On Crossing Minimization Problem

Hsiao-Feng Steven Chen

Avant! Corporation
46871 Bayside Parkway
Fremont, CA 94538
email: schen@avanticorp.com

and

D. T. Lee†
Department of Electrical and
Computer Engineering
Northwestern University
Evanston, IL 60208
e-mail: dtlee@ece.nwu.edu

Abstract

In this paper we consider a problem related to global routing post-optimization: the crossing minimization problem (CMP). Given a global routing representation, the CMP is to minimize redundant crossings between every pair of nets. In particular, there are two kinds of CMP: constrained CMP (CCMP) and unconstrained CMP (UCMP). These problems have been studied previously in [Groe89], where an $O(m^2n)$ algorithm was proposed for CCMP, and in [MS95], where an $(mn^2+\xi^2)$ algorithm was proposed for UCMP, where m is the total number of modules, n is the number of nets, and ξ is the number of crossings defined by an initial global routing topology. We present a simpler and faster $O(mn)$ algorithm for CCMP and an $O(n(m+\xi))$ time algorithm for UCMP. Both algorithms improve over the time bounds of the previously proposed algorithms. The novel part of our algorithm is that it uses the plane embedding information of globally routed nets in the routing area to construct a graph-based framework and obtain a good junction terminal assignment that minimizes the number of crossings.

†Supported in part by Avant! Corporation and by the Office of Naval Research under the Grant No. N00014-97-1-0514.

1 Introduction

Given a specification of a *circuit*, the *VLSI* layout problem is to determine the geometric coordinates of the circuit components and their wiring in the plane such that the overall area of the resulting chip is minimized while all manufacturing process rules requirements are satisfied[Oh86, Leng90]. A specification of a circuit usually consists of a set of circuit components (logical gates, macro cells, etc.), called *modules*, and a collection of *netlists* that describe how the modules are connected. Due to the increasing complexity of modern *VLSI* design, the layout problem has traditionally been decomposed into several subproblems: *floorplanning*, *placement*, *global routing* and *detailed routing*. After floorplanning and placement are done, *global routing* is performed to determine the topology of the wiring or a rough course for each net over a certain data structure representing the routing area. In general-architecture layout designs, and designs applied to sea-of-gate chips or multi-chip modules (MCM), a global router breaks the chip area into a uniform two dimensional array of global routing cells (GRC) [LM84, CSW90, CC91, KP95]. Fig. 1(a) shows a chip layout divided

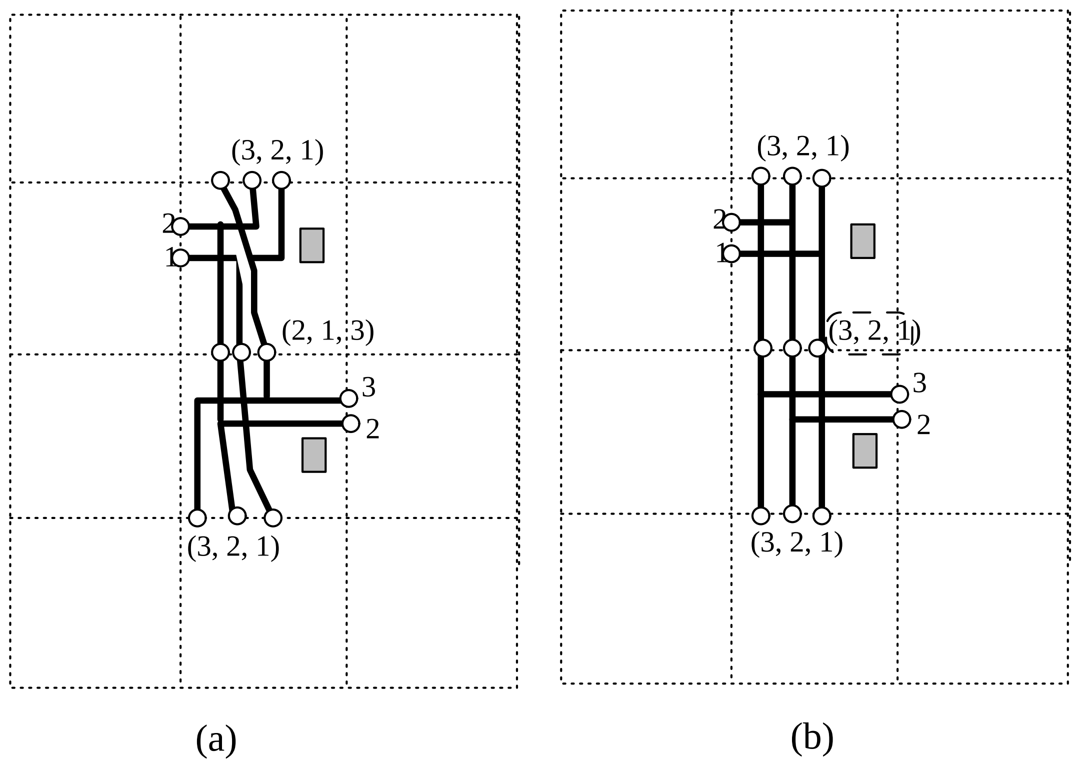


Figure 1: (a) A layout with 4×3 GRC's, (b) A better pin positions

into four by three GRC's where the routings inside GRC[2, 2] and GRC[3, 2] are highlighted. In a building block layout style[CHK+83], the routing area is decomposed into *elementary regions* by adding either vertical or horizontal noncrossing dissecting lines out of each corner of the modules,

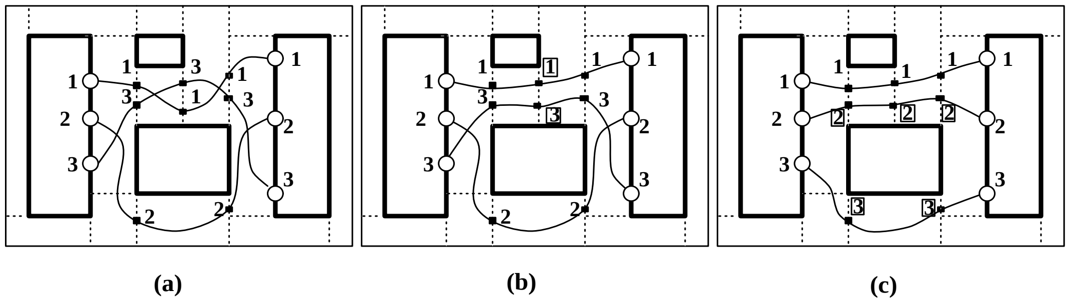


Figure 2: (a) a global routing (b) after CCMP (c) after UCMP minimization.

such that each elementary region is a rectangle. In general, nets are global-routed *sequentially* and *separately* based on different objective functions such as total wire length, maximum delay, minimum congestion, etc. Pins at the boundaries between elementary regions must be placed before detailed routing is performed. After global routing is completed, *detailed routing* is performed, in which specific tracks in each elementary region or GRC are assigned to each net.

Determining pin placement on a net-by-net basis may result in a rip-up and reroute, as the pin positions good for one net may be bad for another net. For example, if nets 1, 2, 3 in Fig. 1(a) are independently routed, the pin positions could be determined in such a way that four vertical tracks are needed and thus, the detailed routing may fail. Fig. 1(b) shows that by changing the pin ordering on the boundary between GRC$[2,2]$ and GRC$[3,2]$ the number of tracks needed is reduced. In order to guard against detailed routing failure, crossing pin assignment or channel pin assignment (CPA) must be performed[KP95, Cong91, CW91] to minimize the numbers of horizontal or vertical tracks used. The following two optimization problems related to crossings of a given global routing were proposed[Groe89, MS95]: (1) minimizing the total number of crossings by eliminating redundant ones and (2) *appropriately* distributing the inevitable crossings such that no region contains more crossings than it is allowed. The first problem is referred to as *crossing minimization problem* (CMP) and the second, *crossing distribution problem* (CDP). In [Groe89] Groenveld addressed CMP and proposed an $O(m^2n)$ time algorithm, where m is the number of the modules and n is the number of nets. In [Groe89], the homotopy of each net in the global routing is not changed during the minimization. That is, the sequence of regions through which a net passes is preserved. In the following, we refer to this kind of CMP as *constrained* CMP (CCMP). Besides CMP Marek-Sadowska and Sarrafzadeh[MS95] introduced the CDP. They presented an $O(mn^2 + \xi^2)$ time algorithm for CMP and an $O(m\xi^{3/2})$ time algorithm for CDP, where ξ is the number of crossings implied by the global routing. Their CMP is *different* from that in [Groe89] in the sense that the homotopy of a *given* net is allowed to change, while the *overall homotopy* of *all* nets remains the same. We therefore refer to this problem as *unconstrained* CMP (UCMP). Note that in CCMP, after applying the minimization, a pair of nets may have more than one crossing (see Fig. 2(b)). In UCMP, each pair of nets has at most one crossing. For instance, nets 2 and 3 may be *uncrossed* by changing their individual homotopy, while their overall homotopy is maintained. That is, the sequence of regions passed through by these two nets remains the same (see Fig. 2(c)). Note also that the running time of algorithm for UCMP depends on ξ, the number of crossings associated with an initial pin placement.

In this paper we present two faster algorithms: an $O(mn)$ time algorithm for CCMP and an $O(n(m+\xi))$ time algorithm for UCMP, improving over the time bounds in previous results. We concentrate only on the CMP instances with two-terminal nets. Transformations from a general global routing CMP instance with multi-terminal nets to one with two-terminal nets have been proposed in [Groe89, MS95]. In the following sections, we formulate the CMP's, provide some preliminaries, and present our algorithms for CCMP and UCMP of a set of 2-terminal nets, followed by a conclusion.

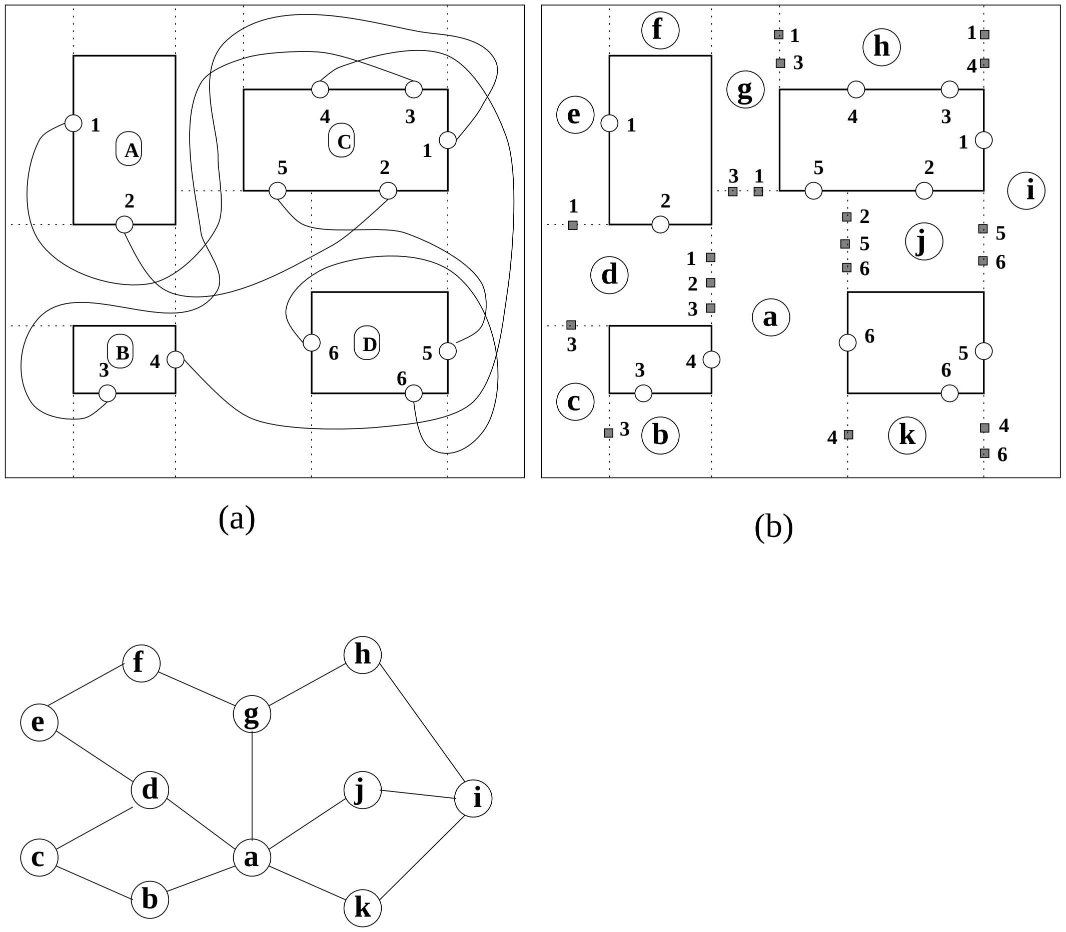
2 Preliminaries

We use the general building block layout style as the global routing problem instance. As mentioned earlier, the results of the layout style can be easily generalized to either channel-based or general-architecture design layout. A CMP instance of a globally routed layout is shown in Fig. 3(a). The thick lines of this figure represent the boundaries of the modules and the dotted lines extending from the corners of the modules represent the boundaries of elementary regions. There are four modules in the figure: module M_A, M_B, M_C and M_D, eleven *elementary* regions: region R_a through R_k, and six 2-terminal nets: net N_1 through N_6. The collection of regions and modules forms a planar map of size $O(m)$, where m denotes the number of modules.

Definition 1 *A placement $\mathcal{P}$ is a planar map in which:*

1. *each edge is either a horizontal or vertical line segment.*
2. *the boundary $\partial\mathcal{P}$ of $\mathcal{P}$ is a rectangle.*
3. *each vertex has degree 2, 3 or 4.*
4. *each face of the graph corresponds to either a module or a region.*

Definition 2 *A region adjacency graph $\mathcal{G}$ is the dual graph of a placement $\mathcal{P}$ where each vertex corresponds to a region of $\mathcal{P}$ and there is an edge $(R_i, R_j) \in \mathcal{G}$ if regions R_i and R_j are adjacent in*



(a) (b)

(c)

Figure 3: A global routing and its region adjacency graph

$\mathcal{P}$.

The region adjacency graph $\mathcal{G}$ for placement $\mathcal{P}$ in Fig. 3(a) is shown in Fig. 3(c).

Definition 3 *A global routing $\mathcal{W}$ with respect to a placement $\mathcal{P}$ is a set of netlists such that associated with each net N_i, there is a list $W_i =< R^i_{\alpha 0}, R^i_{\alpha 1}, \ldots, R^i_{\alpha k_i} >$, where each $R^i_{\alpha j}, 0 \leq j \leq k_i$ is a region through which the 2-terminal net passes, and the two terminals are in regions $R^i_{\alpha 0}$ and $R^i_{\alpha k_i}$, respectively.*

Each W_i in the global routing $\mathcal{W}$ represents a homotopy of net N_i. The homotopy of net N_1 in Fig. 3(a) is represented as $W_1 =< R_e, R_d, R_a, R_g, R_h, R_i >$. In this homotopy representation there is no specification about *where* N_i crosses the boundaries of each region relative to other nets $N_j, i \neq j$. To fully specify the *topology* of each netlist in the global routing, an arbitrary crossing point on a boundary edge is provided for each net that crosses the boundary edge (see below for an example of crossing assignment at each boundary edge). The point of boundary crossing is called a *junction terminal* (i.e. the pins) and the two endpoints of each net are called the *module terminals*. These two types of terminals are depicted by dark squares and white circles in Fig. 3(b) respectively.

3 Solving CCMP

In this section, we study the first type of CMP, i.e., CCMP. Consider a global routing $\mathcal{W}$ and the corresponding adjacency graph $\mathcal{G}$. The goals of CCMP are the following: (1) preserving the original routing homotopy of each net, and (2) determining a best ordering for the junction terminals of the nets on each edge $e \in E(\mathcal{G})$ so that no *redundant* crossing among the nets exists. Note that the input size $|\mathcal{W}|$ is $O(mn)$. Fig. 4(a) shows a layout with seven modules and six globally routed nets, while Fig. 4(b) shows the corresponding adjacency graph $\mathcal{G}$. The junction boundaries between regions are depicted by dashed lines. By "tightening" *loosely* routed net N_i with both its terminals fixed[LM85, DKJ90, DDS91, CL96] one obtains its *rubber-band equivalent* (RBE). For a layout with m modules, the RBE of net N_i, denoted RBE(N_i), for all nets can be obtained in time linear in $|W_i|$[CL96] or $O(|\mathcal{W}|) = O(mn)$ for n nets, after $O(m \log m)$ preprocessing time. Let $\Delta \subseteq$ RBE(N_i)$\cap$ RBE(N_j). If Δ does not contain a module terminal of RBE(N_i), then the two RBE segments of RBE(N_i) connected to Δ are referred to as *incident segments* of RBE(N_i) on Δ.

Definition 4 RBE(N_i) and RBE(N_j) are said to have a *crossing,* if $\Delta \subseteq$ RBE(N_i)$\cap$ RBE(N_j) satisfies the following condition. Let segments $s_1, s_3 \in$RBE(N_i), and segments $s_2, s_4 \in$RBE(N_j)

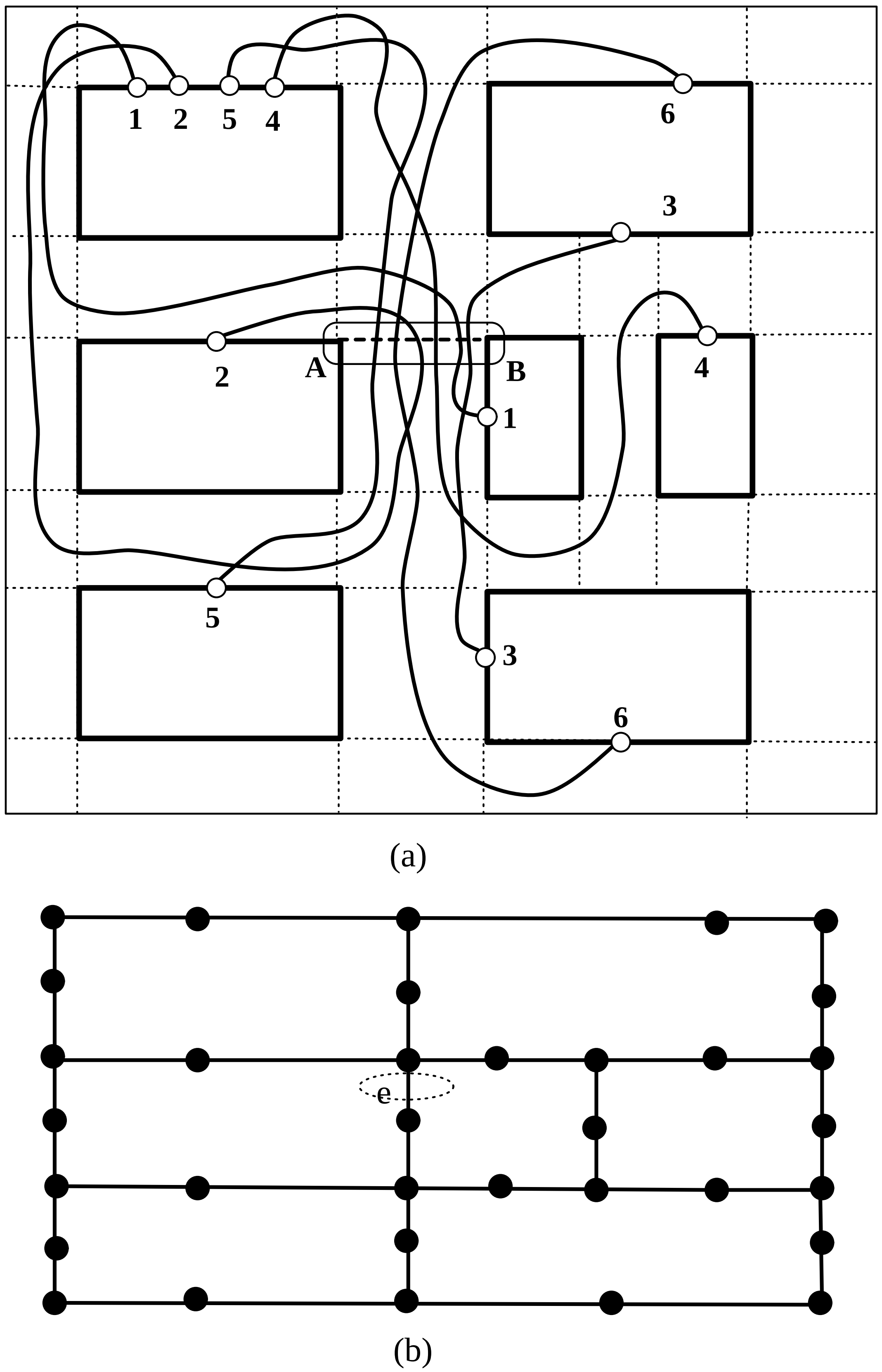


Figure 4: Six nets pass crossing AB

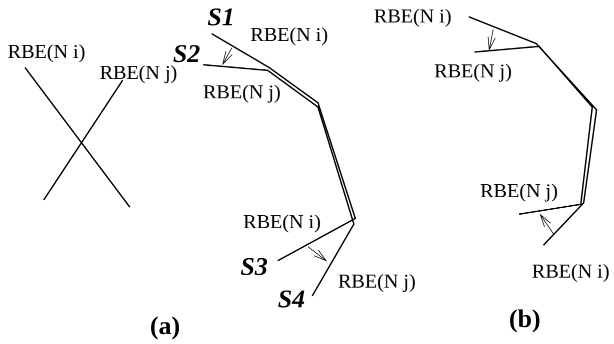


Figure 5: (a) RBE intersection and (b) non-intersection

be incident segments on Δ respectively. The four incident segments on both paths are *cyclically ordered* as (s_1, s_2, s_3, s_4) (Fig. 5).

We define variable $c_{i,j}$, $1 \leq i, j \leq n$ as the number of times RBE(N_i) crosses RBE(N_j). Let the number of crossings determined by a given junction terminal ordering σ be $\mathcal{X}$ and $\mathcal{X}_{min}$ be the minimal number of crossings. Then

$$\mathcal{X}_{min} = \min_{\forall \sigma} \mathcal{X} = \sum_{1 \leq i < j \leq n} c_{i,j}$$

It is not difficult to see that the number of crossings determined by RBEs is minimum. However, using an algorithm based on RBEs to find a junction terminal ordering may not be as efficient.

We shall construct a *bundle graph* in which the vertex set contains *macro* nodes, each representing a boundary edge, and two macro nodes are connected by a *bundle* edge, if there exists at least one net passing through both boundary edges. A bundle edge represents one or multiple overlapping nets. More precisely let us assume that the boundary edges of an elementary region are ordered in the counterclockwise manner, such that the interior of the region always lies to the left as these edges are traversed (Fig .6). We write $\overline{AB}$ to denote an ordered edge *directed from A to B*, and $e(A, B)$ to denote an undirected edge connecting A and B. Suppose that edge $e(A, B)$ is shared by regions $\overline{ABCD}$ and $\overline{BAEF}$. For each edge $\overline{AB}$ we define three boolean variables, $\overline{AB}_l$, $\overline{AB}_r$ and $\overline{AB}_o$, which are set to 0 initially. $\overline{AB}_l$ is set 1 if there is a net N_i passing through edge $\overline{AB}$ and its *left* boundary edge $\overline{DA}$. These nets are referred to as β-list associated with $\overline{AB}_l$, denoted $\beta^+(\overline{AB}_l)$. $\overline{AB}_r$ is set 1 if there is a net N_j passing through edge $\overline{AB}$ and its *right* boundary edge $\overline{BC}$. $\overline{AB}_o$ is set 1 if there is a net N_k passing through edge $\overline{AB}$ and its *opposite* boundary edge $\overline{CD}$. For region $\overline{ABCD}$ we have $\overline{AB}_l \equiv \overline{DA}_r$, $\overline{AB}_r \equiv \overline{BC}_l$ and $\overline{AB}_o \equiv \overline{CD}_o$. All the nets in the collection of the lists $\beta^+(\overline{AB}_l)$, $\beta^+(\overline{AB}_o)$, $\beta^+(\overline{AB}_r)$, denoted $\mathcal{B}^+$-list($\overline{AB}$), are called *outgoing nets* of $\overline{AB}$. Similarly all the nets in the collection of the lists $\beta^+(\overline{AE}_l)$, $\beta^+(\overline{EF}_o)$, $\beta^+(\overline{FB}_r)$, denoted $\mathcal{B}^-$-list($\overline{AB}$), are

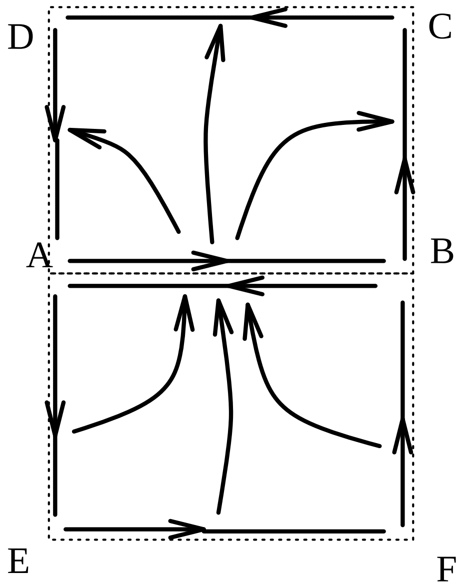


Figure 6: Region boundary edges' ordering and $\mathcal{B}$-lists

called *incoming nets* of $\overline{AB}$. For convenience we let $\beta^-(\overline{BA}_r) \equiv \beta^+(\overline{AE}_l)$, $\beta^-(\overline{BA}_o) \equiv \beta^+(\overline{EF}_o)$, $\beta^-(\overline{BA}_l) \equiv \beta^+(\overline{FB}_r)$. Obviously $\mathcal{B}^+$-list($\overline{AB}$) and $\mathcal{B}^-$-list($\overline{AB}$) contain the same set of nets crossing $\overline{AB}$. These *six* lists are naturally ordered cyclically around the macro node associated with $\overline{AB}$.

3.1 Bundle Graph Construction

We now describe how the bundle graph $\mathcal{BG}$ is constructed. A vertex in $V(\mathcal{BG})$ is a macro node or a module terminal. An edge in $E(\mathcal{BG})$ is a bundle between two vertices in $V(\mathcal{BG})$ if and only if there exist nets between them. Thus the degree of each macro node is at most *six*.

Fig. 7 depicts a subgraph of $\mathcal{BG}$ for all the nets in the $\mathcal{B}$-list of $e = \overline{AB}$. This subgraph, denoted ρ_e, is in fact a *tree* rooted at M_e, the macro node associated with e. This is due to our assumption that each net cannot pass through an elementary region more than once. The subtree ρ_e^+ corresponds to the set of nets in $\mathcal{B}^+$-list($\overline{AB}$) and subtree ρ_e^- corresponds to the set of nets in $\mathcal{B}^-$-list($\overline{AB}$).

$\mathcal{BG}$ can be obtained directly from the global routing specification $\mathcal{W}$. By traversing the netlist $W_i =< R^i_{\alpha 0}, R^i_{\alpha 1}, \ldots, R^i_{\alpha k_i} >$, we can assign net N_i to the β-list associated with $e = \overline{AB}$ as follows. Let us assume the $R^i_{\alpha j}$ and $R^i_{\alpha(j+1)}$ share the edge $e = \overline{AB}$, and N_i crosses $e = \overline{AB}$ from $R^i_{\alpha j}$ to $R^i_{\alpha(j+1)}$. If the next edge $e' = \overline{CD}$ to be crossed is *opposite* to $e = \overline{AB}$, then N_i is assigned to $\beta^+(\overline{AB}_o)$ and hence will be part of the bundle edge between vertices M_e and $M_{e'}$ in $\mathcal{BG}$. The other two cases are treated similarly. Since the net is assigned to a bundle edge without regard to the

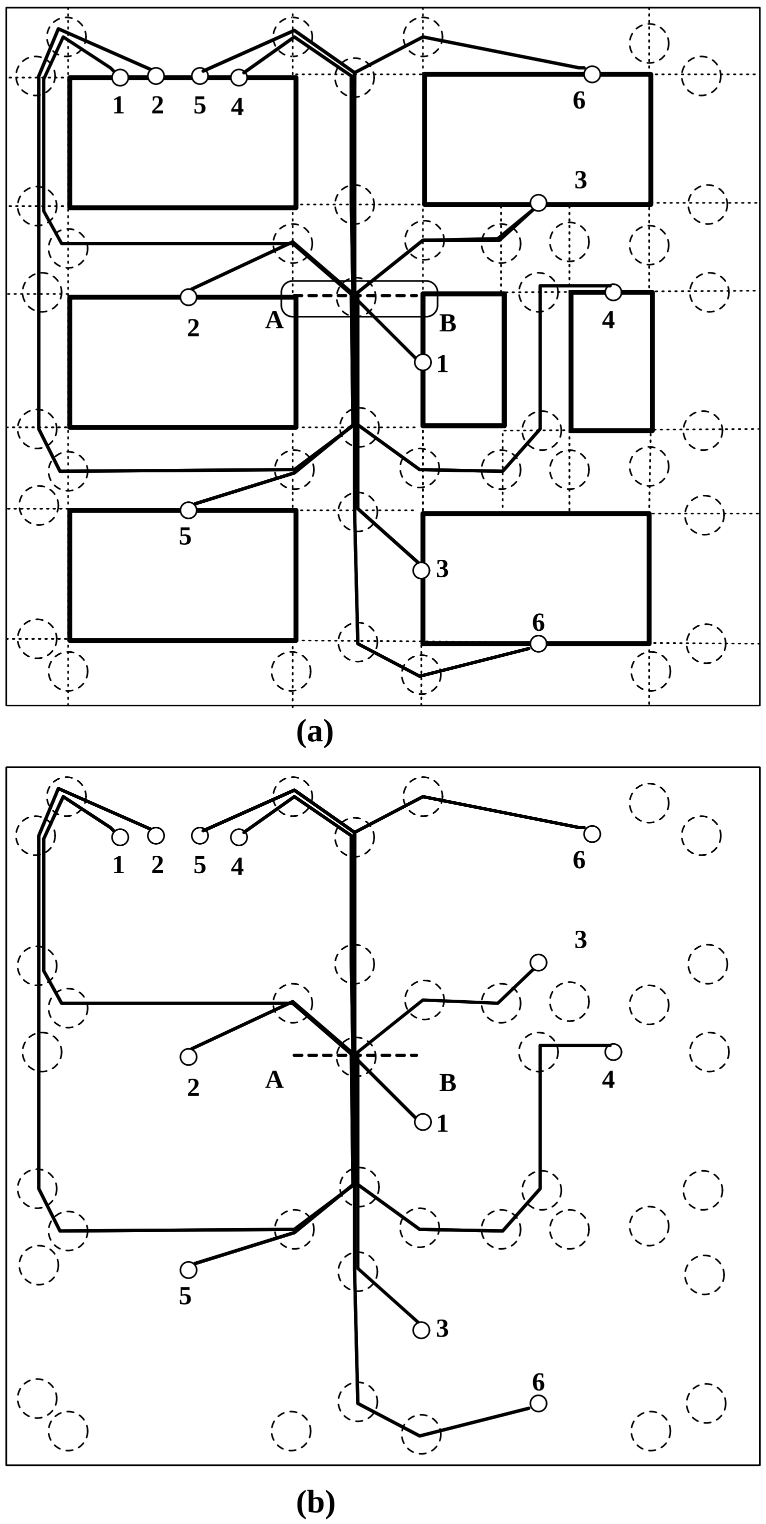


Figure 7: (a)The bundles through the region boundaries. (b)The bundle graph.

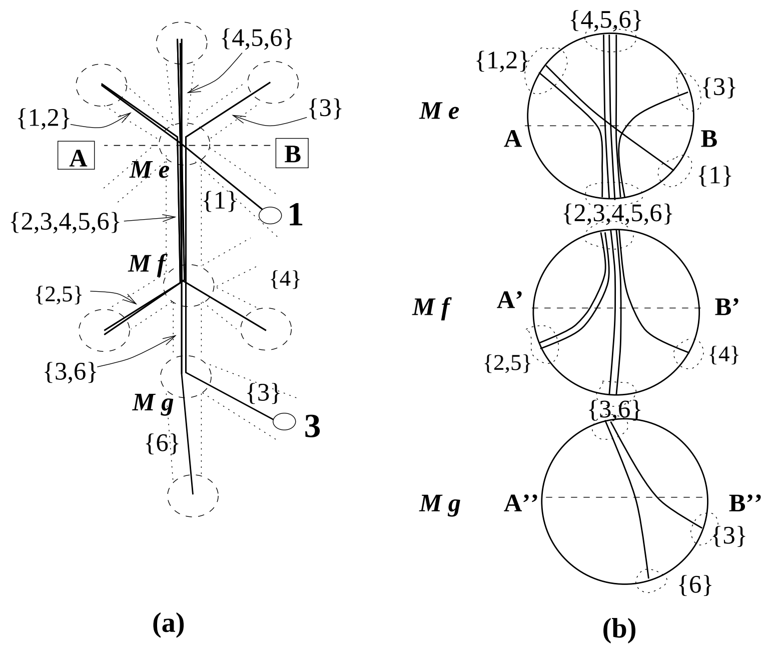


Figure 8: The macro node M_e and tree ρ_e for boundary $e = \overline{AB}$.

ordering, one can use a simple list *insertion* operation to perform the assignment. Thus it is not difficult to see that the bundle graph $\mathcal{BG}$ can be constructed in $O(|\mathcal{W}|)$ time. In Fig. 8(a) nets 4, 5 and 6 are assiged to $\beta^+(\overline{AB}_o)$, nets 1 and 2 are assigned to $\beta^+(\overline{AB}_l)$, and net 3 is assigned to $\beta^+(\overline{AB}_r)$. Nets 2 through 6 are assigned to $\beta^-(\overline{BA}_o)$, and net 1 is assigned to $\beta^-(\overline{BA}_l)$. Fig. 8(b) gives a microscopic view of the *inside* of M_e. Note that though nets 2 through 6 are bundled together in $\beta^-(\overline{BA}_o)$, because of the homotopy of each individual net in the $\beta^+(\overline{AB})$-list, we can conclude that within this bundle net 2 should appear before nets {4,5,6}, followed by net 3 in counterclockwise cyclic order from A to B in order to preserve the same minimal crossings $\mathcal{X}_{min}$ as determined by the RBE representation.

The following rules are used for resolving the order of nets in each of the β-lists.

Proposition 1 *merge/split rule (Fig. 9(a)): Assume bundles $\beta^+(\overline{AB}_x)$, $\beta^+(\overline{AB}_y)$ and $\beta^-(\overline{BA}_z)$ appear in clockwise order from A to B in M_e, where $e = \overline{AB}$. Then within the bundle $\beta^-(\overline{BA}_z)$ the nets in $\beta^+(\overline{AB}_x) \cap \beta^-(\overline{BA}_z)$ should appear before nets in $\beta^+(\overline{AB}_y) \cap \beta^-(\overline{BA}_z)$ in counterclockwise order from A to B.*

Proposition 2 *crossing rule (Fig. 9(b)): Assume bundles $\beta^+(\overline{AB}_x)$, $\beta^+(\overline{AB}_y)$, $\beta^-(\overline{BA}_z)$ and $\beta^-(\overline{BA}_w)$ appear in clockwise order from A to B in M_e, where $e = \overline{AB}$. Then all the nets in $\beta^+(\overline{AB}_x) \cap \beta^-(\overline{BA}_z)$ will cross all the nets in $\beta^+(\overline{AB}_y) \cap \beta^-(\overline{BA}_w)$.*

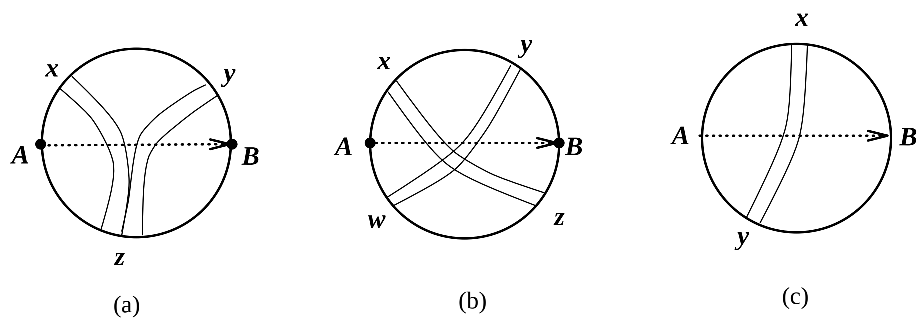


Figure 9: Bundles are merged (a), crossing (b), or ambiguous in a macro node.

Proposition 3 *ambiguity rule (Fig. 9(c)): Assume bundles $\beta^+(\overline{AB}_x)$ and $\beta^-(\overline{BA}_y)$ appear in clockwise order from A to B in M_e, where $e = \overline{AB}$. Then the order of the nets in $\beta^+(\overline{AB}_x) \cap \beta^-(\overline{BA}_y)$ is ambiguous in M_e.*

We first consider the situation where no ambiguity exists among the β-lists in M_e. Since the counterclockwise order among the β-lists is well-defined in M_e, we propose a greedy scan over the β-lists. By *properly* arranging the ordering between the split and merged bundles, we can determine the maximal planar ordering for each unambiguous bundle (or sub-bundle) following Proposition 1. We associate a stack for each β-list, $\beta^+(\overline{AB}_x)$ or $\beta^-(\overline{BA}_y)$, $\pi(\beta^+(\overline{AB}_x))$ or $\pi(\beta^-(\overline{BA}_y))$ respectively to collect the counterclockwise cyclic order of nets from A to B. Initially, all stacks are empty. Assume the scan starts with either end point corresponding to A or B on boundary $e(A, B)$, say B. The scan works in M_e as follows:

Algorithm EXTRACT

Input: A macro node $M_e, e = \overline{A, B}$ and the adjacent bundles

Output: stack contents π's, i.e., $\pi(\beta^+(\overline{AB}_r)), \pi(\beta^+(\overline{AB}_o)), \pi(\beta^+(\overline{AB}_l)), \pi(\beta^-(\overline{BA}_r)), \pi(\beta^-(\overline{BA}_o))$ and $\pi(\beta^-(\overline{BA}_l))$.

Begin

Step 1: (from B counterclockwise to A)

For each bundle $\beta^+(\overline{AB}_x)$ encountered during the scan and for each net $N_i \in \beta^+(\overline{AB}_x)$ such that $N_i \in \beta^-(\overline{BA}_y)$, push N_i onto stack $\pi(\beta^-(\overline{BA}_y))$, denoted *push*$(\pi(\beta^-(\overline{BA}_y)), N_i)$.

Ste 2: (from A counterclockwise to B)

For each bundle $\beta^-(\overline{BA}_y)$ encountered during the scan and for each net $N_i \in \beta^-(\overline{BA}_y)$ such that $N_i \in \beta^+(\overline{AB}_x)$, push N_i onto stack $\pi(\beta^+(\overline{AB}_x))$, denoted *push*$(\pi(\beta^+(\overline{AB}_x)), N_i)$.

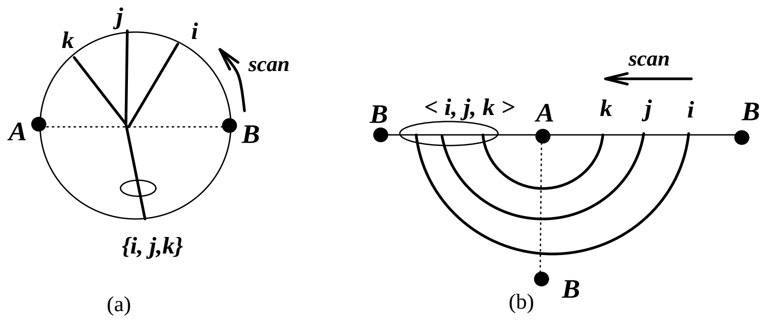


Figure 10: Justification of algorithm EXTRACT

End

To justify that the contents of the stacks keep the maximal planar order following Proposition 1, one can examine how the scan visits the perimeter of M_e. For instance, in Fig. 10(a), $\beta^-(\overline{BA}_o) = \{i, j, k\}$, while $\beta^+(\overline{AB}_l) = \{k\}$, $\beta^+(\overline{AB}_o) = \{j\}$, and $\beta^+(\overline{AB}_r) = \{i\}$. In Fig. 10(b), we flatten the perimeter of M_e to a line. Clearly, when the scan goes from B to A in counterclockwise cyclic order, a *nesting* property appears among the planar nets $\{i, j, k\}$ based on Proposition 1. Therefore, the *last-in-first-out* stack operation can record the exact planar ordering for nets in $\beta^-(\overline{BA}_o)$ with stack $\pi(\beta^-(\overline{BA}_o))$. To answer the question for each net N_i in $\mathcal{B}^+$-list$(\overline{AB})$ ($\mathcal{B}^-$-list$(\overline{AB})$ respectively), *which* bundle in $\mathcal{B}^-$-list$(\overline{AB})$ ($\mathcal{B}^+$-list$(\overline{AB})$ respectively) contains its *counterpart*, we associate with each net member a bundle pointer $\phi(N_i)$. For example, in Fig. 10(a), in $\beta^-(\overline{BA}_o)$, $\phi(N_i) = \beta^+(\overline{AB}_r)$, while in $\beta^+(\overline{AB}_r)$, $\phi(N_i) = \beta^-(\overline{BA}_o)$. $\phi(N_i)$ can be determined during the construction of bundle graph $\mathcal{BG}$ when computing the β-lists as described eariler. Since the above greedy scan procedure puts the connected pair of terminals as *planar* as possible based on the nesting property, all the crossings left are unavoidable (Proposition 2). Note also that even though there are ambiguities (Proposition 3) within some bundles, their partial order with respect to edge $e(AB)$ in M_e is well-defined after the scan. We will show that these ambiguities can be resolved eventually by traversing the whole bundle graph since no net has a self-loop. Before describing the global traversal, we use an example to demonstrate how algorithm **Extract** works. Fig. 11(a) shows a general case for a macro node M_e and its tree ρ_e. In Fig. 11(b) the dashed links are the ϕ pointers following the original global routing topology $\mathcal{W}$. The scan starts from point B to A then back to B again in countercl0ckwise cyclic order. The β-lists around M_e: $\mathcal{B}^+$-list$(\overline{AB}) \equiv \beta^+(\overline{AB}_l) \cup \beta^+(\overline{AB}_o) \cup \beta^+(\overline{AB}_r)$ and $\mathcal{B}^-$-list$(\overline{AB}) \equiv \beta^-(\overline{BA}_l) \cup \beta^-(\overline{BA}_r)$ are defined as follows:

$$\beta^+(\overline{AB}_r) = \{4, 5\},\ \phi(4) = \beta^-(\overline{BA}_l)\ ,\ \phi(5) = \beta^-(\overline{BA}_r)$$

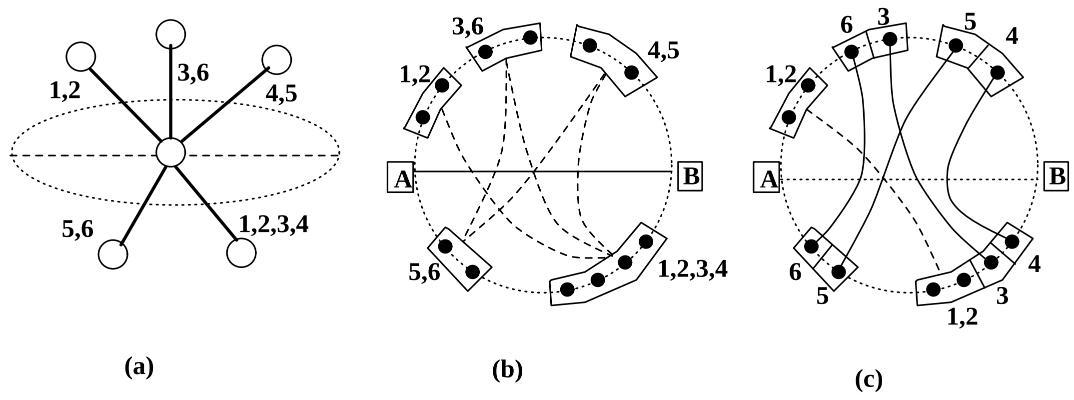


Figure 11: Illustration example of algorithm EXTRACT

$\beta^+(\overline{AB}_o) = \{3, 6\},\ \phi(3) = \beta^-(\overline{BA}_l)\ ,\ \phi(6) = \beta^-(\overline{BA}_r)$

$\beta^+(\overline{AB}_l) = \{1, 2\},\ \phi(\{1, 2\})^1 = \beta^-(\overline{BA}_l)$

$\beta^-(\overline{BA}_r) = \{5, 6\},\ \phi(5) = \beta^+(\overline{AB}_r)\ ,\ \phi(6) = \beta^+(\overline{AB}_o)$

$\beta^-(\overline{BA}_l) = \{1, 2, 3, 4\},\ \phi(\{1, 2\}) = \beta^+(\overline{AB}_l)\ ,\ \phi(3) = \beta^+(\overline{AB}_o)\ , \phi(4) = \beta^+(\overline{AB}_r)$

Initially, there is no order information at all within each bundle. The following is a table listing the order of the bundles visited during the scan, and the associated actions based on the associated ϕ information.

CURRENT BUNDLE	ACTION
$\beta^+(\overline{AB}_r)$	$push(\pi(\phi(4)), 4)$, $push(\pi(\phi(5)), 5)$
$\beta^+(\overline{AB}_o)$	$push(\pi(\phi(3)), 3)$, $push(\pi(\phi(6)), 6)$
$\beta^+(\overline{AB}_l)$	$push(\pi(\phi(\{1, 2\})), \{1, 2\})$
$\beta^-(\overline{BA}_r)$	$push(\pi(\phi(6)), 6)$, $push(\pi(\phi(5)), 5)$,
$\beta^-(\overline{BA}_l)$	$push(\pi(\phi(\{1, 2\})), \{1, 2\})$, $push(\pi(\phi(3)), 3)$, $push(\pi(\phi(4)), 4)$,

In the end, the stack contents from top to bottom (i.e., the counterclockwise cyclic order from B to A for $\beta^+(\overline{AB}_x)$ and the counterclockwise cyclic order from A to B for $\beta^-(\overline{BA}_y)$) are as follows,

$\pi(\beta^+(\overline{AB}_r)) =< 4, 5 >$

$\pi(\beta^+(\overline{AB}_o)) =< 3, 6 >$

$\pi(\beta^+(\overline{AB}_l)) =< \{1, 2\} >$

[1]Since $\phi(1) = \phi(2) = \beta^-(\overline{BA}_l)$, we use $\phi(\{1, 2\}) = \beta^-(\overline{BA}_l)$ and treat $\{1, 2\}$ as *one* ambiguous bundle.

$\pi(\beta^-(\overline{BA_r})) =< 6, 5 >$

$\pi(\beta^-(\overline{BA_l})) =< \{1, 2\}, 3, 4 >$

Based on Proposition 1, we know that the individual stack contains the maximal planar arrangement for the nets in each bundle. Note that the order of nets in the ambiguous sub-bundle $\{1, 2\} \subseteq \beta^+(\overline{AB_l})$ and $\{1, 2\} \subseteq \beta^-(\overline{BA_l})$ has not been determined yet (Proposition 3). In the following section, we explain how the ambiguity is propagated to other macro nodes in $\mathcal{BG}$ until it is resolved.

3.2 Resolving All Ambiguous Bundles

We just described how the ordering of the nets inside a macro node associated with an edge between two regions can be determined. We also noted that some bundles may still be ambiguous after applying **Algorithm** EXTRACT. These ambiguous bundles require a systematic traversal over the macro nodes so that their ambiguity can be resolved efficiently. We claim that a depth-first-search over the bundles and macro nodes would suffice. The traversal recursively goes down from the current macro node M_e to macro node $M_{e'}$ if an ambiguous bundle (or sub-bundle) b_i exists between them, and returns the resolved ordering for b_i from $M_{e'}$. In the example Fig. 11(c), we do not know the exact ordering of nets within the sub-bundle $\{1, 2\}$ but we do know it is a sub-list in $\beta^-(\overline{BA_l})$ where $\{1, 2\}$ should appear before nets 3 and 4 in counterclockwise cyclic order from A to B. Therefore, the traversal continues following either bundle $\beta^+(\overline{AB_l})$ or $\beta^-(\overline{BA_l})$ to the neighboring macro node. The order of $\{1, 2\}$ should be determined eventually in some macro node as long as there is no self-loop in either net N_1 or N_2 (so that the traversal will not go back to bundle $\beta^-(\overline{BA_l})$ or $\beta^+(\overline{AB_l})$ in M_e again, resulting in an infinite loop). For the set of nets without merge/split at all in any macro node, the relative positions of module terminals will eventually determine the maximal planar order of these nets. An extreme case is bus routing: all nets associated with the same bus have the same global routing homotopy. But their relative positions will be determined by the module terminals in the leaf nodes (i.e., nodes of degree one) of $\mathcal{BG}$ at which the depth-first-search terminates. Otherwise, based on Proposition 1, as long as there are merge/split configurations, algorithm EXTRACT can always determine a partial order. For example, in Fig. 12(a) macro node M_f connected to M_e by bundle $\beta^-(\overline{BA_o}) = \{2, 3, 4, 5, 6\}$ may be the next macro node to visit if there are some ambiguous sub-bundles inside $\beta^-(\overline{BA_o})$ after **Algorithm** EXTRACT is applied to M_e. Assume $f = \overline{A'B'}$ and $g = \overline{A''B''}$ are the other two edges in tree ρ_e^- as shown in Fig. 12(a). Fig. 12(b) shows crossing configurations inside the marco nodes M_e, M_f and M_g. In $\mathcal{B}^-$-list within

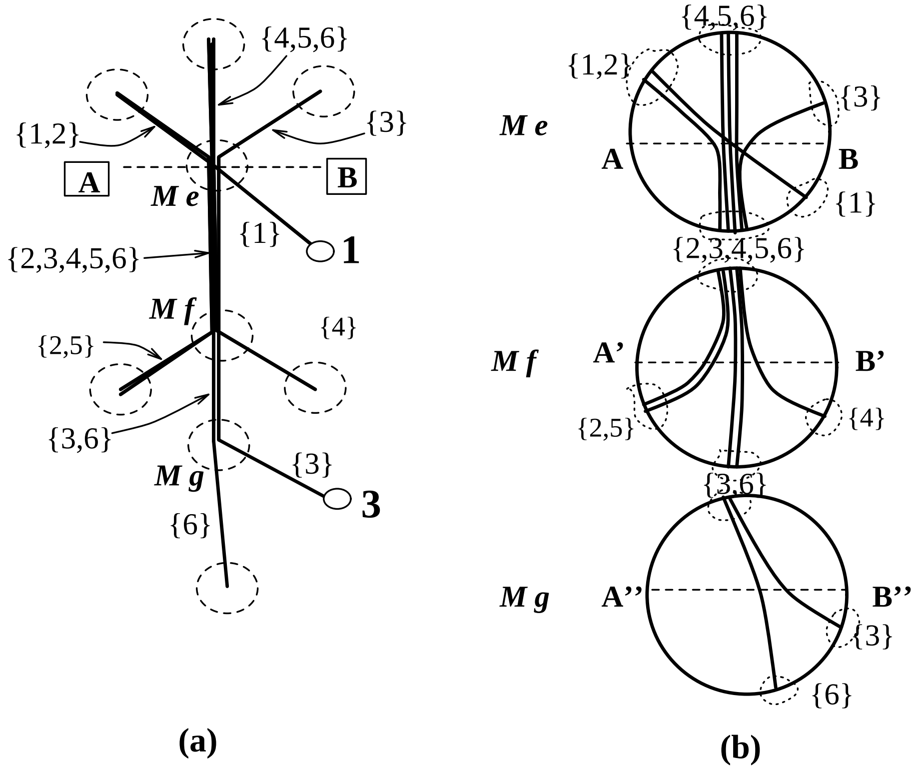


Figure 12: Resolving ambiguous bundles.

M_f, there are three sub-bundles split out of $\mathcal{B}^+$-list. The order of $\beta^+(\overline{A'B'}_o)$ ($\equiv \beta^-(\overline{BA}_o)$) can be determined by the following in counterclockwise cyclic order based on Proposition 1:

$$\pi(\beta^+(\overline{A'B'}_o)) = \pi(\{2,5\}) \parallel \pi(\{3,6\}) \parallel \pi(\{4\})$$

or

$$\pi(\beta^+(\overline{A'B'}_o)) = \pi(\beta^-(\overline{B'A'}_r)) \parallel \pi(\beta^-(\overline{B'A'}_o)) \parallel \pi(\beta^-(\overline{B'A'}_l)) \tag{1}$$

We use symbol $\parallel$ to denote the "concatenation" operation among the ambiguous or unambiguous sub-bundles. At this point we have two ambiguous bundles, $\pi(\beta^-(\overline{B'A'}_r))$ and $\pi(\beta^-(\overline{B'A'}_o))$. Assume macro node M_g is the next node to visit from M_f for bundle $\beta^-(\overline{B'A'}_o)$. Then, by Extract scan, we obtain the counterclockwise order

$$\pi(\beta^-(\overline{B'A'}_o)) = \pi(\beta^+(\overline{A''B''}_o)) = \pi(\{6\}) \parallel \pi(\{3\})$$

or

$$\pi(\beta^-(\overline{B'A'}_o)) = \pi(\beta^+(\overline{A''B''}_o)) = \pi(\beta^-(\overline{B''A''}_o))\|\pi(\beta^-(\overline{B''A''}_l)) =< 6, 3 >$$

($< 6, 3 >$ means the order is $6 \rightarrow 3$ from A' to B' in counterclockwise cyclic order.) Since there is no more ambiguity inside $\pi(\beta^-(\overline{B'A'}_o))$, we can go back to M_f and restore the partially determined order in (1) and get

$$\pi(\beta^+(\overline{A'B'})_o) = \pi(\beta^-(\overline{B'A'}_r))\| < 6, 3, 4 >$$

In the following, we call this merging *backpatch*. We do the same thing for bundle-list $\beta^-(\overline{B'A'}_r)$ and finally $\pi(\beta^+(\overline{A'B'})_o)$ can be determined and returned to M_e from M_f. During the depth-first-search, one can see that the "self-loop-free" assumption is very important. Otherwise, the traversal might go through a set of macro nodes and hit one of the visited nodes again, resulting in an infinite loop. The following describes the details of such a recursive algorithm and then justifies its correctness based on the above self-loop-free assumption.

Algorithm MACRO_TRAVERSAL

Input: The bundle graph $\mathcal{BG}$ and initially, all bundles in $E(\mathcal{BG})$ are not "ordered".

Output: The exact ordering of all β-lists from A to B for all edges $\overline{AB}$.

Method:

Begin

For each un-ordered bundle $\beta^{\pm}(\overline{AB}_x) \in E(\mathcal{BG})$ in macro node M_e.

/* x is l, o or r */

Call Algorithm DFS (M_e, $\beta^{\pm}(\overline{AB}_x)$, *nil*).

End

Algorithm DFS

Input: A macro node $M_e \in V(\mathcal{BG})$, a bundle $\beta^{\pm}(\overline{AB}_x)$ and an ambiguous sub-bundle b contained in $\beta^{\pm}(\overline{AB}_x)$.

Output: $\pi(\beta^{\pm}(\overline{AB}_x))$

Method:

Begin

1. **if** M_e is visited the first time **then**
   Call Algorithm EXTRACT(M_e) to determine the maximal planar order over the bundles around M_e.

2. Assume $\pi(\beta^{\pm}(\overline{AB}_x)) = \pi(b_0)\|\pi(b_1)\|...\|\pi(b_i)\|...\|\pi(b_p)$, for some p, $1 \leq p \leq n-1$, determined by EXTRACT scan, where b_i's are sub-bundles split in M_e.

3. **if** $\forall i, \pi(b_i)$ is unambiguous

   /* no ambiguity, then stop */

   goto step 5.

4. **For each** macro node M_f connected by an ambiguous bundle b_i ($b_i \equiv \beta^{\pm}(\overline{AB}_x) \cap (\beta^{\mp}(\overline{BA}_y) \equiv \beta^{\pm}(\overline{A'B'}_z))$ $f = \overline{A'B'}$.)

   Call Algorithm DFS (M_f, $\beta^{\pm}(\overline{A'B'}_z)$, b_i).

   /* upon return, $\pi(b_i) \subseteq \pi(\beta^{\pm}(\overline{A'B'}_z))$ is unambiguous. */

   Backpatch $\pi(b_i)$ to $\pi(\beta^{\pm}(\overline{AB}_x))$ in step 2.

5. Mark $\beta^{\pm}(\overline{AB}_x)$ as "ordered".

6. **return** $\pi(\beta^{\pm}(\overline{AB}_x))$.

End

Since maximal planar ordering of a bundle is determined by Proposition 1 and once it is determined, it is fixed and propagated back, clearly there is no crossings within a bundle between two macro nodes. Moreover, from Proposition 2, we know that at the end of the traversal, all the crossings left inside the macro nodes are unavoidable and the number equals $\mathcal{X}_{min}$. We remark that the first $\beta^{\pm}(\overline{AB}_x)$ selected in the DFS driving routine MACRO_TRAVERSAL may be chosen to be one containing only module terminals so that all the sub π lists are of cardinality one. Consequently we will invoke fewer recursive calls because the relative ordering of the module terminals is already given as part of the placement $\mathcal{P}$. For example, in Fig. 7, starting from the bundles $\{1,2\}$ or $\{5,4\}$ at the upper left corner, we can immediately determine the order of the nets in these bundles.

Lemma 1 **Algorithm** MACRO_TRAVERSAL *visits each bundle $\beta^{\pm}(\overline{AB}_x)$ in macro node M_e at most n times.*

Proof We prove this by considering the following two cases.

Case 1: When the sub-bundles in $\beta^{\pm}(\overline{AB}_x)$ are all unambiguous either because (i) the nets are connected to module terminals or (ii) their ambiguity gets resolved by recursive calls to DFS, we terminate DFS.

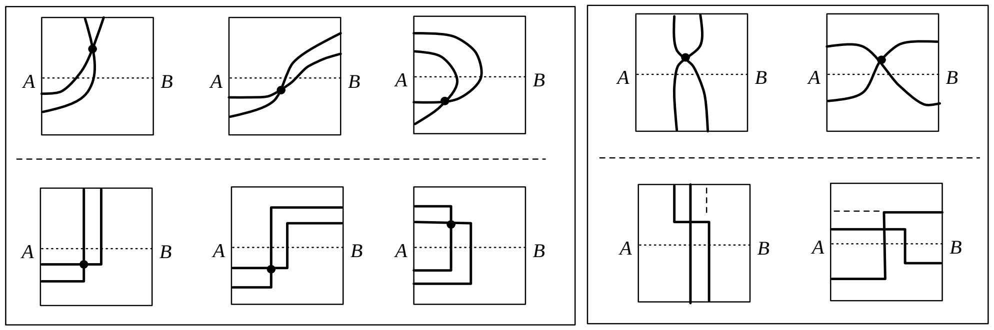
Case 2: Suppose some sub-bundles in $\beta^{\pm}(\overline{AB}_x)$ are ambiguous. Suppose sub-bundle $b_i \subseteq \beta^{\pm}(\overline{AB}_x)$ of M_e connects M_f and $b_i \subseteq \beta^{\pm}(\overline{A'B'}_z)$, where $f = \overline{A'B'}$. A recursive call DFS (M_f, $\beta^{\pm}(\overline{A'B'}_z)$, b_i) will be invoked. The algorithm would fail, if there is an infinite loop. However, this is impossible. If M_e is visited subsequently via a recursive call DFS (M_e, $\beta^{\pm}(\overline{A''B''}_w)$, b), the ambiguous sub-bundle b that lead to the recursion must be distinct from that in the previous recursive call. This is clear, as it was assumed that no net enters an elementary region more than once.

Therefore, an ambiguous sub-bundle b_i can be resolved with one visit (case 1) or two visits (case 2) due to backpatch operation. Thus each bundle can be visited by DFS algorithm at most two times the total number of ambiguous sub-bundles, which is less than n. The lemma follows. $\square$.

Note that since each macro node M_e has at most 6 bundles, the total number of bundles in $\mathcal{BG}$, is at most $3m$, i.e., $|E(\mathcal{BG})| = 3m$, where m denotes the number of macro nodes. So the running time of the whole traversal is $O(mn)$.

Until this point, we have only shown that an ordering of the junction terminals can be made so that the total number of crossings $\mathcal{X}$ is minimized. However, to finalize the topology of the global routing, we still need to determine a junction terminal ordering σ on each $e(AB)$. Note that all unavoidable crossings are *inside* the macro nodes, since each bundle is assumed to have no twistings. Moreover, the unavoidable crossings inside a macro node may be realized by many possible junction terminal orderings σ. In the following, we consider two subproblems associated with CCMP: (1) crossing-to-region assignment and (2) junction terminal ordering, given a crossing-to-region assignment.

We consider the crossing-to-region assignment subproblem first. Basically, the problem of determining the junction terminal ordering σ is equivalent to determining for each macro node M_e which one of the two regions adjacent to $e(AB)$, above or below $e(AB)$ (respectively right or left of $e(AB)$) to place the unavoidable crossings (i.e., region $\overline{ABCD}$ or region $\overline{BAEF}$ in Fig. 6). Note that once it is decided to which region each crossing is to be assigned, the exact junction terminal ordering σ can then be obtained accordingly. The problem of assigning the crossings to the regions is called the *crossing placement problem*[Groe89]. Depending on the crossing placement criterion, the problem can be formulated in different ways. For example, to avoid occupying extra tracks by nets, Groenveld[Groe89] employed a heuristic by checking a spanning tree ρ_e corresponding to each boundary edge e to pick a preferred crossing placement. Marek-Sadowska and Sarrafzadeh[MS95] classified the crossings into two types, *real* and *free* crossings. Real crossings



(a) (b)

Figure 13: (a) some free crossing types and (b) real crossings.

represent crossings that require more tracks in a region while free crossings represent crossings that will not incur track penalty. Free crossings can be placed in either top (right) region (e.g., $\overline{ABCD}$) or bottom (left) region (e.g., $\overline{BAEF}$) so that they don't incur any track penalty. On the other hand, real crossings do incur extra track penalty no matter in which region they are placed. Fig. 13(a) shows some free crossings and their *preferred* placement for which they will not incur track penalty, while (b) shows the two types of real crossings that require one extra track. In terms of bundles, the real crossings are formed among the nets in either (1) $\beta^+(\overline{AB}_o) \cap \beta^-(\overline{BA}_o)$, or (2) $(\beta^+(\overline{AB}_r) \cap \beta^-(\overline{BA}_r)) \cup (\beta^+(\overline{AB}_l) \cap \beta^-(\overline{BA}_l))$. The authors in [MS95] formulate the problem as a matching problem in CDP with a quota for the number of crossings imposed on each region. By finding a maximal matching between the regions and the (unavoidable) real crossings, every crossing is assigned to a region. Once the crossing-to-region assignment is made, σ can be computed in a straightforward manner. Since CDP is an independent problem, we don't go into it further. However, regarding the real and free crossings that occur in the bundles inside M_e, we propose the following simple procedure to do crossing-to-region assignment.

1. Locate free crossings (i.e., crossings other than the two types of real crossings in Fig. 13(b)) between the nets and determine their regions with the preferred configurations.

2. Evenly distribute the real crossings into the two regions (or apply a CDP algorithm to determine the overall real crossings distribution).

We omit the details here. The interested reader is referred to [C97].

Thus, we conclude with the following theorem.

Theorem 1 *Given a global routing specification of a circuit placement $\mathcal{P}$ with m modules and n nets, the CCMP problem can be solved in $O(mn)$ which is asymptotically optimal in the worst case.*

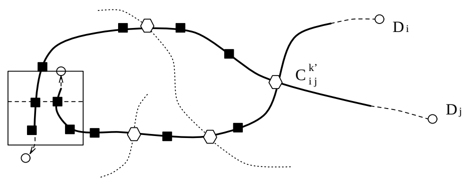


Figure 14: Feature 2 violation.

4 Solving UCMP

A UCMP problem instance is also described by the pair $(\mathcal{W}, \mathcal{G})$. The goal is to determine a best ordering of the junction terminals of the nets on each edge $e \in \mathcal{G}$ so that each pair of nets has at most *one* crossing. However, now $\mathcal{W}$ is allowed to change when necessary[2]. The following two features are identified in [MS95]:

- **Feature 1:** Every two nets cross each other at most once.
- **Feature 2:** Every two nets that cross at one region do not cross a boundary of another region in opposite directions as shown in Fig. 14.

The redundant crossing elimination algorithm given in [MS95] ensures that **Feature 1** be maintained first and then modify the result into one that satisfies **Feature 2**. It is easy to establish that maintaining **Feature 1** minimizes the number of crossings. However, maintaining **Feature 2** does *not necessarily* reduce the number of crossings, as was suggested in [MS95]. We will demonstrate that the number of crossings may increase if **Feature 2** is to be maintained. In this paper we only consider UCMP with **Feature 1**. The problem of maintaining **Feature 2** and minimizing the number of crossings simultaneously remains *open*.

We assume as before that no net can enter and exit a region more than once. From this point forward, we assume that junction and module terminals along the boundary of each region are stored in a doubly-linked circular list, starting with the top right corner of each rectangular region in clockwise direction. A junction terminal on a boundary edge will be stored twice, one in each of the two regions sharing the common edge. These terminals can be stored in an ordered list for each boundary edge individually as each netlist is traversed and then combined into a doubly-linked circular list for each region. Consider region R_i in Fig. 3(b) and its left boundary edge as an example. The left boundary edge initially contains two module terminals 1 and 5, and has three parts, top, middle and bottom, that may contain junction terminals. When we traverse N_1, we

[2]The change to $\mathcal{W}$ is confined locally and will not introduce *new* crossings, as described in more detail later.

cross the top part and assign the junction terminal 1 to the list as the *first* element from top down. Later, when we traverse N_4, the same top part is crossed a second time, so junction terminal 4 is assigned *after* junction terminal 1. In this manner, we obtain an initial topological representation for all netlists in the global routing. Other topological representations are possible. The assignment of the junction terminals shown in Fig. 3(b) is done in this manner: That is, for vertical edges, the junction terminals are ordered from top down and for horizontal edges, they are ordered from right to left. The specification of the relative (topological) positions of the junction terminals and module terminals on the boundaries of the regions is denoted Π, and the circular list of terminals associated with region R_i is denoted as Π_i.

4.1 Net Crossing Detection

Assume that a topological representation like the one shown in Fig. 3(a) is given initially. Let C_{ij}^k denote the crossing between net N_i and net N_j in region R_k. A crossing C_{ij}^k between two nets is considered *redundant* if there exists another crossing C_{ij}^l, $k \neq l$. For example, in Fig. 2(b), the two C_{23}'s are redundant crossings. The goal of UCMP is to remove such crossings completely in the given topological representation.

In each region R_k suppose that the terminals in the circular list Π_k are each identified with the index of the net passing through the terminal. If the pattern (i, j, i, j) appears in Π_k, then it *imposes* a crossing C_{ij}^k between N_i and N_j in R_k. This is referred to as the *crossing condition.*

To facilitate detection of redundant crossings we construct explicitly a *crossing graph* from the topological representation of the global routing.

Definition 5 *A crossing graph $\mathcal{D}$ of the global routing $\mathcal{W}$ with respect to topology Π is a planar graph in which a vertex $u \in V(\mathcal{D})$ represents a (module or junction) terminal or a crossing imposed by Π, and an edge $(u, v) \in E(\mathcal{D})$ either connects two consecutive junction terminals, u and v, on a boundary edge, or is part of the path associated with net N_i, $D_i =< a_i, J_{\alpha 1}^i, J_{\alpha 2}^i, \ldots, J_{\alpha k}^i, a_i' >$, where a_i, a_i' are the vertices corresponding to the two module terminals of N_i, and $J_{\alpha d}^i$, $d = 1, 2, \ldots k$, is a vertex corresponding to a junction terminal or a crossing $C_{ij}^l, i \neq j$ in some region R_l.*

Let τ and ξ denote the numbers of terminals and crossings respectively. It is obvious that $\tau = O(mn)$ in $\mathcal{P}$. Since $\mathcal{D}$ is planar, $|E(\mathcal{D})| = O(|V(\mathcal{D})|) = O(\tau + \xi)$. In Fig 15, D_i and D_j are two paths associated with N_i and N_j in $\mathcal{D}$, where white circles, dark squares and white hexagons represent respectively the module terminals, junction terminals and crossings. The portion of D_i between two dark squares corresponds to a routing path that lies in some region R_k. Due to the assumption previously stated (no net enters and exits a region more than once) there is at most one crossing C_{ij}^k for each R_k, and $j \neq i$. Based on the *crossing condition* in each region, we first present an algorithm that detects all the crossings and then an algorithm that computes the crossing graph explicitly.

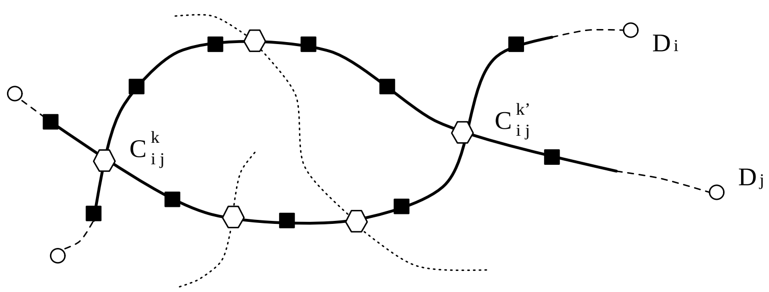


Figure 15: A pocket

Algorithm CROSSING_DETECTION

Input: A region R_k and the circular list Π_k of the terminals on its boundary.

Output: A collection of crossings C_{ij}^k for $i \neq j$.

Method:

begin

1. Initialize an empty stack S implemented as a linked list [HS76].
2. **For each** terminal t_i of net N_i in Π_k,

Case 1 If t_i is visited the first time, push t_i onto S.

Case 2 If t_i is visisted the second time:

Repeat

Scanning the terminals t_j on the stack starting from the topmost terminal.

For each terminal $t_j \neq t_i$ report crossing C_{ij}^k.

Until terminal t_i is found.

Delete t_i from list S.

end

We conclude with the following lemma.

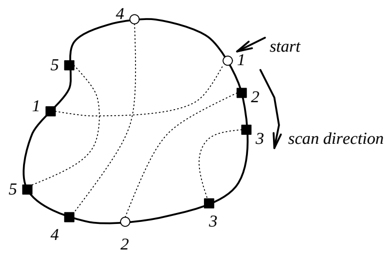


Figure 16: Crossing detection

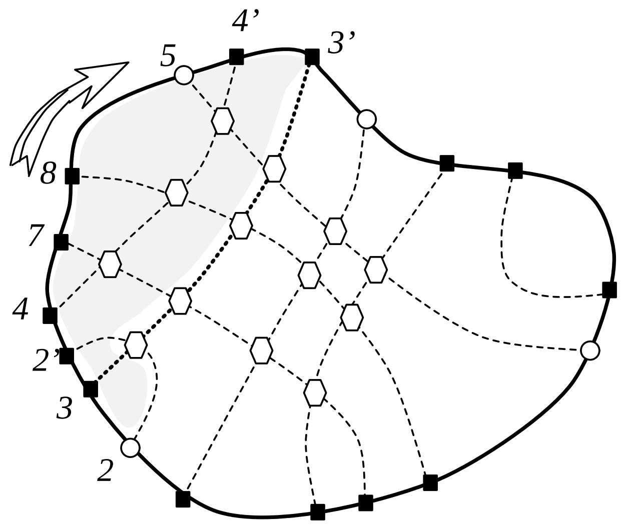


Figure 17: The crossing connection in a region

Lemma 2 *All the crossings* C_{ij}^k, $1 \le i, j \le n$ *for all regions* R_k, *can be detected in* $O(\tau + \xi)$ *time.*

For example, in Fig 16, as we traverse the boundary clockwise starting from say terminal 1 (for net N_1), we operate the stack as follows,

current terminal	stack content	action
1	$\{\}$	push(1st terminal of N_1)
2	$\{1\}$	push (1st terminal of N_2)
3	$\{1, 2\}$	push(1st terminal of N_3)
3	$\{1, 2, 3\}$	pop(2nd terminal of N_3)
2	$\{1, 2\}$	pop(2nd terminal of N_2)
4	$\{1\}$	push(1st terminal of N_4)
5	$\{1, 4\}$	push(1st terminal of N_5)
1	$\{1, 4, 5\}$	2nd terminal of N_1; report C_{15} and C_{14} and delete 1
5	$\{4, 5\}$	pop (2nd terminal of N_5)
4	$\{4\}$	pop (2nd terminal of N_4)

We now describe how to explicitly compute $\mathcal{D}$ within the same time bound using a similar method.[3] Consider region R_k in $\mathcal{P}$ like the one shown in Fig 17. Define the interval $I_i(a, a')$ along the boundary of R_k from terminal a to its counterpart a' associated with net N_i in clockwise direction. Note that if N_i passes through R_k, there must exist *two* terminals on the boundary of R_k. The path $\Phi_i(a, a')$ of N_i from terminal a to terminal a' divides R_k topologically into two parts. The key observation on which the algorithm for computing $\mathcal{D}$ is based, is that the crossings C_{ij}^k that appear on $\Phi_i(a, a')$ must occur in the same order as the terminals of N_j's that cross N_i and belong to $I_i(a, a')$. These crossings on $\Phi_i(a, a')$ are said to be *induced* by these terminals in $I_i(a, a')$. For example, in Fig 17, $I(3, 3')$ contains terminals $2', 4, 7, 8, 5, 4'$ and $\Phi(3, 3')$ contains crossings $C_{32}, C_{37}, C_{38}, C_{35}$. Note that in **Algorithm** CROSSING_DETECTION those terminals scanned in **Case 2** to report crossings produce the actual crossings in order along the path from the second terminal to the first terminal. An implementation of this algorithm can be found in [C97]. We omit the details but report the following result:

Lemma 3 *The crossing graph $\mathcal{D}$ can be computed in $O(\tau + \xi)$ time.*

4.2 Pocket and Untangling

We first briefly explain how to modify a global routing without introducing new crossings such that it possesses **Feature 1** in the following two subsections.

Definition 6 *Suppose N_i and N_j have more than one crossing in the crossing graph $\mathcal{D}$. A pocket between nets N_i and N_j is a loop that is formed by subpaths of D_i and D_j, and two crossings C_{ij}^k and $C_{ij}^{k'}, k \neq k'$, such that no other crossing $C_{ij}^{k''}$, $k'' \neq k$, $k'' \neq k'$, exists between them on either D_i or D_j.*

An example of a pocket between net N_i and net N_j is shown in Fig 15.

Assume that all the edges in graph $\mathcal{D}$ are each labeled by the index of the net associated with it when it is constructed, so that the path D_i for each net N_i can be easily traced. The two crossings forming a pocket are redundant crossings and should be eliminated. To remove a pocket, we can exchange the labels of all the edges belonging to the pocket as shown in Fig. 18. One way to implement this idea is to trace every pair of nets. Whenever a pocket is found, exchange the labels of the edges belonging to the pocket. One can do it by constructing a *crossing table* as described in [MS95] and then update the table each time a pocket is removed. The update is needed because the labels of the edges have changed and any crossing involving either one of these two nets needs

[3] The crossing detection algorithm can in fact be incorporated with the crossing graph computation.

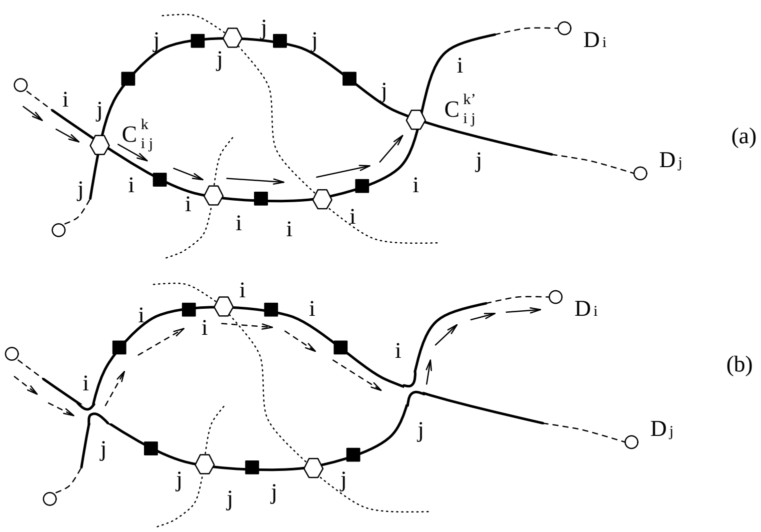


Figure 18: Untangling

to be changed as well. Since each net could go through $O(m)$ regions, the crossing table can be constructed in $O(mn^2)$ time. Each time a pocket is removed, two crossings are eliminated. Since each update removes 2 crossings, and takes time proportional to the number of remaining crossings, the total update time is $\sum_{i=2}^{\xi}(i-2) = O(\xi^2)$, resulting in an overall time $O(mn^2 + \xi^2)$ [MS95].

In the following sections, we propose an algorithm that can remove all the redundant crossings in time $O(n\xi)$ with $O(mn+\xi)$ preprocessing time. The key observation is that $\mathcal{D}$ is a planar graph, a traversal of which takes time proportional to the size of the graph. Before we explain our idea, we need to define a basic operation used in the algorithm. Consider the pocket shown in Fig 18(a). To detect the pocket, we trace either N_i or N_j and maintain a counter for each crossing encountered.

Assume we start the trace from D_i. When $C_{ij}^{k'}$ is encountered after the first crossing C_{ij}^{k}, a pocket is detected. We *untangle* this pocket by switching the labels of edges incident upon the two crossings so that both crossings C_{ij}^{k} and $C_{ij}^{k'}$ are removed. Consider the second crossing $C_{ij}^{k'}$ as in Fig 19(a), and let the vertex representing $C_{ij}^{k'}$ be v and the four edges adjacent to v be (b', v), (b, v) (labeled as i) and (v, a'), (v, a) (labeled as j), where the shaded portion represents the pocket.

Removing C_{ij}'s involves the following steps:

Procedure UNTANGLE

1. Delete vertex v ($C_{ij}^{k'}$).

2. Delete edges (a, v), (b, v), (v, a'), and (v, b').

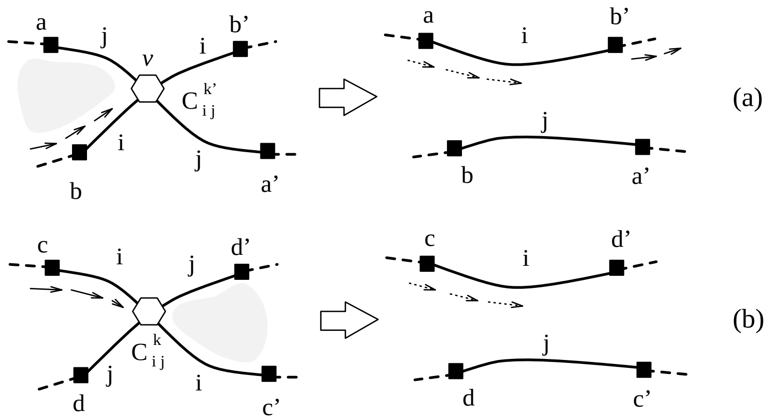


Figure 19: Changing the connectivity

3. Add edge (a, b') with label i.

4. Add edge (b, a') with label j.

C_{ij}^{k} can be handled similarly as shown in Fig 19(b). Note that after this untangling, the next vertex to be visited in tracing D_i is set to be the first vertex after C_{ij}^{k} before untangling (vertex d' in Fig 19(b)) following the same direction as shown in Fig. 18(b). That is, we trace the *new* portion of D_i obtained after untangling, and continue the tracing operation. Some implementation details need to be addressed, however. For example, there are two edges at the second crossing $C_{ij}^{k'}$ whose labels need to be switched but the labels of the other two edges should stay the same. We need a mechanism to identify the two edges that are part of the pocket. Similar arguments hold for the edges incident upon the first crossing C_{ij}^{k}. Secondly the *directions* of the edges that govern the tracing directions must be consistent after the relabeling operation. For instance consider Fig. 18. After the untangling, the *new* portion of D_i which is part of the detected pocket is to be traced in the direction as indicated by the dotted arrows. If D_j goes in different direction, then the pointers of these edges, which were part of D_j should all be *reversed*. Similarly, the pointers of those edges that were part of D_i must be reversed as well.

4.3 The Algorithm

We give a high-level description of the algorithm. We assume that in crossing graph $\mathcal{D}$ we impose a fixed but arbitrary direction on the edges that belong to the same net. This direction will be used to guide our tracing operation. We denote this graph with added orientations as $\mathcal{D}_o$.

We first compute a *net intersection* graph defined as follows.

Definition 7 *The net intersection graph $\mathcal{H}$ is an undirected multiple edge graph such that vertex $v_i \in V(\mathcal{H})$ corresponds to path D_i in $\mathcal{D}$, and edge $(v_i, v_j) \in E(\mathcal{H})$ corresponds to a crossing C_{ij} between paths D_i and D_j in $\mathcal{D}$.*

This graph can be constructed in time proportional to the size of $\mathcal{D}$, which is $O(mn + \xi)$, and requires $O(n + \xi)$ space.

The traversal is performed for each net. Whenever a second crossing is encountered, UNTANGLE operation is performed and edges in the pocket between the two crossings are relabeled. The first crossing can be found from graph $\mathcal{H}$ through an auxiliary link. Meanwhile, all the crossings associated with the pocket are correctly maintained by changing the adjacency structure (the edges between two crossings that form a pocket will be switched between their adjacent vertices (nets) in $\mathcal{H}$.) For example, Fig. 20(a) and (b) show a UCMP instance and its net intersection graph respectively. If we begin traversal from terminal 1 of net 1, we will encounter crossing C_{12} twice. Therefore, a pocket is detected and crossings C_{23} and C_{24} are switched from net 2 to net 1. Net 2 is shortened since there were no crossings between the two crossings C_{12}'s as shown in Fig. 21. Note that in Fig. 21(b), the edges in $\mathcal{H}$ are updated accordingly. We continue to follow the *new* portion of net 1 that contains C_{14} and C_{13} (previously, C_{23} and C_{24}) and detect crossings C_{13} twice, which gives another pocket. This is shown in Fig. 22.

The following summerizes the above steps.

Algorithm CROSSING_MINIMIZATION

Input: Crossing graph $\mathcal{D}_o$

Output: Relabeled $\mathcal{D}_o$ in which each pair of net has at most one crossing.

Method:

begin

for each net N_i trace the path $D_i = < a_i, J^i_{\alpha 1}, J^i_{\alpha 2}, \ldots, J^i_{\alpha k}, a'_i >$.

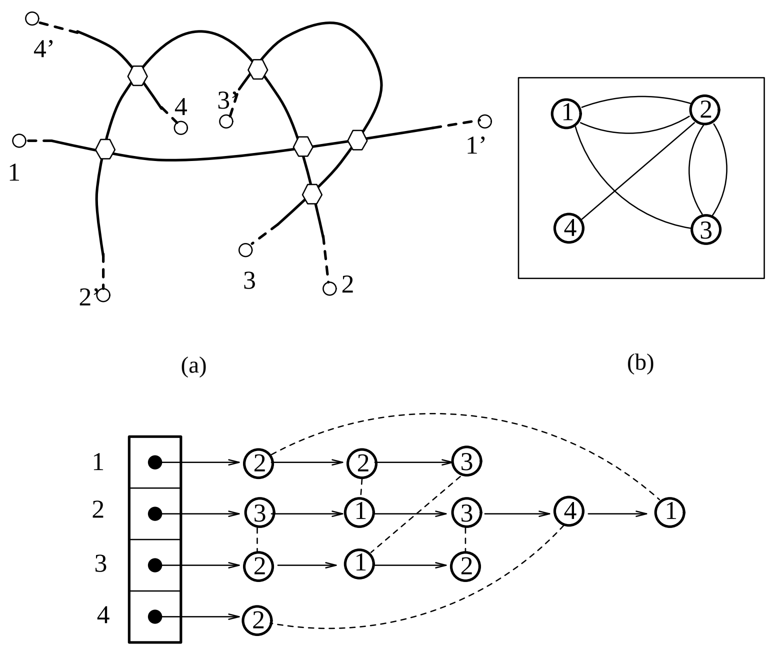
do

for each edge $e = (u, v)$ in $\mathcal{H}$ corresponding to a crossing in $\mathcal{D}$

do

if v is the 2nd crossing of a pocket, **then**

- Find the 1st vertex w of the pocket in $\mathcal{H}$.
- Relabel all the edges of the pocket between v and w



(a) (b) (c)

Figure 20: (a)A crossing graph $\mathcal{D}$ and (b)its intersection graph $\mathcal{H}$.

in $\mathcal{D}$ accordingly.

- Remove the edges corresponding to w and v from $\mathcal{H}$.
- Switch the edges (crossings) to their new adjacent vertices (nets) in $\mathcal{H}$.

Set v to be the next node after w.

end

Basically, the above algorithm traces $\mathcal{H}$ over each adjacency list for each vertex (net) and whenever a *second* crossing is encountered, the two sublists in $\mathcal{H}$ corresponding to the crossings in the portions in the pocket between the two crossings are switched to reflect the crossing change. In Figs. 21 and 22, we show how the crossings and pockets in Fig. 20 are removed by the above algorithm.

4.4 Time Complexity

Recall that for each crossing there are four incident edges along the two crossing nets, and thus there are a total of $O(\xi)$ subpaths between two crossings in $\mathcal{D}$. We call this subpath between two consecutive crossings *elementary path* in $\mathcal{D}$.

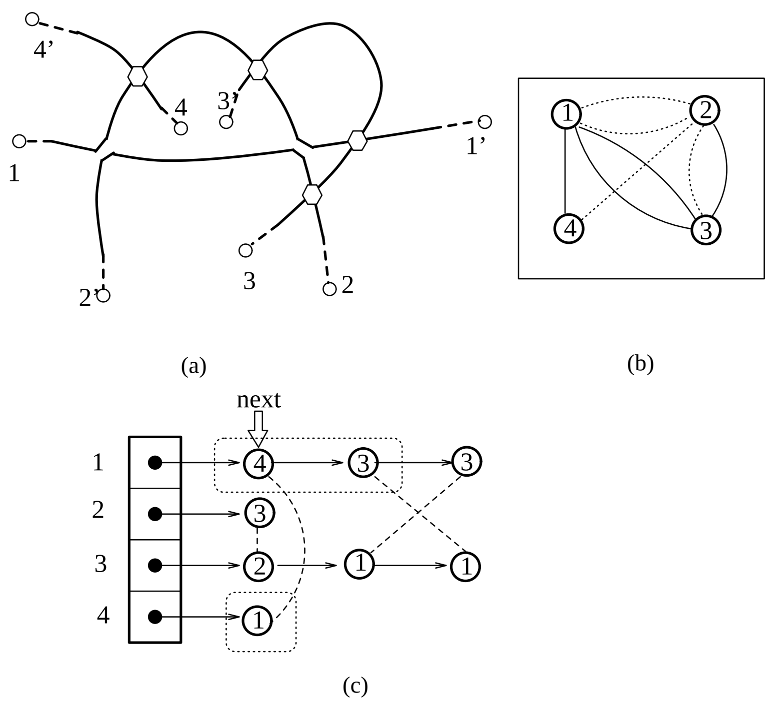


Figure 21: After removing C_{12}

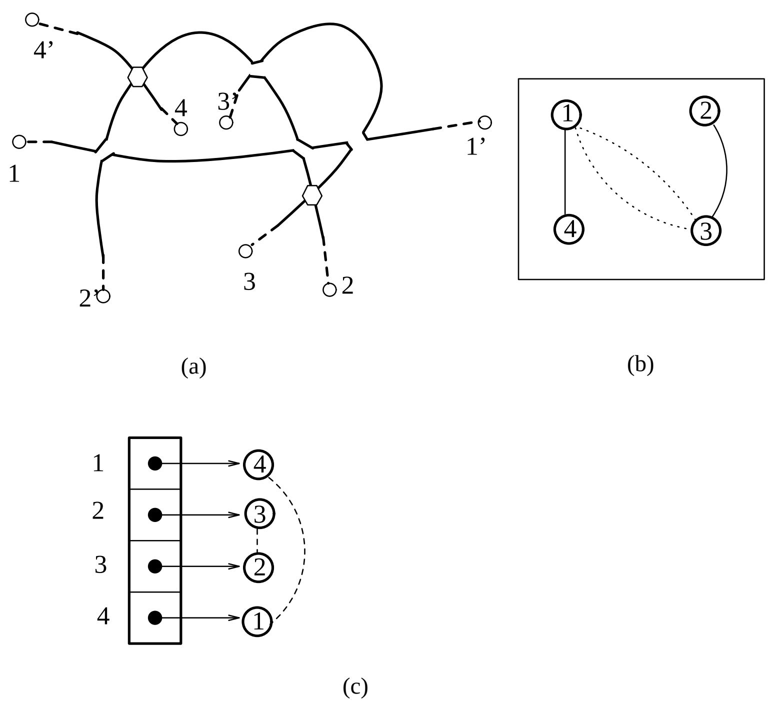


Figure 22: After removing C_{13}

Lemma 4

(1) A traversal along path D_i for $i = 1, 2, \ldots, n$, in $\mathcal{D}$ removes all pockets involving D_i. (2) All the elementary paths on D_i, after untangling and relabeling operations, will not be visited again.

This can be easily proved by induction. After D_1 is traversed, D_1 can only intersect any other path at most once. Therefore it will not be involved in the untangling or relabeling operations for subsequent traversals. Similarly when D_i is traversed, it will only involve paths $D_j, j > i$.

Now let us look at the time complexity of the algorithm. During the untangling procedure, an elementary path will be moved from net to net. It may happen that an elementary path gets visited n times for n nets. As shown in Fig 23, elementary path $\overline{AB}$ initially belongs to net 1 and then if the nets are visited in the order of the net number indicated, the untangling will make $\overline{AB}$ part of net 2, then 3 and so forth. Finally $\overline{AB}$ belongs to net 6.

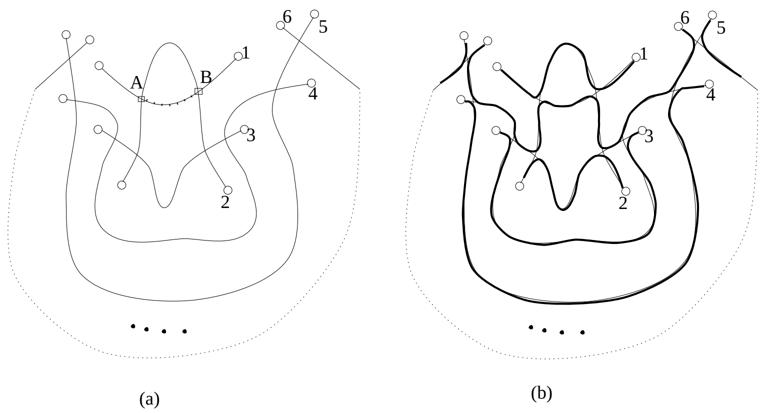


Figure 23: Elementary path AB could be visited at most n times.

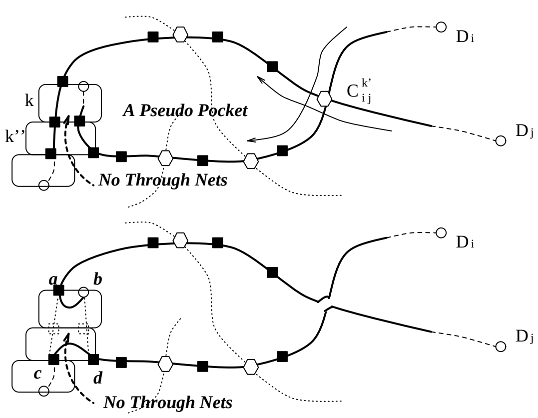


Figure 24: Pseudo Pocket Handling

Lemma 5 *Every elementary path in $\mathcal{D}$ between two crossings is visited at most $2n$ times in the worst case.*

The factor 2 follows from the relabeling operation during untangling operation. The above lemma implies that **Algorithm** CROSSING_MINIMIZATION takes $O(|\mathcal{D}| + n\xi)$ time.

Theorem 2 *Given a global routing topology of a circuit placement $\mathcal{P}$ with m modules and n nets, the total number of crossings satisfying* **Feature 1** *can be minimized in $O(|\mathcal{D}|+n\xi)$ (or $O(mn+n\xi)$) time, where ξ is the total number of crossings in the initial global routing topology.*

Note that this is a very pessimistic estimation. On the average the time bound is expected to be $O(mn + \xi + k\xi) = O(mn + \xi)$ with a repetition factor $k << n$ which is the number of times each elementary path is visited. Note that we still need a postprocessing to finalize the permutations on all the region boundaries. This is done by tracing through each net N_i in $\mathcal{D}$ and relabeling all the junction terminals along the topology. The traversal together takes time $O(mn + \xi)$.

4.5 Feature 2 Handling

We now look at **Feature 2** more closely. In Fig 24, nets N_i and N_j have crossing $C_{ij}^{k'}$ in region $R_{k'}$ and pass through regions R_k and $R_{k''}$ in opposite directions, which is a violation of **Feature 2**. When it ocurrs, we have a *pseudo pocket*. We call these regions (regions R_k and $R_{k''}$) which contain no real crossings but are part of a pseudo pocket *pseudo crossing regions* $\Re(i, j)$ between N_i and N_j. As proposed in [MS95], crossing $C_{ij}^{k'}$ can be removed by reconfiguring the routing as shown in Fig. 24 (i.e., connecting a to b and connecting d to c). However, that this reduces further one more crossing ($C_{ij}^{k'}$) is *true* only if there are no nets going through boundaries $\overline{ab}$ and $\overline{cd}$ formed by the junction terminals a, b, c, and d. In [MS95], guaranteeing **Feature 2** is essential for solving CDP. Ensuring **Feature 2**, not only may increase the number of crossings, but also may create redundant crossings as the next example demonstrates.

In Fig. 25(a) nets N_i and N_j violate **Feature 2**. After reconfiguring the routing to meet the requirement of **Feature 2** we obtain Fig. 25(b). Four more redundant crossings are created and additional iterations of redundant crossing elimination have to be performed. Maintaining **Feature 1** and **Feature 2** is not as straightforward as it may seem. The main difficulty may be due to the fact that eliminating the pseudo pocket changes the *overall* global homotopy specified by the global routing in pseudo crossing regions $\Re(i, j)$. To validate the CDP algorithm of [MS95], we suggest the following modification: consider a **Feature 2** violation in Fig. 26(a), where a crossing C_{ij} between

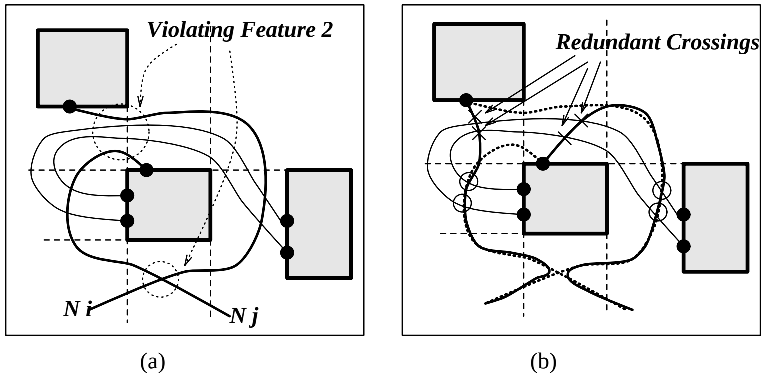


Figure 25: A Feature 2 needs further handling.

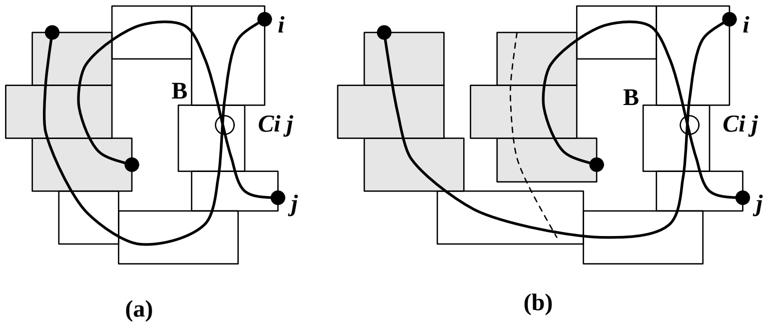


Figure 26: Transformation to validate CDP algorithm from (a) to (b)

net N_i and net N_j and their *B-regions* associated with a boundary B (the regions through which both net N_i and net N_j pass) are shown. We suggest a transformation similar to the one used in [Groe89] by duplicating all the pseudo crossing regions. Because there is no cycle between any two nets, the dominant property among the crossings can be defined and thus the **PERMUTE** procedure of [MS95] is still applicable for the general case. Due to this transformation, the size of the *B-regions* increases. Its increase depends on how many crossings are involved in a cycle. For ξ crossings, the CDP solution takes $O(m\xi)$ time for each crossing and thus $O(m\xi^2)$ for all crossings.

5 Conclusion

In this paper we have presented faster algorithms for CMP which can be applied to optimize the global routing and increase the routability or decrease the number of possible iterations of rip-up/reroute for detailed routing. The CCMP algorithm makes use of the bundle graph transformation, maintaining the topology given by the global routing. The depth-first-search over the bundle graph $\mathcal{BG}$ can retrieve the maximal planar order for placing the crossing, and takes time proportional to the size of the global routing $\mathcal{W}$, i.e., $O(mn)$. The second algorithm solving UCMP makes use of the topological information of the global routing, and constructs a crossing graph $\mathcal{D}$ explicitly, from which redundant crossings are detected and removed. In the worst case, the algorithm takes $O(mn + n\xi)$ time but on the average, it is expected to run much faster. We have implemented CCMP algorithm and UCMP algorithm using *GeoSheet*, a visualization tool, developed at Northwestern University and tested some cases.

Test Cases	#modules	#nets	#crossings (Before)	#crossings (After CCMP)	cpu time(ms) on SPARC 10
$design1$	6	12	56	27	4
$design2$	6	18	92	50	5
$design3$	6	32	344	166	8
$design4$	6	41	625	323	10
$design5$	8	8	28	12	4
$design6$	8	16	120	77	8
$design7$	8	30	367	241	13
$design8$	16	16	88	56	4
$design9$	16	23	201	131	10
$design10$	16	37	478	261	15

Table I. CCMP experimental results

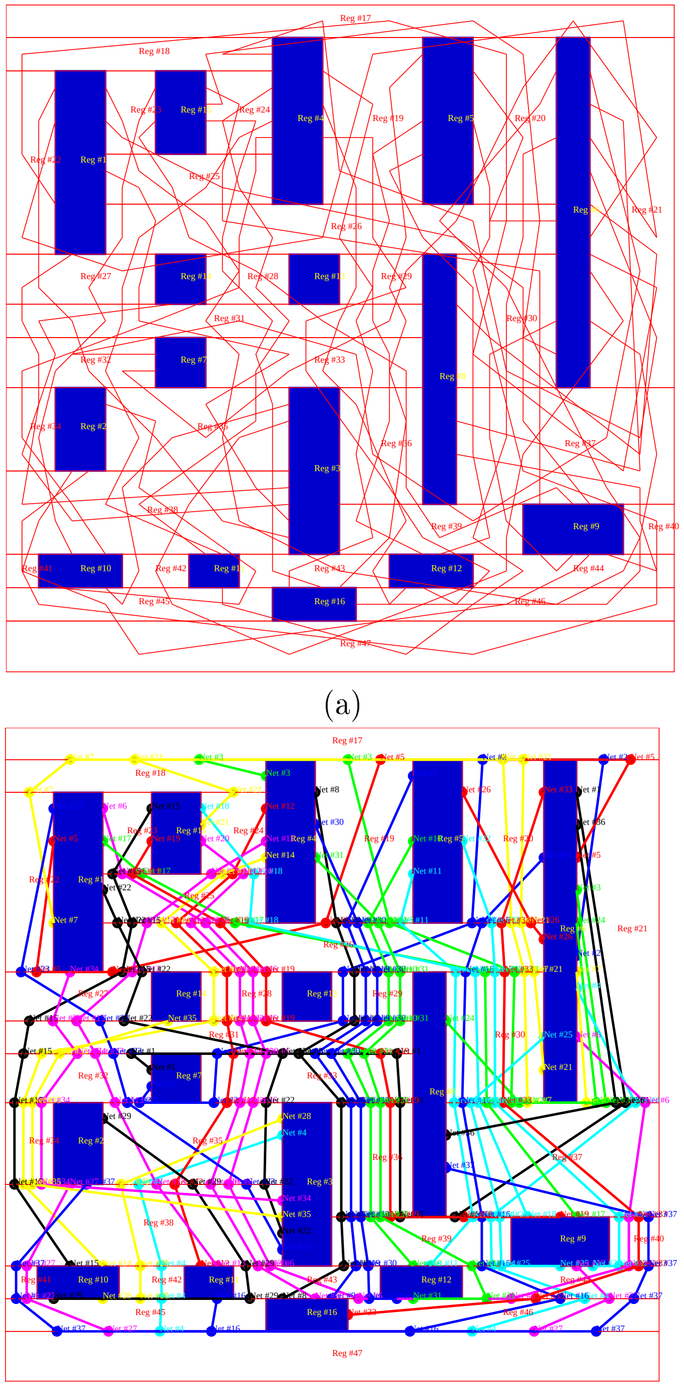
Test Cases	#modules	#nets	#crossings (BEFORE)	#crossings (AFTER)
$design1$	4	6	13	2
$design2$	6	12	31	12
$design3$	9	16	51	21

Table II. UCMP experimental results

For CCMP Fig. 27(a) shows the input global routing and Fig. 27(b) shows the output results. For UCMP Fig. 28 shows the test result of $design3$. Multi-terminal CMP can be transformed into a two-terminal CMP by a technique proposed in [Groe89]. Finally we remark that the problem of maintaining **Feature 2** of [MS95] and minimizes the number of crossings simultaneously remains *open*.

6 Acknowledgement

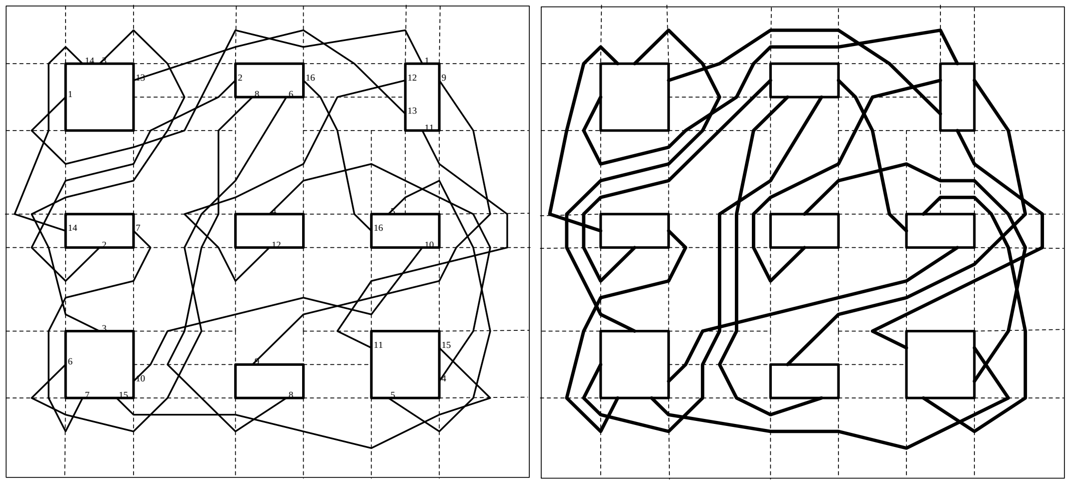
We would like to thank the anonymous referees for their comments that helped improve the presentation of the manuscript and Chen-Hsiao Shih for his assistance in implementing the algorithms.



(a)

(b)

Figure 27: (a) before crossing minimization, (b) after crossing minimization



(a) (b)

Figure 28: (a) before crossing minimization, (b) after crossing minimization

References

[CW91] Y. Cai, and D. F. Wang, "Optimal Channel Pin Assignment", *IEEE Trans. Computer-Aided Design*. vol. CAD-10, no. 11, pp. 1413-1424, Nov. 1991.

[CC91] R.C. Carden IV and C. K. Cheng, "A Global Router using an efficient Approximate Multicommodity Multiterminal Flow Algorithm", *Proc. 28th Design Automation Conf*, pp. 316-321, 1991.

[CHK+83] N. P. Chen, C. P. Hsu, E. S. Kuh, C. C. Chen and M. Takahashi, "BBL: A Building Block Layout System for Custom Chip Design", in *Proc. Int. Conf. Computer-Aided Design*, pp. 40-41, 1983.

[CL96] H.-F. S. Chen, and D. T. Lee, "A Faster Algorithm for Rubber-Band Equivalent Transformation for Planar VLSI Layout", *IEEE Trans. Computer-Aided Design*. vol. CAD-15, no.2, pp. 217-227, Feb. 1996.

[C97] H.-F. S. Chen, "Topological Optimization for VLSI Layout Design", *Ph.D. Thesis, Northwestern Univeristy*,Evanston, IL, June 1997.

[CSW90] C. Chiang, M. Sarrafzadeh, and C. K. Wong, "Global Routing Based on Steiner Minmax Trees", *IEEE Trans. Computer-Aided Design* vol. CAD-9, no. 12, pp. 1318-1325, Dec. 1990.

[Cong91] J. Cong, "Pin assignment with Global Routing for General Cell Designs", *IEEE Trans. Computer-Aided Design*. vol. CAD-10, pp. 1401-1412, Nov. 1991.

[DAK85] W.-M. Dai, T. Asano, and E. S. Kuh, "Routing Region Definition and Ordering Scheme for Building-Block Layout", *IEEE Trans. Computer-Aided Design*. vol. CAD-4, No.3, pp. 189-197, July 1985.

[DDS91] W.-M. Dai, T. Dayan, and D. Staepelaere, "Topological routing in SURF: Generating a Rubber-Band Sketch", *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 45-48, 1991.

[DKJ90] W.-M. Dai, R. Kong and J. Jue, "Rubber Band Routing and Dynamic Data Representation", *Proc. IEEE Int. Conf. Computer- Aided Design*, pp. 52-55, Nov. 1990.

[DSK87] W.-M. Dai, M. Sato, and E. S. Kuh, "A Dynamic and efficient Representation of Building-Block Layout", *Proc. of 24th Design Automation Conf.*, pp. 376-384, 1987.

[Groe89] P. Groenveld, "On global wire ordering for macro-cell routing," Proc. Design Automation Conf., 1989, pp. 155-160.

[HS76] E. Horowitz and S. Sahni, "Fundamentals of Data Structures", Computer Science Press, Potomac, Maryland, 1976.

[KP95] W.-C. Kao, and T.-M. Parng, "Cross Point Assignment with Global Rerouting for General-Architecture Designs", *IEEE Trans. Computer-Aided Design.* vol. CAD-14, No.3, pp. 337-348, March 1995.

[LM85] C. E. Leiserson and F. M. Maley, "Algorithms for routing and testing routability of Planar VLSI Layouts", *Proc. 17th Ann. ACM Sym. Theory Computing*, pp. 69-78, 1985.

[Leng90] T. Lengauer, "Combinatorial Algorithms for Integrated Circuit Layout", John Wiley & Sons, 1990.

[LM84] J. T. Li and M. Marek-Sadowska, "Global Routing for Gatearray", *IEEE Trans. Computer-Aided Design.* vol. CAD-4, pp. 298-307, Oct., 1984.

[MS95] M. Marek-Sadowska, and M. Sarrafzadeh, "The Crossing Distribution Problem," *IEEE Trans. Computer-Aided Design*, vol. CAD-14, no. 4, pp. 423-433, April 1995.

[PS85] F. Preparata and M. Shamos, *Computational Geometry, an introduction*, New York: Springer-Verlag, 1985.

[Oh86] T. Ohtsuki, (Ed.) "Layout Design and Verification", Advances in CAD for VLSI, vol. 4, North-Holland, 1986.