECC graph with the following properties:

1. The costs are distributed on vertices and edges and the total cost is the summation of weighted costs over all vertices and edges.
2. The cost of a vertex is purely determined by its layer assignment.
3. The cost of an edge is purely determined by the layer assignment of the two vertices it connects.

example	after 10th tree		after phase 1		after phase 2		after phase 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 3: Accumulated numbers of via reduction after the first 10 trees and each phase.(The K-LAG algorithm converges for test1, test3, and mcc1 in the first two phases)

4. If there exists unmarked vertices, goto step 1.
5. If there is improvement, unmark all vertices, goto step 1.
6. Output the refined layer assignment σ^*.

In each phase, each vertex will appear in at least one induced subtree. But it might appear in more than one subtree. Although we start from an unmarked vertex each time to grow an induced subtree, the induced subtree might also contain a vertex which has marked before. Thus, the same vertex may appear in different induced subtrees. The run time of the K-LAG is $O(P \times \sum_{V' \in VP}(\mid V' \mid + \mid CUT(V', V - V') \mid))$, where P is number of phases and VP is the set of subtrees found in each phase. In practice, we observed that P is usually two or three.

The choice of scheduler is 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 the K-LAG algorithm for the K-layer constrained via minimization using C++. We use the routing solutions generated by the V4R router [15] to test our algorithms. The test examples are shown in Table 1, as reported in [15]. We have added in one column (the fifth column) to show the number of wire segments used by the V4R router in the table. It is the count of the number of rectangles in the routing layers in the V4R MAGIC output files. The test examples test1, test2, test3, and mcc1 are routed in four layers. The example mcc2 is routed in six layers. The stacked vias are not allowed except for the distribution vias 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 [15].

The experimental results reported here were done on an HP9000/770 workstation with 160M of memory. Table 2 shows the results obtained by the K-LAG algorithm. The second column shows the number of vertices in our ECC graph. The third column shows the total number of induced subtrees constructed by the K-LAG algorithm in all phases. The fourth column gives the average size of the induced subtrees constructed by the K-LAG algorithm. The numbers in parentheses give the percentage of the subtree sizes versus the total number of vertices in the graph. 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%. The fifth column shows the numbers of vias in the V4R routing solutions.[3]

The sixth and seventh columns show the number of vias we reduced and the percentage of the reduction. All the reductions are above 12%. This is a significant reduction because 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 [15] that the average number of via used by V4R is only 2.5 vias per two-terminal nets. The CPU times we show are the total optimization times of our program. The last column shows the memory usage of our program.

In fact, most of the via reductions are achieved by optimizing the first few trees. If we do not insist that our algorithm stops only when no more via reduction is possible, we can still get high quality solution in just a few runs of the K-LATI algorithm. Table 3 shows the accumulated reductions after optimizing the first 10 induced subtrees, the accumulated reductions after each phase, and the total number of reductions. The CPU times are also shown. We can see that most of the via reduction is achieved in the first phase of the K-LAG algorithm. This shows that the convergence of K-LAG is fast. If we further enhance the scheduler to

[3] The via numbers shown here are different from those reported in [15]. 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 [16](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 column 5 and 6.

in T. If $label(v) = 1$, we know it has exactly one connection to T that we constructed so far. In this case, we add v to T, increase the labels of v's neighbors by one, and insert any of its neighbors with label 1 into Q. We repeat this process to extract another vertex from Q again. We will stop when Q is empty.

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

ALGORITHM: FMIST(Finding Maximal Induced SubTree)
INPUT: An ECC graph $G = (V, E)$ and a vertex r.
OUTPUT: A maximal induced subgraph which is a tree rooted at r.

1. For each v, $label(v) = 0$.
2. $label(r) = 1$, $parent(r) = r$, push r into queue Q.
3. While Q is not empty do
   Remove a vertex v from Q.
   If $label(v) = 1$
   Add v into the tree T.
   For each u, s.t. $e_{u,v} \in E$, increase $label(u)$ by one.
   If $label(u) = 1$ then $parent(u) = v$, and push u to the queue Q.

Figure 3 shows how this marking process in the FMIST algorithm works. The grey nodes are the vertices we put in the induced subtree, the black nodes are the non-tree nodes with labels larger than 1, and the hash nodes are the nodes currently labeled as 1. Note that in Figure 3(c), when both the vertices 7 and 2 are added to T, then the label of vertex b increases twice and becomes 2, which makes it a non-tree vertex. Similarly, d and 12 become non-tree vertices when 8 is added to T.

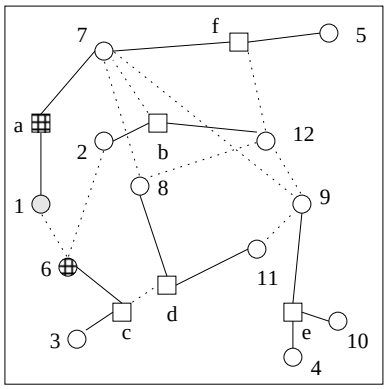
Theorem 2 *Given an ECC graph $G = (V, E)$, the FMIST algorithm computes a maximal induced subtree $T = (V', E')$ in $O(\mid V' \mid + \mid CUT(V', V - V') \mid)$ time.*

Proof: We proof the correctness by induction. It is trivial that a single vertex in the graph G is an induced subtree. Assume $T' = (V'', E'')$ is an induced subtree. When we add a vertex v to T', the labeling process guarantees that there is exactly one vertex in T' with an edge to v. Therefore, the induced subgraph generated by $V'' \cup \{v\}$ is both connected and without cycles. Therefore, it is still a tree. When the algorithm stops, we can not add in any other vertex in the graph without forming a cycle. So the subtree it found is maximal. For the time complexity, we need to traverse each tree edge twice and the edge in the cut set $CUT(V', V - V')$ once. The run time is $O(\mid V' \mid + \mid CUT(V', V - V') \mid)$.

Now we are ready to present our heuristic algorithm for the general layer assignment problem. Given an initial feasible layer assignment solution σ with the ECC graph, we refine σ to minimize the layer assignment cost through a number of phases. In each phase, 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 phase, we start another phase. Otherwise, we stop the program.

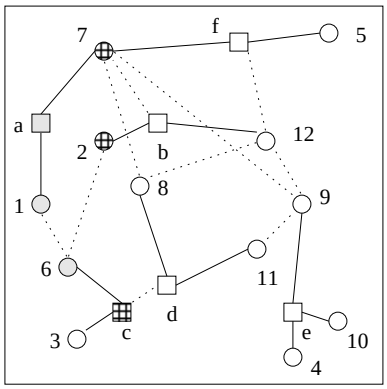
ALGORITHM: K-LAG(K-layer Layer Assignment for ECC Graph
INPUT: An ECC graph $G = (V, E)$, a feasible layer assignment σ, a cost matrix $M_{i,j}$ for each edge $e_{i,j}$, and a cost array A_i for each vertex i.
OUTPUT: A refined layer assignment σ^* with minimal cost.

1. Use a scheduler to get a vertex r.
2. Run FMIST algorithm with r to get a maximal induced subtree T rooted at r and mark all vertices in T.
3. Apply the K-LATI algorithm to T to get an optimal layer assignment of T.



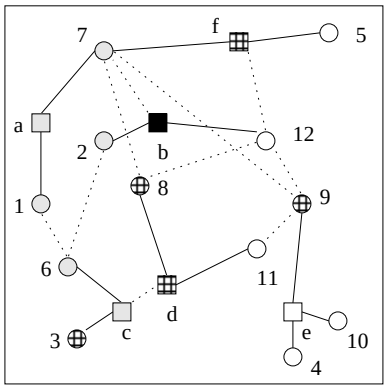
$T = \{1\}$, $Q = \{a, b\}$

(a)



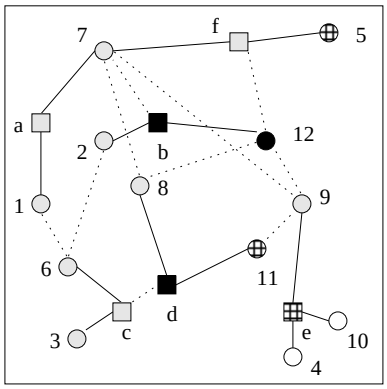
$T = \{1, a, 6\}$, $Q = \{7, 2, c\}$

(b)



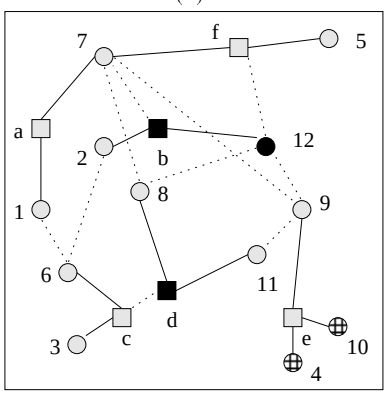
$T = \{1, a, 6, 7, 2, c\}$, $Q = \{f, 9, b, 8, d, 3\}$

(c)



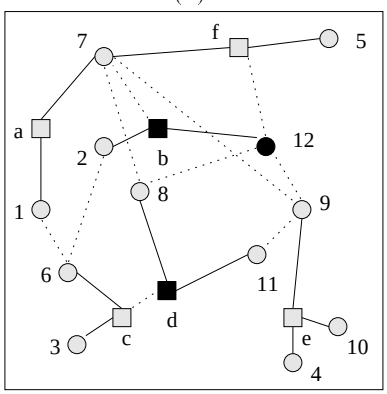
$T = \{1, a, 6, 7, 2, c, f, 9, 8, 3\}$, $Q = \{5, 12, e, 11\}$

(d)



$T = \{1, a, 6, 7, 2, c, f, 9, 8, 3, 5, e, 11\}$, $Q = \{10, 4\}$

(e)



$T = \{1, a, 6, 7, 2, c, f, 9, 8, 3, 5, e, 11, 10, 4\}$, $Q = \{\}$

(f)

Figure 3: Example of finding a maximal induced subtree.

Proof: The optimality follows because we have introduced the costs of the edges to the non-tree vertices in the cost array for each tree vertex, and because K-LAT is optimal. The time complexity of K-LATI is derived as follows. Step 1 takes $O(K|\ CUT(V', V - V')\ |)$ for the most restricted case on stacked vias and $O(K^2|\ CUT(V', V - V')\ |)$ for the least restricted case, as we only need to consider every edge in $CUT(V', V - V')$ once. Step 2 takes $O(|\ V'\ |)$ time as shown in Theorem 1. Therefore, K-LATI runs in $O(|\ V'\ | + |\ CUT(V', V - V')\ |)$ time.

To illustrate the K-LATI algorithm, we use the example in Figure 3(f), which shows the ECC graph of the layout shown in Figure 2(a). The grey nodes forms an induced subtree. We assume the number of layers is four and stacked vias are allowed. Suppose vertex 1 is the root, then the set $\{2, 3, 4, 5, 8, 10, 11\}$ are the leaf vertices with no children. Let us focus on a small subtree rooted at vertex 9. It has two children, vertex 11 and vertex e. Vertex e has two children, vertex 4 and vertex 10. Vertex 9 has a non-tree neighbor 12 in the graph and vertex 11 has a non-tree neighbor d in the graph. So the layer cost array for vertex 9 is $B_9[\,] = \{0, 0, MAX, 0\}$ after we add the costs of the edges to non-tree vertices. Vertex 12 is in layer 3 and conflicts with vertex 9. Similarly, the layer cost array for vertex 11 is $B_{11}[\,] = \{MAX, MAX, 0, 0\}$, because vertex d is in layer $(3, 4)$ and is connected to vertex 11 by a continuation edge. Now we begin the bottom up cost computation. Since vertices 4, 10, 11 are leaf vertices, their subtree costs equal to their layer cost arrays. So $MC_4[\,] = \{0, 0, 0, 0\}, MC_{10}[\,] = \{0, 0, 0, 0\}$, and $MC_{11}[\,] = \{MAX, MAX, 0, 0\}$. Now we compute the cost for vertex e. Assume that the cost for a stacked via is the height it spans, and recall that we encode the layer pair for a via vertex in the following sequence: $(1, 1), (2, 2), (3, 3), (4, 4), (1, 2), (2, 3), (3, 4), (1, 3), (2, 4), (1, 4)$. So, the layer cost array for via vertex e is $B_e[\,] = \{0, 0, 0, 0, 1, 1, 1, 2, 2, 3\}$. Because we can always assign the layers for vertices 4 and 10 to be the same as e and the costs for subtrees rooted at 4 and 10 are all zero, the $MC_e[\,]$ is the same as $B_e[\,]$, $MC_e[\,] = \{0, 0, 0, 0, 1, 1, 1, 2, 2, 3\}$. The corresponding assignment arrays for vertices 4 and 10 are: $m_4[\,] = \{1, 2, 3, 4, 1, 2, 3, 1, 2, 1\}$ and $m_{10}[\,] = \{1, 2, 3, 4, 1, 2, 3, 1, 2, 1\}$. Note that $m_4[5]$ can be either 1 or 2 as either layers 1 or 2 for vertex 4 gives the same cost for the subtree rooted at e. When there are several layer assignments for a vertex with the same minimum cost, we simply pick the one with smaller layer number. This is why $m_4[5] = 1$. Now we can compute the cost of the subtree rooted at vertex 9. Assigning vertex 9 to layer 1, the minimum sum of $MC_{11}[j]$ and $M_{9,11}[1, j]$ is 0 with $j = 3$ or $j = 4$. So we have $m_{11}[1] = 3$. The minimum sum of $MC_e[j]$ and $M_{9,e}[1, j]$ is 0 with $m_e[1] = 1$. We add these two minimum costs to $B_9[1]$ and get $MC_9[1] = 0$. The computation of the minimum cost of assigning vertex 9 to other layers is similar. Eventually, we will get $MC_9[\,] = \{0, 0, MAX, 0\}$, $m_{11}[\,] = \{3, 3, 4, 3\}$, and $m_e[\,] = \{1, 2, 3, 4\}$.

Given the above optimal layer assignment algorithm K-LATI, our algorithm finds a sequence of induced subtrees and apply K-LATI algorithm on them one by one. In order to take advantage of the K-LATI algorithm, we need to find a sequence of a large induced subtrees.

First let us explain how to grow a large induced subtree. It is natural that we might desire to find the maximum induced subtree. However, it was shown in [18] that the problem of finding a maximum induced subtree in a graph is NP-hard. Furthermore, we are not just interested in finding one large subtree, we want to have many large subtrees to cover different portions of the graph. If the maximum induced subtree is unique, but not large enough, this algorithm might not suit our needs. For our purpose, we found that to grow maximal induced subtrees from different vertices is good enough. In practice, we also found the maximal subtree sizes for the ECC graph is large enough. On the average, it covers 11%-38% of the whole ECC graph.

Our induced subtree finding algorithm starts with an arbitrary vertex v and finds a maximal induced subtree rooted at v. It is basically a graph traversal algorithm with proper labeling. Each vertex v in the graph is initially labeled with $label(v) = 0$. We maintain a queue Q for the vertices which are adjacent to some vertex in the induced subtree T under construction. Initially, Q is empty. We choose a vertex r as the root, label it with 1, and add it to T. Once a vertex is added to T, we increase the label of all its neighbors by one. Once a vertex has label 1 after the label increase, we put it into Q. Then we extract the vertices from Q one by one. If the extracted vertex v has label larger than 1, we know that v is a neighbor for two or more vertices in T. In this case, vertex v is discarded because if we add it to T, we would create cycle(s)

be stacked. In the most restricted case, vias can connect only to adjacent layers, $l(v) = 2K - 1$. For the least restricted case, there can be $K(K+1)/2$ possible layer assignments for the via vertex.

Step 1 takes $O(K \mid V \mid)$ for the most restricted case and $O(K^2 \mid V \mid)$ for the least restricted case. For Step 2, each edge $e_{u,v}$ requires examing $l(u)l(v)$ edge costs for the computation of MC_u. Since the number of edges in the tree is $\mid V \mid -1$, the complexity of Step 2 is $O(K^2 \mid V \mid)$ for the most restricted case and $O(K^4 \mid V \mid)$ for the least restricted case. Step 3 takes $O(K)$ for the most restricted case and $O(K^2)$ for the least restricted case, and Step 4 takes $O(\mid V \mid)$ time. So, this algorithm runs in $O(K^2 \mid V \mid)$ for the most restricted case and $O(K^4 \mid V \mid)$ for the least restricted case. Since K is a small integer constant ($K = 2$ *to* 5 for modern ICs, and usually less than 10 for most PCB/MCM designs), we have $O(\mid V \mid)$ linear time algorithm for optimal layer assignment for trees.

4 An Efficient Heuristic for General Graphs

In general, the ECC graph of a layer assignment problem may not be a tree. In this section, we shall first show that the K-LAT algorithm is also applicable to the induced subtrees in a general ECC graph. Then we shall use it as the basis for our heuristic algorithm to optimize a sequence of induced subtrees in a general ECC graph. Let us first introduce the concept of induced subtrees.

Definition 1 *A subgraph $G' = (V', E')$ is an* induced subgraph *of $G = (V, E)$ if and only if $V' \in V$ and $E' = \{e_{i,j} \mid v_i \in V', v_j \in V'$ and $e_{i,j} \in E\}$. When an induced subgraph of G is a tree, it is called an induced subtree.*

Definition 2 *Given an ECC graph G and a layer assignment σ, the layer assignment problem for an induced subgraph G' is to find a layer assignment σ'^* s.t. $\forall v \in V - V', \sigma'^*(v) = \sigma(v)$ and $COST(\sigma'^*)$ is minimum.*

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. To utilize our optimal algorithm K-LAT, we are interested in an induced subgraph which is a tree. This is our reason for considering induced subtrees.

The extension of the optimal K-LAT algorithm for trees to induced subtrees is quite simple. We need to only add costs of the edges between a tree vertex v and the adjacent non-tree vertices to the layer cost array 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. After such modification of the vertex cost arrays, we can apply the K-LAT algorithm directly. Our optimal algorithm for induced subtrees can be summarized as follows.

ALGORITHM: K-LATI(K-LAT for Induced subtrees)
INPUT: An ECC graph $G = (V, E)$, an induced subtree $T = (V', E')$, a feasible layer assignment σ, Cost matrices $M_{i,j}$ for each edge $e_{i,j}$, and cost array A_i for each vertex i.
OUTPUT: A minimum cost layer assignment for the induced subtree T.

1. For each $v \in T$, $B_v[i] = A_v[i] + \sum_{x \in V - V'} M_{v,x}[i, \sigma(x)], 1 \leq i \leq l(v)$
2. Invoke K-LAT on T with the cost array A_v replaced by B_v for each vertex v.

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, the K-LATI algorithm finds a minimum layer assignment of T in $O(\mid V' \mid + \mid CUT(V', V - V') \mid)$ time, where the $CUT(V', V - V')$ is the set of edges connecting V' and $V - V'$.*

Therefore, we can transform the minimization on the summation over several independent terms into a summation over the minimization of each term. This leads to Eq. 3. In Eq. 4, we itemize the minimization over all possible value of j. Pulling out the constant term $M_{u,v}[i,j]$ and applying the definition of $MC_v[j]$, we get Eq. 5.

The above equations basically tell us how to compute the MC_u recursively from the layer cost array A_u, the edge cost Matrix $M_{u,v}$ and the MC_v of each child v of u. Intuitively, it just says that to compute the $MC_u[i]$, we need to minimize the summation of the cost of a subtree rooted at v and the edge cost between u and v for each child v of u.

Based on this formula, our algorithm consists of bottom up cost computation and top down layer assignment. For the bottom up cost computation, we need to make sure that when MC_u is being computed, for each child v of u, MC_v has been computed. To satisfy this condition, we choose to use the postorder traversal of the tree (other orders are also possible). If $MC_v[i]$ is achieved with $\sigma(u) = j^*$ for $v \in ch(u)$, we record j^* in an array m_v.

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 MC_r. Then we go through a top down layer assignment process to compute the optimal layer assignment of each vertex in the tree. Because we have recorded in the m_v array for each vertex v on what layer the vertex v should be assigned to give the minimum cost once its parent's optimal layer assignment is known, a top down traversal simply assigns layers $m_v[i]$ to vertex v if the parent is known to have layer i.

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

ALGORITHM: K-LAT(K-layer Layer Assignment for Trees)
INPUT: An extended conflict-continuation graph G = (V, E) which is a tree. A cost matrix $M_{i,j}$ for each edge $e_{i,j}$. A cost array A_i for each vertex i.
OUTPUT: A minimum cost layer assignment.

1. For each v, $MC_v[i] = A_v[i], 1 \leq i \leq l(i)$
2. For each vertex u in the postorder traverse of the tree
   For each layer $1 \leq i \leq l(u)$
   $MC_u[i] = MC_u[i] + \sum_{v \in ch(u)}(min_{1 \leq j \leq l(v)}\{M_{uv}[i,j] + MC_v[j]\})$

   For each $v \in ch(u)$, let j^* be the layer number of v which give the minimum value for $MC_u[i]$, $m_v[i] = j^*$.
3. For root r of the tree, find g^* which gives the minimum cost among $MC_r[g]$'s $(1 \leq g \leq l(r))$. $\sigma(r) = g^*$.
4. For each vertex v on the preorder traverse of the tree, let u be the parent of v, $\sigma(v) = m_v[\sigma(u)]$.
5. Return σ.

Theorem 1 *Given an ECC graph $G = (V, E)$ which is a tree, the K-LAT algorithm finds an optimal K-layer assignment in $O(\mid V \mid)$.*

Proof: The optimality follows by induction. First, it is trivial to see that the computation of the minimum subtree cost for each layer at the leaves are optimal. For the induction part, suppose that the cost array MC_v for each child v of u is optimally computed, according to Eq. 5. Then the MC_u is also optimally computed. So we can conclude that the MC_{root} is optimally computed by the K-LAT algorithm. Therefore, the top-down layer assignment will give us an optimal layer assignment of the tree.

For the complexity of this algorithm, we first discuss the number of possible layers $l(v)$ per vertex v. For a wire segment vertex, $l(v) = K$. For a via vertex v, this number depends on how many layers the via can

$$COST(\sigma) = \sum_{i \in V} A_i[\sigma(v_i)] + \sum_{e_{i,j} \in E} M_{ij}[\sigma(v_i), \sigma(v_j)]$$

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.

Now we can easily explain how to choose the value of MAX. It should be larger than the cost of any feasible assignment, such that any infeasible assignment will have a larger cost to any feasible assignment. For the via minimization problem, it is sufficient to set $MAX = K \mid V \mid$.

3 Optimal Algorithm for Trees

Although all the vertices and edges costs are defined in terms of the layer assignment of adjacent vertices, our objective is the summation over all the layer assignment costs of vertices and edges. A local change of layer assignment, however, usually has chained effects to enforce restrictions on layer assignment of remote vertices, which may have a great impact on the total cost. Therefore, a simple-minded local refinement algorithm may not have a global view to see all the chained effects and often produces poor results.

We want to find an efficient algorithm which can take advantage of the locality of the cost computation but also provides an efficient way to compute the impact on total cost when local optimization of layer assignment is made.

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.

Before we propose the optimal layer assignment algorithm for trees, we define some notation. Recall that the possible layer numbers for vertex v are integers from 1 to $l(v)$ under our encoding. The value of $l(v)$ depends on whether v is a segment vertex or via vertex and how many layers a stacked via could span. The cost matrix $M_{u,v}$ for edge $e_{v,u}$ is an $l(v) \times l(u)$ matrix. The layer assignment cost array A_v for vertex v is a linear array of size $l(v)$. We define $ch(u)$ to be the set of child vertices 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 $MC_v[i]$ for a leaf vertex v is simply $A_v[i]$. For a non-leaf vertex u, it can be computed as follows:

$$\begin{aligned}
MC_u[i] &= min_\sigma\{COST_u(\sigma) \mid \sigma(u) = i\} && (1)\\
&= min_\sigma\{A_u[i] + \sum_{v \in ch(u)} M_{u,v}[i, \sigma(v)] + COST_v(\sigma)) \mid \sigma(u) = i\} && (2)\\
&= A_u[i] + \sum_{v \in ch(u)} min_\sigma\{(M_{u,v}[i, \sigma(v)] + COST_v(\sigma)) \mid \sigma(u) = i\} && (3)\\
&= A_u[i] + \sum_{v \in ch(u)} min_j\{min_\sigma\{(M_{u,v}[i, j] + COST_v(\sigma)) \mid \sigma(u) = i, \sigma(v) = j\}\} && (4)\\
&= A_u[i] + \sum_{v \in ch(u)} (min_j\{M_{u,v}[i, j] + MC_v[j]\}) && (5)
\end{aligned}$$

The reduction from from Eq. 1 to Eq. 2 is based on definitions of $COST_u(\sigma)$. In Eq. 2, because the layer assignment of u is fixed, the layer assignments for its children are independent of each other. They can be optimized independently.

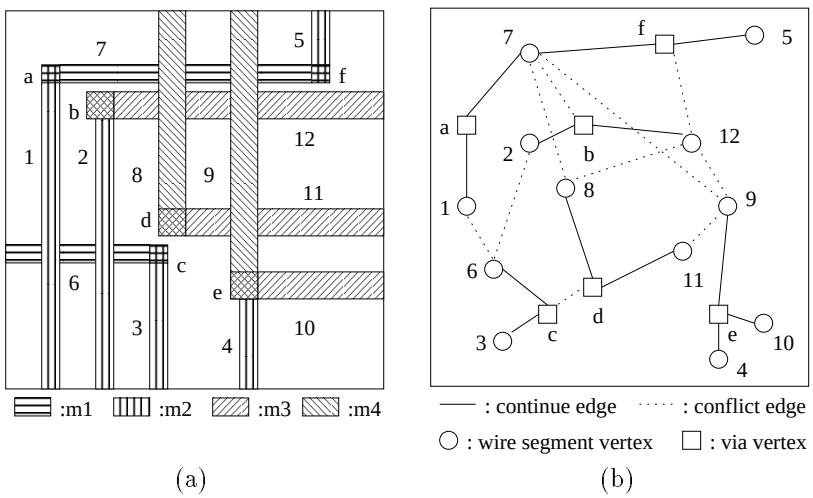


Figure 2: (a) a given layout. (b) the corresponding ECC graph.

distinguish the different cases (see the details of cost functions below).

In our algorithm, we use cost functions to represent the concepts of conflict and continuation. We shall first describe how to use the cost functions for the via minimization problem in this section. It is also possible to define suitable cost functions to solve the layer assignment problems for crosstalk and delay minimization, which will be explained in Section 6.

We define the cost functions for the via minimization problem as follows: For each edge $e_{i,j}$, we associate a cost matrix $M_{i,j}[m,n]$, which gives the cost for assigning v_i to layer m and v_j to layer n at the same time. The cost matrix $M_{i,j}$ will capture the penalty on the conflicting layer assignment between vertices i and j. If there is a conflict for the assignment, the entry will be MAX, otherwise, it will be 0, where MAX is a large number to be defined later. The conflicts may be characterized by the the spacing rule violation, overlapping of wires of different nets, or disconnect of wires or vias of the same net which should be connected. This scheme is very general, for the cases when two vertices conflict on only some assignments, we just fill in the corresponding entries with MAX and others with 0.

For each vertex i, we also associate a cost array $A_i[m]$ of size $l(i)$ to specify the costs of assigning vertex i to layer m. Clearly the cost array for a via can be used to specify whether stacked vias are allowed or not, how many layers can be stacked, etc. For example, if no stacked vias are allowed, for each via vertex i, $A_i[(l_l, l_u)] = 0$ if $l_l = l_u$, 1 if $l_u - l_l = 1$, or MAX if $l_u - l_l > 1$. We can also allow stacked vias up to m layers and count each stacked via spanning s layers with cost $s-1$, by assigning cost for $A_i[(l_l, l_u)] = l_u - l_l$ if $l_u - l_l \leq m$, or MAX if $l_u - l_l > m$.

Note that multi-way splitting wires do not cause any problem in our representation. Much previous work on constrained via minimization can only handle up to 3-way splits (cf. [5] [8]), but we can handle any number of splits. The reason is that via cost is associated with a via vertex only, so we do not need to worry about splitting the cost into wire segment vertices.

We may also replace the cost function with more complex ones. For example, we can put layer constraints on a certain net or part of it such that it can not be assigned into these forbidden layers. Our representation can also handle via minimization for gridless routing where wire segments have variable widths and spacing requirements on the same or different layers. We just need to consider different conflicts on each layer and put them into the cost functions.

Given a layer assignment σ which assigns each vertex v into layer $\sigma(v), (1 \leq \sigma(v) \leq l(v))$, we compute the cost of σ as follows.

v_j if and only if v_i and v_j are in the same net and they are connected to each other. Assigning v_i and v_j to different layers will require a via to be inserted. The set E^f is the set of conflict edges, an edge $e_{i,j} \in E^f$ exists between vertices v_i and v_j if and only if v_i and v_j are in different nets and they can not be assigned to the same layer. This graph is sufficient for the representation of the two-layer via minimization problem. But, it is not suitable for multi-layer representation[1] (see the example in [12]).

We extend the concept of a conflict-continuation graph so that it can handle the multi-layer problem properly. In our extended conflict-continuation (ECC) graph, the vertex set contains via vertices in addition to wire segment vertices.[2] Each via vertex is a possible location to insert a via. It connects two or more wire segments of the same net in the given layout. It may overlap with some wires, but our formulation can represent these conflicts. A via vertex can also have a layer assignment. The layer assignment for a via is a pair of integers $(l_l, l_u), l_l \leq l_u$, indicating that a via spans from layer l_l to layer l_u. Note that we can easily represent vias connecting more than one layer (stacked vias) under this representation. If no stacked vias are allowed, we restrict $l_u \leq l_l + 1$. For the sake of simplicity in later explanation, the integer pair (l_l, l_u) is encoded into one integer. When we say that we assign the via vertex to some layer j, we mean that j will be decoded back into the integer pair. For a K-layer layer assignment problem, we encode (i, i) to i for $1 \leq i \leq K$, $(j, j+1)$ to $K + j$ for $1 \leq j \leq K - 1$, and so on. Such encoding is based on the ordering where integer pairs with smaller $l_u - l_l$ will be ordered before the integer pairs with larger $l_u - l_l$. If two integer pairs have the same $l_u - l_l$, the one with smaller l_l is ordered before the other one. Using this encoding, if stacked vias are only allowed to span up to s layers, valid integer pairs are encoded one-to-one into integers from 1 to $\sum_{0 \leq i \leq s-1}(K - i)$. Let $l(v)$ denote the number of possible layers for vertex v. If v is a wire segment vertex, $l(v) = K$. If v is a via vertex can span up to s layers, $l(v) = \sum_{0 \leq i \leq s-1}(K - i)$. Thus, for any vertex v, the possible layer assignments map one-to-one to integers from 1 to $l(v)$.

In our ECC graph, the continuation edge set contains only via-to-segment continuation edges. A continuation edge between two vertices exists if it connects a via vertex and a wire segment vertex of the same net which are adjacent to each other. The conflict edge set contains segment-to-segment, segment-to-via, and via-to-via conflict edges. A conflict edge exists if the two vertices it connects can not be assigned to the same layer. A conflict edge $e_{u,v}$ between the wire segment vertex u and another vertex v is redundant and can be removed if u connects to a via vertex w which also has a conflict edge to v. This is because u and w are connected by a continuation edge, the layers which via vertex w spans must contain the layer assigned to u. Since the conflict edge $e_{w,v}$ puts the constraint that the layer(s) which w spans will not overlap with the layer(s) occupied by v, the constraint between u and v is also implied.

Our representation is very general and it can handle multi-layer gridless layout in advanced IC and MCM designs. It accommodates different widths of wires and vias in the same layer or different layers as well as different wire-to-wire, wire-to-via and via-to-via spacing in the same layer or different layers. It also handles stacked vias when allowed by technology. The specific design rules are used as input for constructing the ECC graph.

To give an example for our representation, let us consider the layout shown in Figure 2(a). Wire segments are labeled from 1 to 12, and via candidates are labeled from a to f. Note that e is a stacked via connecting $m2$ to $m4$. The corresponding vertices are shown in Figure 2(b). The solid lines are continuation edges and the dotted lines are conflict edges. We assume that wire segments 7 and 12 are too close to be put in the same layer. So, there are conflict edges $e_{7,b}$, $e_{7,12}$, $e_{f,12}$. Note that we did not put an edge between vertices 7 and 12, as it would be a redundant conflict edge and can be eliminated because of the existence of conflict edge between via vertex b and segment vertex 12. If vertex b does not have a conflict layer assignment with vertex 7, nor does vertex 12. We also assume that via vertex c and d are too close to be put in the same layer, e.g. if c connects layers $m1$ and $m2$, and d connects layers $m2$ and $m3$ at the same time, there will not be enough spacing on layer $m2$. So, in this situation, we need a conflict edge $e_{c,d}$. For the case that they may share some layers but not all layers, we will still introduce a conflict edge but use the cost function to

[1] However, we can show that this graph can be used if no stacked vias are allowed and the layout will not put two vias too close.

[2] Chang and Du[12] also introduced via vertices, but our representation differs from theirs in defining conflict edges and considering layer assignments for the via vertices.

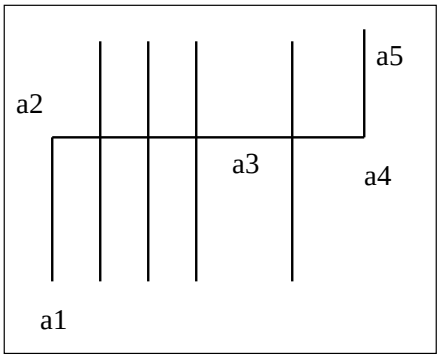


Figure 1: Wire Segmentation

a large portion of the layout optimally each time, which leads to highly optimized layer assignment solutions. The application of our layer assignment algorithm to constrained via minimization is very successful. We have achieved 13%-16% via reduction on the routing solutions generated by the V4R router[15], which is a router known to have low usage of vias. Our algorithms can handle very large designs with very good solutions; we successfully applied our algorithm to routing examples of over 30,000 wire segments and over 40,000 vias. Finally, our layer assignment algorithm can be also used for delay and crosstalk minimization in high-performance IC, PCB and MCM designs.

2 Problem Formulation

Given a K-layer layout solution, we assume that each net is divided into a set of wire segments. Each wire segment is assigned with a unique label $i \in N$. No vias are allowed within a wire segment. Therefore, vias can only be used to connect different wire segments of the same net. The designer may specify the segmentation of wires to impose certain layout constraints and control the trade-off between the flexibility and complexity of the layer assignment problem. For example, Fig. 1 shows a simple layout. Assume that the wires connecting points $a1$, $a2$, $a4$ and $a5$ are of the same net, and all the other vertical lines are of different nets. A natural way of segmenting the wires is to break the nets at the points where the horizontal and vertical wires meet (cf. [2]). In this case, we will break the net which contains $a1$-$a5$ into $(a1, a2)$, $(a2, a4)$, and $(a4, a5)$. Another possible choice is to break the nets whenever there is enough space to insert vias (cf. [5]). Suppose that there is enough space only at point $a3$ to put a via between $a2$ and $a4$. In this case, we will further break $(a2, a4)$ into $(a2, a3)$ and $(a3, a4)$. In the first case, we have less flexibility to insert vias. But the problem size is smaller because we have fewer number of segments to consider. In the second case, we get more freedom to insert vias, but it also produces more wire segments for the optimization problem. Depending on the design, the designer may prefer one option to another, based on the consideration of flexibility they want and the complexity they can handle. There are also other possible ways to break nets. For example, we might want to restrict the maximum length of a single wire segment. It is also possible that we want to restrict the minimum length such that we do not to break wires too often. All these choices will impact the flexibility and complexity. Our algorithm can handle any given wire segmentation. When no segmentation is given, we choose the first segmentation method described above.

Before explaining our formulation, we first introduce the concept of a conflict-continuation graph $G = (V, E)$ used by Pinter[5]. A *free run* is a maximal piece of wire that does not overlap any other wire, and can accommodate at least one via. A *wire segment* is a piece of wire connecting two free runs. Each vertex $v_i \in V$ represents wire segment i. The edge set $E = E^c \cup E^f$ represents two kinds of relationships among wire segments. The set E^c is the set of continuation edges. An edge $e_{i,j} \in E^c$ exists between vertices v_i and

solution too much in terms of wire length and routability, but still provide enough flexibility for further optimization of vias, crosstalk, and delay.

In this paper, we present an efficient algorithm for multi-layer layer assignment. We will focus on the application of this algorithm to the multi-layer constrained via minimization problem. Nevertheless, since of our formulation of the problem is very general, we can also apply our algorithm to the multi-layer layer assignment problem for crosstalk and delay minimization with proper choice of cost functions.

Traditionally, the layer assignment problem is discussed in the context of the via minimization problem for two-layer VLSI routing. Vias are used for the connections of wire segments in different layers in the VLSI layout. The existence of vias often reduces the manufacturing yield, degrades the circuit performance, and increases layout area (more difficult to compact routing solutions, e.g. refer to [19]). Therefore, minimizing the number of vias without affecting the routability is a useful and important problem in layout optimization.

There are two approaches for the two-layer via minimization problem: constrained via minimization [1]-[9] and unconstrained via minimization (or topological via minimization) [10]-[11]. The constrained via minimization problem optimizes an existing routing solution and is restricted to changing the layer assignment for the wire segments only. Therefore, it is also referred to as the layer assignment problem. Topological via minimization, on the other hand, computes both the topologies and the layer assignments of all the nets to minimize the overall via count. However, topological via minimization may effect routability considerably and is usually not used in practice. Moreover, the two-layer topological via minimization problem was shown to be NP-hard [11].

For the two-layer constrained via minimization problem, if routing is restricted to be on a Manhattan plane, polynomial time optimal algorithms exist for this problem. These algorithms transform the problem into the maximum cut problem for planar graph which is polynomial time solvable [2][4][5][8]. But, if routing is not restricted to be on a Manhattan plane, the two-layer constrained via minimization problem was also shown to be NP-hard [9].

For the multi-layer constrained via minimization problem, Chang and Du first proved that the three-layer constrained via minimization problem is NP-hard for routing on a Manhattan plane [12]. They also proposed a heuristic algorithm which checks vias one by one and does a local search up to n levels on the via-crossing graph to eliminate the via being checked (in their implementation n is only two). Fang, *et al.* [13] proposed a two-phase heuristic algorithm. They use heuristic ordering and backtracking to assign the layers of wire segments one by one. Then, they do local perturbations to eliminate vias greedily. But, when one via is eliminated, other vias might be introduced. Ahn and Sahni [14] proved that the three-layer HVH constrained via minimization problem is also NP-hard, they proposed a track-by-track heuristic algorithm for layer assignment in the HVH constrained via minimization problem.

However, the methods in the existing work all suffer from one or more of the following problems:

1. They can handle only a fixed number of layers.
2. They assume a grid-base routing solution.
3. They can not produce good solutions due to very limited range of local search.
4. They can not scale to large designs efficiently.

All the experimental results on constrained via minimization reported in the literature are on small test cases: the number of vias is about only two or three hundred.

In this paper, we introduce the extended conflict-continuation (ECC) graph to represent the multi-layer layer assignment problem. This representation is very general; we can handle any number of layers for the multi-layer layer assignment problem. Furthermore, we can handle the layer assignment problem for gridless layout. 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 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. Our experiments shows that the average number of vertices of the induced subtree is between 11%-38% of the total number of vertices of the ECC graph. Therefore, our algorithm is able to refine

An Efficient Approach to Multi-layer Layer Assignment with Applications to Via Minimization *

Chin-Chih Chang and Jason Cong
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095

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. Our experiments show that the average number of vertices of the induced subtree found by our algorithm is between 11%-38% of the total number of vertices of the ECC graph. Therefore, our algorithm is able to refine a large portion of the layout optimally each time, which leads to highly optimized layer assignment solutions. 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[15], 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. Finally, our layer assignment algorithm can also be used for delay and crosstalk minimization in high-performance IC, PCB, and MCM designs.

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 the packaging level of interconnections, and provide more layers for routing. It mounts the bare chip on a common substrate, and the substrate consists of multiple routing layers used for chip-to-chip interconnection. Depending on what type of MCM technology, the interconnection layers it provides could vary from a few layers up to even 50 layers[15][17]. 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, 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 to help the router to meet the various design constraints and produce better routing solutions. The multi-layer layer assignment problem has become an interesting topic for post-layout optimization because it will not change the original layout

*this work is funded by DARPA/ETO under the contract DAAL01-96-K-3600