

Wire Planning with Bounded Over-the-Block Wires ^{*}

Hua Xiang
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
huaxiang@us.ibm.edu

I-Min Liu
Cadence Design System
San Jose, CA 95134
imliu@cadence.com

Martin D. F. Wong
ECE Dept, UIUC
Urbana, IL, 61801
mdfwong@uiuc.edu

Abstract

Hierarchical approach greatly facilitates large-scale chip design by hiding distracting details in low-level objects. However, the low-level designs have to have a global view of high-level object connections so that some resources can be allocated in advance, and this makes wire planning an important issue in physical design. In this paper, we present two exact polynomial-time algorithms for wire planning with bounded over-the-block wires. The constraints on over-the-block wires help the longest over-the-block wires within a block to satisfy signal integrity without buffer inserted. Both algorithms guarantee to find an optimal routing solution for a two-pin net as long as one exists. One requires less memory, while the other may take less running time when processing a large number of nets. According to different application requirements, users can choose an appropriate one.

Category: B.7.2 [Integrated Circuits]: Design Aids - Placement and routing; J.6 [Computer Applications]: Computer-Aided Engineering - Computer-aided design

Terms: Algorithms, Design

Keywords: routing, over-the-block, wire planning

1. Introduction

Due to the extreme complexity in System-on-Chip (SoC) design, the hierarchical approach is widely used. By hiding the vast amount of distracting details in low-level objects, design problems can be greatly simplified such that those problems can be solved in efficient ways.

In high-level design, a design is partitioned into several functional blocks, and then each functional block can be designed independently in low levels. However, an implicit constraint is that all these modules have to get connected in a certain way so that they can accomplish the required functionality as a whole. Moreover, as technology scales down, interconnect delay, especially global interconnect delay, has become the dominant factor in achieving high performance in deep sub-micron design. Therefore, wire planning, which plans the routing of global interconnect among macro blocks, has become an important stage in physical design.

Recently there are work on extending buffering algorithm to consider route [4, 5, 6, 7]. We think in general it is the right direction to consider the impact of routing on buffering, especially in the final implementation stages. However, explicit buffering consideration in routing at early design stages is very CPU intensive. Furthermore, the final buffering can be done only based on the detailed RC data, which can only be obtained after the low-level block implementation is completed. Before low-level block implementation, it does not make sense to put down all buffers – what we need is to reserve “proper” routing resource for the global route when doing block implementation.

In our work, we abstract buffering by looking at the problem at a higher lever, and consider the problem of Wire Planning with bounded over-the-block wires (WP). Informally, the problem can be described as follows. Given a placement of macro blocks, some large blocks are evenly divided into several subblocks. A two-pin net is to be routed by going through these blocks/subblocks, i.e., to find a block/subblock sequence as the routing of the net. However, some blocks (or some subblocks) are routing obstacles; while some blocks, such as IPs (Intellectual Property) whose internal structures have been fixed, only allow routing but no buffers can be inserted. We call the latter kind of blocks “routing-block”. Since a routing-block cannot hold any buffers, the longest over-the-block wires in it have to be limited within a certain range since if the distance between two buffers is large, the signal slew rate is slow which in turn may cause signal integrity problems. In WP problem, each routing-block has an interconnect bound such that the longest over-the-block wires in the routing block can only cross a certain number of subblocks. The bound is in line with designers’ practice in keeping long routes “buffer-able” to meet timing and transition time constraints. Different routing-blocks may set different bounds.

Figure 1 illustrates an example of 16 macro blocks. Since some macro blocks are big, they are partitioned into several small subblocks in order to get more accurate estimation. Blocks B_2 , B_3 , B_9 , B_{10} , B_{11} and B_{12} are divided into 1×3 , 2×3 , 2×1 , 3×1 , 2×1 and 3×3 subblocks respectively. B_4 , B_6 and B_{14} as well as a subblock of B_3 are obstacles. B_3 , B_{10} and B_{12} are routing-blocks and the longest over-the-block wires in these three blocks should not go across more than 2 subblocks. One two-pin net is to be routed between B_1 and B_{16} . However, the routing illustrated

^{*}This work was partially supported by the National Science Foundation under grant CCR-0306244.

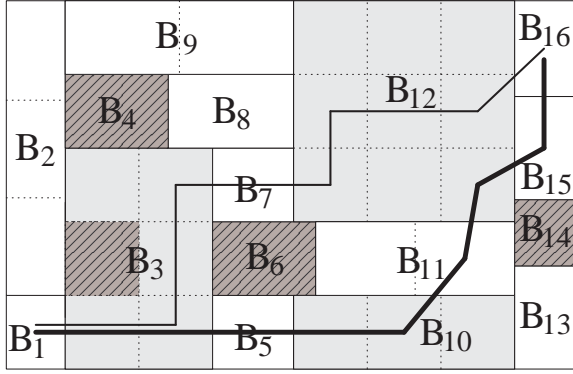


Figure 1: A placement with 16 macro blocks. Dark blocks are obstacles while gray ones can be used for routing but cannot hold buffers. The longest over-the-block wires in the three routing-blocks can cross at most 3 subblocks. The routing between B_1 and B_{16} illustrated by thin lines is not valid; while the routing shown by wide lines is a feasible solution.

by thin lines does not satisfy the interconnect constraints since the over-the-block wires in both blocks B_3 and B_{12} cross 4 subblocks. On the other hand, the routing illustrated by wide lines is a valid solution. For short, let OB-wire refer to over-the-block wire.

In this paper, we present two exact polynomial-time algorithms for wire planning with bounded over-the-block wires. Both algorithms guarantee to find a feasible routing solution with minimum wire length as long as a solution exists. Furthermore, both algorithms use shortest path algorithm, but the constructions of the underlying graphs, which incorporate interconnect constraints, are different. One approach requires less memory, but the other approach may take less time to adjust the graph after routing one net. According to different requirement, users can choose an appropriate algorithm to solve the problem.

The rest of the paper is organized as follows. Section 2 defines the WP (Wire Planning with Bounded Over-the-Block Wires) problem. In Section 3, we present two exact polynomial-time algorithms based on the shortest path algorithm with different graph constructions. Finally we show the experimental results in Section 4 and conclude the paper in Section 5.

2. Problem Formulation

A placement of m macro blocks $\mathcal{B} = \{B_1, \dots, B_m\}$ is given. Since some blocks are large, they are divided into several subblocks. Suppose block B_i is evenly divided into $p_i \times q_i$ blocks, and these subblocks form a $p_i \times q_i$ block grid. Let $R_i = p_i \times q_i$ be the number of subblocks of B_i , and r_i^j ($j = 1, \dots, R_i$) refer to a subblock of block B_i . For convenience, if B_i is not divided into subblocks, we still say it is divided into $p_i \times q_i$ subblocks where $p_i = q_i = 1$.

Among these m blocks, some blocks are routing-blocks which only allow over-the-block routing but no buffers can be inserted.

Then an interconnect bound d_i is set for each routing-block B_i such that all over-the-block wires within B_i go through at most d_i subblocks.

Furthermore, some blocks even do not allow over-the-block routing and they become routing obstacles. Some subblocks within a block can also be obstacles. Finally if a block is neither a routing block, nor an obstacle, then buffers can be inserted and OB-wires in the block are not constrained.

With all these requirements, the target is to find the shortest over-the-block routing for a two-pin net such that the routing satisfies the interconnect constraints. The routing wire length is measured by the Manhattan distance between the centers of two macro blocks.

3. WP Algorithm

If no interconnect constraints are considered, the routing of a two-pin net can be interpreted as a shortest path between two blocks. But when interconnect constraints are involved, they prevent us from applying shortest path algorithm directly. Therefore, our approaches are to construct a graph so that interconnect constraints can be incorporated in the graph. Then the shortest path algorithm is called to find the optimal routing solution. The main scheme of a WP algorithm is as follows:

Algorithm WP-Algorithm ()

1. Construct a graph incorporating constraints;
2. Apply shortest path algorithm on the graph;
3. Derive routing solution for the given net;

In the following two subsections, we present two ways of constructing the graph for a given WP problem.

3.1 WP-Path Algorithm

As we notice that all interconnect constraints are confined to OB-wires within routing-blocks. Therefore, the main idea of WP-Path algorithm is to explore all legal connections between any two boundary subblocks within the same routing-block.

Figure 2 illustrates an example. Figure 2 (a) shows a routing-block B_i and it is divided into 3×3 subblocks. Suppose the interconnect bound is 3. In Figure 2 (b), each subblock is represented by a node and the edges imply that the corresponding two blocks can be connected by an OB-wire which crosses at most 3 subblocks. In other words, any valid OB-wire in B_i corresponds to an edge in Figure 2 (b).

However, if two or more successive edges are selected for one OB-wire, the interconnect constraint may not be satisfied. For example, in Figure 2 (b), if edges (r_i^1, r_i^3) and (r_i^3, r_i^9) are selected, then the corresponding OB-wire goes through 5 subblocks (i.e., $r_i^1, r_i^2, r_i^3, r_i^6$ and r_i^9). In order to avoid selecting successive edges related to a routing-block, one node is splitted into two nodes which are called in-node and out-node respectively. Figure 3 illustrates an

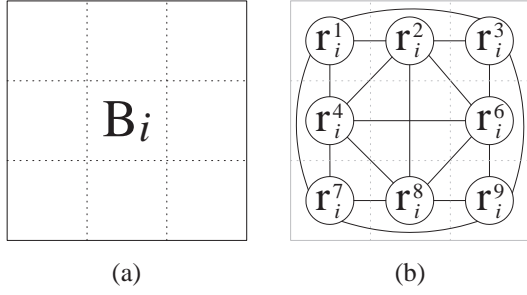


Figure 2: (a) A routing-block B_i is divided into 3×3 subblocks. Suppose the interconnect bound is 3. (b) A graph denotes all valid OB-wires within B_i .

example. Two nodes r_i^j and r_i^k belong to the same routing-block B_i , and there is an edge \hat{e}_2 connecting the two nodes. But edge \hat{e}_1 connects r_i^j to a node which does not belong to block B_i . Then node r_i^j is splitted to an in-node v_i^j and an out-node \bar{v}_i^j . Similarly for node r_i^k . Also the undirected edge \hat{e}_2 becomes two directed edges e_2 and e_2' pointing from an in-node to an out-node. Furthermore, since edge \hat{e}_1 connects a node not belonging to B_i , it becomes two edges e_1 and e_1' . e_1 is incident to the in-node v_i^j , while e_1' is incident from \bar{v}_i^j .

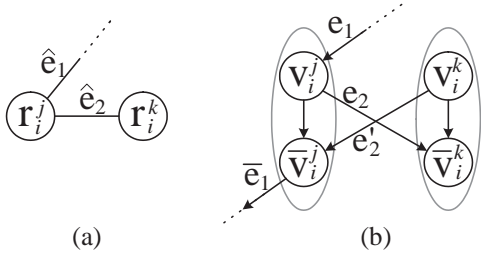


Figure 3: (a) Two nodes r_i^j and r_i^k are within the same routing-block B_i . \hat{e}_1 connects to a node which does not belong to block B_i . (b) Each node is splitted into an in-node and an out-node, and each undirected edge is represented by two directed edges.

Based on the above strategy, we construct a directed graph **Path Graph** $G_p = (V_p, E_p)$ for a WP problem as follows.

1. Nodes $V_p = V_{ps} \cup V_{in} \cup V_{out}$ where

$$V_{sp} = \{u_i^j | i = 1, \dots, m; r_i^j \text{ is a subblock of } B_i, \text{ and } B_i \text{ is neither an obstacle nor a routing-block.}\}$$

$$V_{in} = \{v_i^j | i = 1, \dots, m; r_i^j \text{ is not an obstacle and it is a subblock on the boundary of } B_i.\}$$

$$V_{out} = \{\bar{v}_i^j | i = 1, \dots, m; r_i^j \text{ is not an obstacle and it is a subblock on the boundary of } B_i.\}$$

For convenience, node $v_i^j \in V_{in}$ is called in-node, while node $\bar{v}_i^j \in V_{out}$ is called out-node. Without misunderstanding, the corresponding subblock of a node v_i^j (or \bar{v}_i^j or u_i^j) is r_i^j .

2. Edges $E_p = E_{pa} \cup E_{pb} \cup E_{pc} \cup E_{pd} \cup E_{pe}$ where

$$\begin{aligned} E_{pa} &= \{(u_i^j, v_k^l) | u_i^j \in V_{ps}, v_k^l \in V_{in}, r_i^j \text{ and } r_k^l \text{ are adjacent.}\}; \\ E_{pb} &= \{(\bar{v}_i^j, u_k^l) | \bar{v}_i^j \in V_{out}, u_k^l \in V_{ps}, r_i^j \text{ and } r_k^l \text{ are adjacent.}\}; \\ E_{pc} &= \{(u_i^j, u_k^l) | u_i^j, u_k^l \in V_{ps}, u_i^j \neq u_k^l, r_i^j \text{ and } r_k^l \text{ are adjacent.}\}; \\ E_{pd} &= \{(v_i^j, \bar{v}_i^j) | v_i^j \in V_{in}, \bar{v}_i^j \in V_{out}\}; \\ E_{pe} &= \{(v_i^j, \bar{v}_i^k) | v_i^j \in V_{in}, \bar{v}_i^k \in V_{out}, j \neq k, \text{ and there exists routing between two subblocks } r_i^j \text{ and } r_i^k \text{ such that the routing goes across at most } d_i \text{ subblocks.}\} \end{aligned}$$

Edges in $E_{pd} \cup E_{pe}$ connects nodes whose corresponding subblocks belong to the same routing-blocks. As we notice that those edges are always incident from an in-node to an out-node. Therefore, no path can take two successive edges related to the same routing-block. Furthermore, any valid OB-wire in a routing-block corresponds to an edge in $E_{pd} \cup E_{pe}$. On the other hand, one edge in $E_{pd} \cup E_{pe}$ implies that at least one valid OB-wire exists between the two corresponding subblocks. For convenience, we say an edge of $E_{pd} \cup E_{pe}$ is within the related routing-blocks.

3. Edge Cost

$$\text{If } e \in E_{pa} \cup E_{pb} \cup E_{pc}, \text{ and } e = (\alpha_i^j, \beta_k^l),$$

$$C(e) = |x_i^j - x_k^l| + |y_i^j - y_k^l| \text{ where}$$

(x_i^j, y_i^j) and (x_k^l, y_k^l) are the center coordinates of subblocks r_i^j and r_k^l ;

$$\text{If } e \in E_{pd}, C(e) = 0;$$

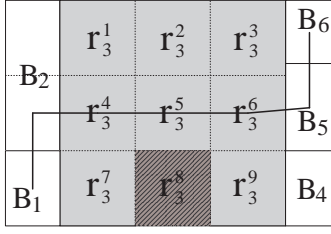
$$\text{If } e \in E_{pe}, \text{ and } e = (v_i^j, \bar{v}_i^k),$$

$C(e)$ is the shortest routing length within routing-block B_i between two subblocks r_i^j and r_i^k .

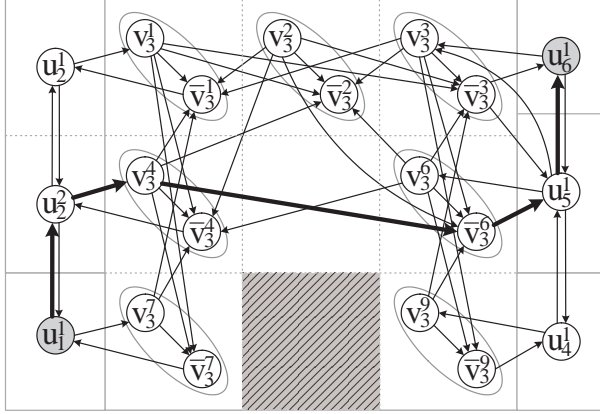
Figure 4 illustrates an example. Figure 4 (a) includes 6 macro blocks. B_3 is a routing-block and it is divided into 3×3 subblocks. Especially, one subblock r_3^8 of B_3 is obstacle. Also block B_2 is divided into 1×2 subblocks. Suppose the interconnect bound for B_3 is 3 and we want to find the routing between B_1 and B_6 under the interconnect constraint. Figure 4 (b) is the corresponding Path Graph G_p . Since r_3^8 is obstacle and r_3^5 is not a boundary subblock, they are not included in G_p .

In Figure 4 (b), edges within ellipses belong to E_{pd} . The two nodes in one ellipse are the in-node and out-node corresponding to a subblock. Since this kind of edges is used to connect in-nodes and out-nodes, the cost is zero. For other edges within the routing-block B_3 , i.e., edges belonging to E_{pe} , an edge (v_3^j, \bar{v}_3^i) ($i, j = 1, \dots, 9; v_3^j \in V_{in}; \bar{v}_3^i \in V_{out}; i \neq j$) is added if subblock r_3^i can reach r_3^j by crossing at most 3 three subblocks of B_3 . For example, r_3^7 can reach r_3^1 by going through r_3^4 . So one edge (v_3^7, \bar{v}_3^1) is added. But the shortest routing between r_3^7 and r_3^9 has to go through r_3^4, r_3^5 and r_3^6 . Therefore (v_3^7, \bar{v}_3^9) is not inserted in G_p . The cost for this kind of edges is the shortest routing length between the two subblocks.

Once the Path Graph G_p is constructed, we can apply the shortest path algorithm [1, 3] to decide the routing of a net. In this ex-



(a)



(b)

Figure 4: (a) A WP problem with 6 macro blocks. Routing-block B_3 is divided into 3×3 subblocks, and suppose the interconnect constraint is 3. One net is to be routed between blocks B_1 and B_6 . The solid lines indicate a routing solution. (b) The corresponding Path Graph G_p . The wide lines illustrate a shortest path from u_1^1 to u_6^1 .

ample, we plan to route one net between B_1 and B_6 . Therefore, we let u_1^1 be the starting point and u_6^1 be the ending point. The shortest path from u_1^1 to u_6^1 is illustrated by wide lines in Figure 4 (b). And it is easy to derive the over-the-block routing from this solution as shown in Figure 4 (a).

In order to construct a Path Graph G_p , we need to find all valid connections within each routing-block, i.e., how to create edges in E_{pe} . For each routing-block B_i , an undirected graph \tilde{G}_i is set up by presenting each subblock, which is not obstacle, as a node and adding edges according to block adjacency. The cost of all edges is 1. Then apply all-pair-shortest path algorithm on \tilde{G}_i . If the distance between two boundary subblocks is less than or equal to the interconnect bound of B_i , then add edges in G_p accordingly.

Suppose each block B_i is divided into R_i ($R_i = p_i \times q_i$) subblocks. If B_i is a routing-block, the nodes related to B_i are $O(p_i + q_i)$; otherwise, the number of nodes is (R_i) . Therefore, we get

$$|V_p| = O\left(\sum_{i=1}^m R_i\right)$$

For edges in G_p , $|E_{pa} \cup E_{pb} \cup E_{pc}| = O(|V_p|)$ since each edge connects two adjacent blocks/subblocks, and the adjacency rela-

tionship among blocks/subblocks on a plane constitutes a planar graph. Also $|E_{pd}| = |V_{in}| = |V_{out}|$. Finally $|E_{pe}| = O(\sum_{i=1}^m \min\{(p_i + q_i)^2, d_i \cdot (p_i + q_i)\})$ where d_i is the interconnect bound for block B_i . Without loss of generality, we assume $d_i = O(p_i + q_i)$. Then $|E_{pe}| = O(\sum_{i=1}^m (p_i + q_i)^2)$. Therefore, we get

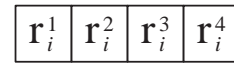
$$|E_p| = O\left(\sum_{i=1}^m (p_i + q_i)^2\right)$$

However, during the construction of Path Graph G_p , we first need to set up $\tilde{G}_{pi} = (\tilde{V}_{pi}, \tilde{E}_{pi})$ for each routing-block B_i in order to decide edges in E_{pe} . Since $|\tilde{V}_{pi}| = O(R_i)$, $|\tilde{E}_{pi}| = O(R_i)$, and all-pair shortest path algorithm takes $O(|\tilde{V}_{pi}| |\tilde{E}_{pi}| + |\tilde{V}_{pi}|^2 \log |\tilde{V}_{pi}|)$ running time [1, 3], the creation of edges inside routing-block B_i takes $O(R_i^2 \log R_i)$ running time. Therefore, the total construction time for G_p is $O(\sum_{i=1}^m R_i^2 \log R_i)$.

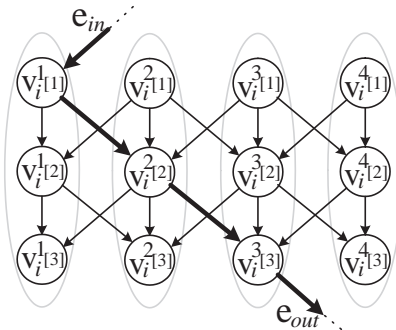
Once the Path Graph $G_p = (V_p, E_p)$ is constructed, the single-source shortest path algorithm can be used to find the routing with over-the-block wiring constraint for a two-pin net, and it can be accomplished in $O(|V_p| |E_p|)$ time.

3.2 WP-Split Algorithm

In this section, we propose another algorithm by constructing a directed graph **Split Graph** G_s . The main idea of this approach is to represent a subblock in a routing-block, which is not an obstacle, by d_i nodes. Then each path segment in the routing-block starts with a node whose index is 1, and ends at a node whose index is d_i . Since all nodes with index d_i only have out-going edges pointing to a node which is not related to B_i , the length of a path segment is forced to be d_i . Figure 5 illustrates an example.



(a)



(b)

Figure 5: (a) r_i^1, r_i^2, r_i^3 and r_i^4 are subblocks of a routing-block B_i . Suppose the interconnect bound for B_i is 3. (b) Each subblock is represented by a node array. A path segment in B_i must go from a node with index 1 to a node with index 3.

The construction of a Split Graph $G_s = (V_s, E_s)$ is as follows. If

B_i is a routing-block, let $L_i = d_i$ where d_i is the interconnect bound of B_i . Otherwise, let $L_i = 1$.

1. Nodes $V_s = \cup_{i=1}^m V_{si}$ where

$$V_{si} = \{v_i^j[k] | r_i^j \text{ is a subblock of } B_i \text{ and it is not an obstacle}; \\ k = 1, \dots, L_i\}.$$

2. Edges $E_s = E_{sa} \cup E_{sb} \cup E_{sc}$ where

$$E_{sa} = \{(v_i^j[k], v_i^j[k+1]) | v_i^j[k], v_i^j[k+1] \in V_{si}, k = 1, \dots, L_i - 1\};$$

$$E_{sb} = \{(v_i^j[k], v_i^j[k+1]) | v_i^j[k], v_i^j[k+1] \in V_{si}, k = 1, \dots, L_i - 1, \\ j \neq l, \text{ and subblocks } r_i^j \text{ and } r_i^l \text{ are adjacent.}\};$$

$$E_{sc} = \{(v_i^j[L_i], v_k^l[1]) | v_i^j[L_i] \in V_{si}, v_k^l[1] \in V_{sk}, \\ \text{and two subblocks } r_i^j \text{ and } r_k^l \text{ are adjacent,} \\ \text{but do not belong to the same routing-block.}\}$$

As we notice that edges belonging to E_{sb} always connect nodes whose indexes increase by one. Then a path segment within a routing-block B_i is always from a node with index 1 to a node with index L_i . In other words, any OB-wire in B_i cannot exceed the interconnect bound d_i .

3. Edge Cost

If $e \in E_{sa}$, $C(e) = 0$;

If $e \in E_{sb}$, $e = (v_i^j[k], v_i^l[k+1])$, $C(e) = |x_i^j - x_i^l| + |y_i^j - y_i^l|$ where (x_i^j, y_i^j) and (x_i^l, y_i^l) are the center coordinates of subblocks r_i^j and r_i^l ;

If $e \in E_{sc}$, $e = (v_i^j[L_i], v_k^l[1])$, $C(e) = |x_i^j - x_k^l| + |y_i^j - y_k^l|$ where (x_i^j, y_i^j) and (x_k^l, y_k^l) are the center coordinates of subblocks r_i^j and r_k^l ;

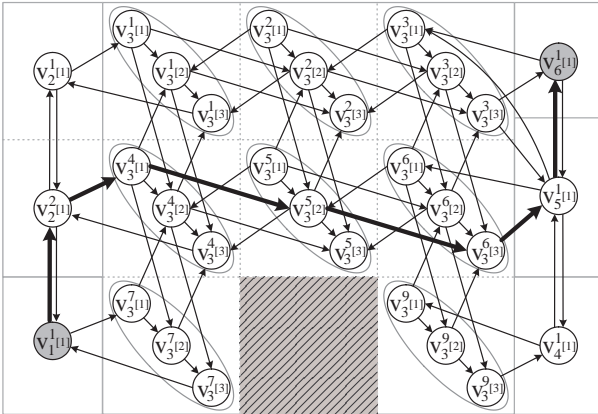


Figure 6: The corresponding Split Graph for Figure 4 (a). The wide lines illustrate a shortest path from $v_1[1]$ to $v_6[1]$.

We still use the example of Figure 4 (a) to illustrate our approach. Since the interconnect bound for B_3 is 3, three nodes are created for each subblock excluding obstacles. Then edges are added among these nodes. Edges in ellipses belong to E_{sa} and their cost is 0. For any other edge e inside B_3 , i.e., $e \in E_{sb}$, the index of

its source node is less than that of its target node, and the difference is 1. On the other hand, for subblocks of B_2 , they are represented by only one node since B_2 is not a routing-block and has no interconnect constraint. Finally, if two subblocks are adjacent and they do not belong to the same routing-block, then the tail of a node array points the head of another node array. This kind of edges belongs to E_{sc} . The wide lines shows a shortest path from $v_1[1]$ to $v_6[1]$, and it corresponds to the routing in Figure 4 (a).

Since a routing-block is divided into $R_i = p_i \times q_i$ subblocks and each subblock r_i^j is represented by d_i nodes, $|V_s| = O(\sum_{i=1}^m d_i \times R_i)$. $|E_{sa}| = O(|V_s|)$. $|E_{sb}| = O(|V_s|)$ and $|E_{sc}| = O(|V_s|)$. Therefore $|E_s| = O(|V_s|)$.

Once the Split Graph $G_s = (V_s, E_s)$ is constructed, the single-source shortest path algorithm can be used to find the over-the-block routing for a two-pin net, and it can be accomplished in $O(|V_s||E_s|) = O((\sum_{i=1}^m d_i \times R_i)^2)$ time.

3.3 Comparison

In the above, we have presented two approaches to set up a graph which implies interconnect constraints. The comparison between the two graphs are listed in Table 1. $R_i = p_i \times q_i$ is the number of subblocks in a block, and d_i is the interconnect bound of a routing-block B_i . Without loss of generality, we assume $d_i = O(p_i + q_i)$.

Table 1: Algorithm Comparison

Algorithm	WP-Path	WP-Split
Nodes	$O(\sum_{i=1}^m R_i)$	$O(\sum_{i=1}^m d_i \times R_i)$
Edges	$O(\sum_{i=1}^m (p_i + q_i)^2)$	$O(\sum_{i=1}^m d_i \times R_i)$
Setup	$O(\sum_{i=1}^m R_i^2 \log R_i)$	$O(\sum_{i=1}^m d_i \times R_i)$
Path	$O((\sum_{i=1}^m R_i)(\sum_{i=1}^m (p_i + q_i)^2))$	$O((\sum_{i=1}^m d_i \times R_i)^2)$
Adjust	$O(R_i^2 \log R_i)$	$O(d_i^2)$

The advantage of WP-Path algorithm is that it requires less memory since the underlying graph is smaller. However, it takes longer time to construct the graph. ‘‘Setup’’ in the table shows the construction time. ‘‘Path’’ is the time to search for a shortest path based on the graph. And since the size of the underlying graph of WP-Path is smaller than that of WP-Split, it takes shorter time to find a routing solution for one net.

Furthermore, after routing one net, some subblocks may become routing obstacles since the net consumes all routing resource. Therefore, we have to judge whether the edges within a routing-block are still valid. For example, in Figure 4 (a), the routing of the net goes through routing-block B_3 . Suppose subblock r_3^4 can accommodate only one net. After routing this net, r_3^4 becomes a routing obstacle. Then in Figure 4 (b), several edges such as (v_3^1, v_3^7) and (v_3^2, v_3^4) inside B_3 are not valid any more. For the two approaches, WP-Path have to recalculate edges within every routing-block along the path, while WP-Split only needs to remove some edges from the

graph. The time required for adjusting edges within one routing-block is shown in Table 1 “Adjust”. In general, WP-Split is faster for graph adjustment, and it is suitable for handling a large number of nets.

4. Experimental Results

Our algorithms were implemented in C++ on PC (733MHz) with 128M memory. We tested WP-Path and WP-Split algorithms on three test files which were generated randomly. We compared the results of our two approaches with another shortest path approach which finds a shortest path from the source node to the end node without considering interconnect constraints. If the interconnect constraint is not satisfied, the net is discarded. Each time, one net is selected and is to be routed. Furthermore, we assume that all sub-blocks can accommodate a large number of nets so that the routing of one nets is not affected by others. In this way, we can concentrate on the routing quality of WP algorithms. Both WP-Path and WP-Split algorithms guarantee to find feasible routing as long as one solution exists. As shown in Table 2, all nets can be routed by our approaches, while less than 50% of the nets can find feasible routing by the simple shortest path approach.

Table 2: Test results of WP-Path and WP-Split

File		test1	test2	test3
Macro Blocks		50	100	150
Routing-Blocks		7	12	18
Nets		350	1600	3500
Total Subblocks		1337	1736	2604
Obstacles		18	48	131
Shortest Path	Time(s)	11	23	76
	(per net)	(0.031)	(0.014)	(0.022)
Path	Nets	165	613	1211
		(47.14%)	(38.31%)	(34.6%)
WP-Path	Time(s)	9	104	202
	(per net)	(0.026)	(0.065)	(0.058)
WP-Path	Nets	350	1600	3500
		(100%)	(100%)	(100%)
WP-Split	Time(s)	18	91	227
	(per net)	(0.051)	(0.057)	(0.065)
WP-Split	Nets	350	1600	3500
		(100%)	(100%)	(100%)

The underlying graph of WP-Path usually has smaller size than that of WP-Split. For the test file “test2”, the number of nodes in WP-Path is about 1200 and edges are about 25,000; while WP-Split needs about 16,500 nodes and 73,000 edges. WP-Path gains in memory usage. However, WP-Split has the advantage that it can make changes to the underlying graph more easily after routing one

net. Therefore, if a larger number of nets are to be processed, WP-Split may requires less running time than that of WP-Path. Figure 7 shows the relationship between the number of nets and the running time for test file “test2”. As the number of nets increases, the running time of WP-Split becomes shorter than that of WP-Path.

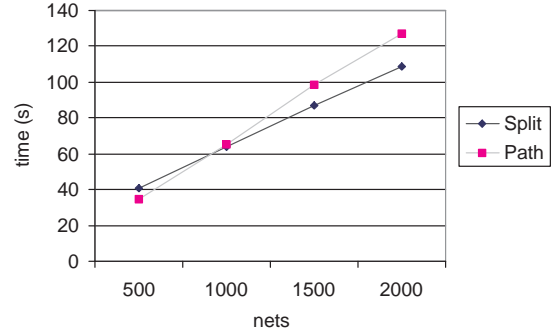


Figure 7: The comparison of WP-Path and WP-Split on the relationship between running time and the number of nets.

5. Conclusion

In this paper, we present two exact polynomial-time algorithms for wire planning with bounded over-the-block wires. Both algorithms guarantee to find an optimal routing solution for a two-pin net as long as one exists. One requires less memory, while the other may take less running time when processing a large number of nets. According to different application requirements, users can choose an appropriate one.

6. References

- [1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows*, Prentice Hall, 1993.
- [2] C.J. Alpert, A. Devgan, and S.T. Quay, Buffer insertion for noise and delay optimization, *Proc. 35th IEEE/ACM Design Automation Conf.*, pp. 362C367, 1998.
- [3] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, The MIT Press, 1992.
- [4] L.D. Huang, M.H. Lai, D.F. Wong, and Y.X. Gao, maze routing with buffer insertion under transition time constraints, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22-1, pp 91-95, Jan 2003.
- [5] J. Lillis, C.K. Cheng, and T.T. Lin, Optimal and efficient buffer insertion and wire sizing, in *CICC*, pp 259-262, 1995.
- [6] L.P.P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *ISCAS*, pp. 865-868, 1990.
- [7] H. Zhou, D. F. Wong, I-M. Liu, and A. Aziz, Simultaneous routing and buffer insertion with restrictions on buffer locations. *IEEE Transactions on Computer-Aided Design*, July, 2000.