TRACK ASSIGNMENT CONSIDERING CROSSTALK-INDUCED PERFORMANCE DEGRADATION

A Thesis

by

QIONG ZHAO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2012

Major Subject: Computer Engineering

TRACK ASSIGNMENT CONSIDERING CROSSTALK-INDUCED PERFORMANCE DEGRADATION

A Thesis

by

QIONG ZHAO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Jiang Hu
Committee Members,	Peng Li
	Rabi N. Mahapatra
Head of Department,	Costas N. Georghiades

May 2012

Major Subject: Computer Engineering

ABSTRACT

Track Assignment Considering Crosstalk-Induced Performance Degradation.

(May 2012)

Qiong Zhao, B.S., Beijing University of Posts and Telecommunications

Chair of Advisory Committee: Dr. Jiang Hu

Track assignment is a critical step between global routing and detailed routing in modern VLSI chip designs. It greatly affects some very important design characteristics, such as routability, via usage and timing performance. Crosstalk, which is largely decided by wire adjacency, has significant impact on interconnect delay and circuit performance. Therefore, the amount of crosstalk should be restrained in order to satisfy timing constraints. In this work, a track assignment approach is proposed to control crosstalk-induced performance degradation. The problem is formulated as a Traveling Salesman Problem (TSP) and solved by a graph-based heuristic. The proposed approach is implemented and tested on benchmark circuits from the ISPD2011 contest and the experimental results are quite promising.

To my family

ACKNOWLEDGEMENTS

I would like to thank all of those who encouraged me and helped me during my study and research at Texas A&M University.

Especially, I would like to extend my heartfelt gratitude to my advisor, Dr. Jiang Hu, who gave me constant guidance as well as warm encouragement throughout this research project. He was always patient and kind whenever I had questions or met problems. Sometimes I got stuck at some point, and he would explain the problem in every detail until I understand. I could not have completed this thesis without his help and guidance. I also would like to thank Dr. Peng Li and Dr. Rabi Mahapatra for being my committee members, and for their suggestions and support on this project.

I would like to thank Dr. Zhuo Li from IBM for his constructive suggestions on this project, and Dr. Natarajan Viswanathan from IBM for his very helpful information and explanation on the benchmark circuits.

I would also like to thank my parents for their ceaseless support and encouragement, without which I would never have been able to complete my graduate study. I also thank my husband who gave me endless love and understanding.

Last but not least, thanks also to my friends and colleagues and the department faculty and staff for giving me a warm and kind environment for study, and for helping me in every aspect during my life at Texas A&M University.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

1. INTRODUCTION

1.1 Routing in VLSI Circuits

In VLSI design, routing is an important and complex step. Global routing, followed by detailed routing, is performed after placement is done, and it outputs the rough locations of nets. Exact location and layer for each segment of a net is decided by detailed routing.

In the global routing step, the whole routing region is divided into grids. Each grid, or tile, is a rectangular region which provides limited routing resource in both horizontal and vertical direction. Fig. 1 illustrates the gridded routing region. Routing resource is represented by the number of tracks across each edge of a grid.

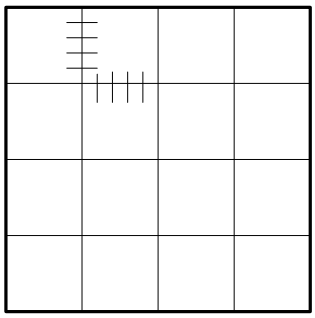

Fig. 1. A routing region divided into 4x4 grids. Each grid has a capacity of 4 horizontal tracks and 4 vertical tracks.

This thesis follows the style of *IEEE Transactions on Computer Aided Design of Integrated Circuits and System*.

1.2 Track Assignment

Due to the variety of constraints, detailed routing can be very complex. Thus, an intermediate step is needed between global routing and detailed routing, i.e., track assignment [1]. Taking the global routing result as input, this intermediate stage assigns a track for each wire segment. Several types of constraints can be taken into account in the track assignment procedure, such as wire conflict, timing constraints, and delay increment. Fig. 2 and Fig. 3 show a simple example of track assignment. The original positions of wire segments are illustrated in Fig. 2, and one feasible assignment for those wires are shown in Fig. 3.

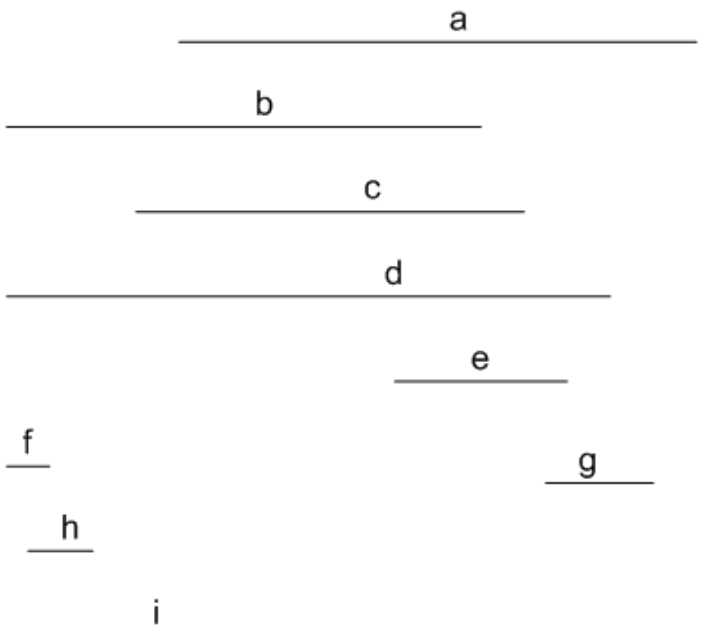

Fig. 2. Original positions of 9 wire segments which are to be assigned.

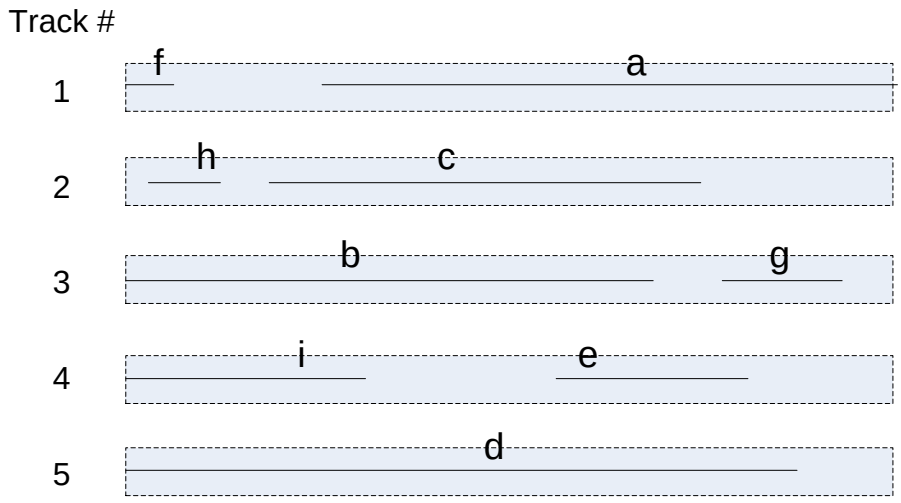

Fig. 3. A feasible track assignment for wires given in Fig. 2.

1.3 Crosstalk

If we only consider wire conflict, there are off-the-shelf algorithms that can be applied to solving the track assignment problem, for example the left-edge algorithm [2]. However, in real circuits there are many other constraints that need to be taken care of, and one important constraint is delay increment. Delay increment is mainly induced by crosstalk noise between parallel wires adjacent with each other. Crosstalk is usually estimated using coupling capacitance [3], which appears when two wires carrying signals are parallel and near to each other. The exact value of coupling capacitance depends on the coupling length, the distance between the two wires and the switching factor [4]. The expression can be written as follows [5], [6], [7]:

$$CC(i,j) = \alpha \cdot f_{ij} \cdot \frac{l_{ij}}{d_{ij}^{\beta}} \tag{1}$$

where i and j are two adjacent wire segments, α and β are technology dependent constants, l_{ij} is the coupling length, d_{ij} is the wire spacing between i and j, and f_{ij} is the switching factor for i and j.

Since we only consider crosstalk between wires in neighboring tracks, and the switching factor can be a constant, generally the crosstalk value increases proportionately with the coupling length. Fig.4 gives an illustration on this capacitive coupling.

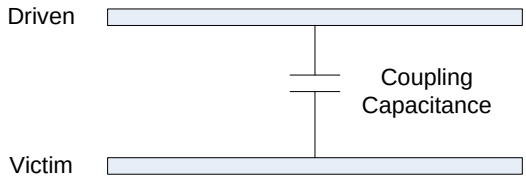

Fig. 4. Coupling capacitance between two adjacent wires.

1.4 Delay Increment

Crosstalk noise can have a great negative impact on wire delay. In this project we use the Elmore delay model for delay increment analysis. A simple example illustrating this model is given in Fig. 5, where two wire segments of different nets are assigned to adjacent tracks. Suppose the coupling capacitance between wire *a* and *b* is *CC*. According to the Elmore delay model, the delay increment at sink *a2* induced by wire *b* is $d_{ab} = R_{a0,a1} \cdot CC + R_{a1,a2} \cdot \frac{CC}{2}$, where $R_{a0,a1}$ is the resistance between *a0* and *a1*, and $R_{a1,a2}$ is the resistance between *a1* and *a2*.

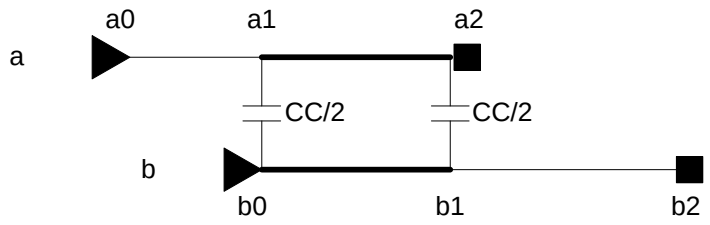

Fig. 5. π-model for delay increment induced by coupling capacitance.

In real circuits, the amount of extra delay caused by crosstalk should be limited to a bearable range. In the problem formulated in this project, we set a bound of delay increment for each wire segment, and our task is to do track assignment for wire segments such that there is no wire conflict and all the bounds are satisfied.

1.5 Related Work

There are prior works on crosstalk-aware routing resource assignment problems. An ILP formulation is proposed in [8], where decision variables are stored in a 0-1 matrix indicating the track assignment for each wire. A redundant variable matrix indicating wire adjacency information is introduced to build the expressions of the crosstalk constraints. However, those redundant variables greatly increase the complexity of the problem, and therefore limit the applicability of this formulation. In [9], an ILP formulation based on wire packing graph (WPG) is built. A maximal clique is found in the conflict graph for all wires, and a WPG is constructed with each vertex indicating a possible wire-track combination. There are no redundant variables, but the size of the graph can be very big because the number of vertices is the product of the number of wires and the number of tracks. Battery [1] proposed a graph model, weighted bipartite

matching, for track assignment problems. In this model, wires are represented by vertices on one side of the Bipartite graph, and tracks are on the other side. A minimum weight matching solution is therefore a feasible assignment. However this model is not suitable for problems with dynamically changing weight. In Wu's work [5] a method based on Hamiltonian path is proposed. Considering both coupling capacitance induced delay and the detour induced delay, they formulate the track assignment problem as a Sequential Ordering Problem (SOP) and solve it using an SOP solver. Their objective is to maximize the minimum slack, which is different from our constraint-based problem. Another work [10] on crosstalk-aware track assignment introduces a method based on the linear assignment algorithm. A cost matrix is built to store the estimated cost on every possible assignment for each wire segment. The linear assignment algorithm is applied repeatedly to assign the maximum clique found in the interval graph during each step. The algorithm does not take into account the dynamically changing accumulated crosstalk. Xue's work [11] considers bounds on crosstalk, and they partitioned the bound of every net according to the routing regions that the net passes through. Inside each region, crosstalk is evaluated in terms of risk. They tune the bounds in different regions of the same net in order to maximize the chance to find a risk-free assignment. However, in our project the bound of each net segment is treated as a whole.

2. PROBLEM FORMULATION

The track assignment problem, as well as many other CAD problems, can be formulated using an Integer Linear Programming (ILP) model. Although there is no specific objective to maximize or minimize in our track assignment problem, we do have different types of constraints. Further in this section we will show that the problem can actually be formulated as a 0-1 ILP problem.

We make such assumptions in this problem formulation:

i). Wires in one layer are all in same direction (horizontal or vertical); There is no crosstalk between layers;

ii). Only consider crosstalk between wires in adjacent tracks on the same layer.

With the assumptions above, the **track assignment considering cross-talk induced performance degradation problem** can be formulated as follows.

Given a global routing result which contains the initial positions of a set of wire segments $\boldsymbol{W}$*, a vector of delay increment bounds* $\boldsymbol{B}$*, a matrix* $\boldsymbol{D}$ *of delay increment between each pair of potentially adjacent wire segments, and a set of available tracks* $\boldsymbol{M}$*, find a feasible solution such that each wire segment* $\boldsymbol{i} \in \boldsymbol{W}$ *is assigned to a track* $\boldsymbol{k} \in \boldsymbol{M}$ *without causing any overlap, and in the meantime no delay increment bound is violated.*

The main computing complexity of this track assignment problem lies in the delay increment calculation, because the delay increment of every wire segment is changing dynamically with the modification of each wire position. Each time a wire segment is

assigned, its delay increment caused by all of its neighbor wires should be added. At the same time, the delay increment of every neighbor wire caused by the newly assigned wire should be updated as well. If any bound violation occurs, we drop the solution immediately. To make it convenient for calculation, we do track assignment for one row of tiles at a time, and repeat the same procedure for the rest of the rows.

Another important factor to be considered is that the delay increment matrix ***D*** is not symmetric. For any pair of adjacent wires that have overlap in span, the delay increment caused by each other can be different because they may have different upstream resistance. This is illustrated in Fig. 6.

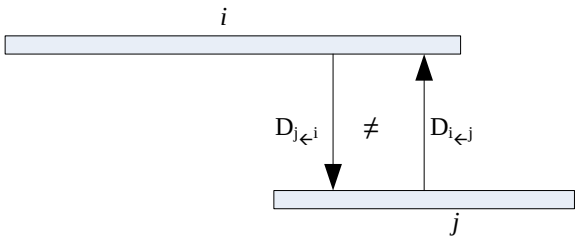

Fig. 6. Delay increment is not symmetric.

2.1 Types of Constraints

From analysis above, we deduce three types of constraints that are required in our problem formulation:

i). Every wire segment should be assigned to an available track, and no wire segment can occupy more than one track;

ii). Wire segments assigned to the same track should not have overlap in span, i.e. they should not conflict with each other;

iii). For each wire segment, the sum of delay increment caused by its neighbor wires should not exceed the given bound.

Next, we introduce the following matrices to facilitate later detailed discussion on these constraints used by the ILP model. The decision variables are constructed as a 0-1 matrix **T** with:

$$t_{ik} = \begin{cases} 1, & if\ wire\ i\ is\ assigned\ to\ track\ k \\ 0, & otherwise \end{cases} \quad (2)$$

where $i = 1, \ldots, m$ with m being the total number of wire segments, and $k = 1, \ldots, n$ with n being the total number of available tracks. The matrix $\boldsymbol{T}$ has m rows and n columns, so the total number of decision variables is $m \times n$. Since all variables are binary, the problem becomes a 0-1 ILP problem.

We also construct some constant matrices. The first one is the 0-1 matrix $\boldsymbol{O}$, which stores the overlap information among wire segments:

$$o_{ij} = \begin{cases} 1, & if\ wire\ i\ and\ wire\ j\ have\ overlap\ in\ span \\ 0, & otherwise \end{cases} \quad (3)$$

where $i, j = 1, \ldots, m$. The values of elements in matrix $\boldsymbol{O}$ can be obtained from the given positions of wire segments. For the convenience of calculation, we set $o_{ii} = 0$.

Another constant matrix is the delay increment matrix $\boldsymbol{D}$, whose elements represent the potential delay increment between pairs of wires if they are adjacent:

$$d_{ij} = delay\ increment\ of\ wire\ i\ caused\ by\ its\ neighbor\ wire\ j \quad (4)$$

Note that matrix $\boldsymbol{D}$ is not symmetric, as demonstrated earlier in this section. $d_{ii} = 0$ because a wire segment does not have coupling capacitance with itself. The values of elements in matrix D can be calculated using (1) and the Elmore delay model.

We also assume a given constant matrix, or vector, $\boldsymbol{B} = (B_1, \ldots, B_m)$ where B_i is the delay increment bound of wire *i*.

The three types of constraints are constructed based on the elements of those matrices. Constraints of type i) can be expressed as follows:

$$\sum_{k=1}^{n} t_{ik} = 1 \tag{5}$$

where $i = 1, \ldots, m$. These constraints ensure that each wire segment is assigned to one and only one track.

Type ii) constraints guarantee that there is no wire conflict, and is in the form of:

$$t_{ik} + t_{jk} \leq 2 - o_{ij} \tag{6}$$

where $i = 1, \ldots, m$ and $k = 1, \ldots, n$. The o_{ij} is an element of matrix *O*, as described in formula (3). If two wires *i* and *j* are both assigned to track *k* and they have overlap in span, i.e., $o_{ij} = 1$, this inequality will be violated with *LHS* = 2 and *RHS* = 1. Otherwise the inequality holds because the *LHS* is at most 1 and the *RHS* is at least 1.

The last type of constraints is the most complex. In order to prevent the delay increment of any wire from exceeding its bound, we must consider all kinds of possible adjacencies. Redundant decision variables could be created to represent the adjacency information, as described in [8], when the circuit scale is not very large. However, real circuits often have millions of wires. In such cases redundant variables will greatly increase the complexity of the ILP problem and significantly slow down the solving procedure. To improve scalability we explored the structure of the decision matrix $\boldsymbol{T}$ and propose a new way to represent these constraints using only the decision variables in $\boldsymbol{T}$.

For each t_{ik}, we look at its adjacent two columns $t_{*,(k-1)}$ and $t_{*,(k+1)}$, as illustrated in Fig. 7. Among the three numbers t_{ik}, $t_{j,(k-1)}$ and $t_{j,(k+1)}$, at most two of them can be '1'. That is to say,

$$t_{ik} + t_{j,(k-1)} + t_{j,(k+1)} - 1 \leq 1 \tag{7}$$

where $i, j = 1, \dots, m$ and $k = 1, \dots, n$; and we define that $t_{j,(-1)}=0$ and $t_{j,(n+1)}=0$. Equality holds only when wire i and wire k are adjacent. This expression contains all possible adjacencies for every possible assignment of each wire. Based on them we can derive the final expressions of constraints considering delay increment bound as follows:

$$\sum_j (t_{ik} + t_{j,(k-1)} + t_{j,(k+1)} - 1) \times d_{ij} \leq B_i \tag{8}$$

For each wire i, it has n constraints ($k = 1, \dots, n$) with the above form. Only one of them, however, is effective, which is when $t_{ik} = 1$ i.e. wire i is assigned to track k. For all other cases when $t_{ik} = 0$, the value of the *LHS* of the inequality would be no more than zero, thus would not contribute to bound violation. This expression guarantees that the delay increment bound for each wire segment is satisfied, since it provides a traversal of all possible adjacencies of all pairs of wire segments.

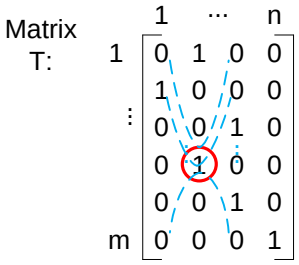

Fig. 7. Explore the structure of the decision variable matrix ***T***. The sum of any three numbers connected by a blue dashed line is no more than two.

2.2 Solving Integer Linear Programming Problem

With all three types of constraints constructed, our track assignment problem is completely formulated as a constraint based 0-1 ILP problem. We built and solved this ILP model using a commonly known ILP solver CPLEX. A toy case was used to test the correctness of this model. The wire positions in this case are the same as shown in Fig. 2, and the solver generated solution as in Fig. 3. The delay increment values were set according to the coupling length, and the bounds were set randomly. There is no bound violation in this solution. However, in certain cases the bounds may have been set to tight for the solver to find a feasible solution, i.e., the problem becomes unsatisfiable.

Actually this 0-1 ILP problem can be transformed to Satisfiability (SAT) problem in polynomial time by the NP-complete theory [12], [13], and there is existing algorithm to do this transform [14]. To compare results, we also tested the same toy case using a public domain SAT solver PBS [15]. The solution given by the SAT solver was not exactly the same as the solution given by CPLEX, but was quite similar. This is true since there could be multiple feasible solutions. Both solvers seem to work well on test cases with small number of wires and tracks.

TABLE I

RESULTS GIVEN BY CPLEX OF SEVERAL TEST CASES

num_wire	num_track	runtime	memory	num_var
9	5	15"'	0.77M	81
50	50	4"15"'	148M	3725
100	100	45"92"'	1.18G	14950

Table 1 shows the runtime and memory consumed by CPLEX for various problem sizes. As can be observed, the exponential relationship implies a critical limitation of ILP solvers as well as SAT solvers. ILP solvers are a little better, but still quite slow when the number of wires reaches the hundreds level. This is not a feasible method for real world circuits, which typically have millions of wire segments and far more exceed the capability of current ILP solvers. Therefore, in the next section, a novel heuristic algorithm with significantly lower time and space complexity is developed to make the track assignment task applicable to real-world complex cases.

3. ALGORITHM

The heuristic algorithm is intended to solve the track assignment problem more efficiently. From the previous section we see that the ILP model works for small cases, but when it comes to real circuit the performance of the ILP solver greatly degrades because of the increasing number of variables and constraints. In our heuristic method, we make use of graph theory to formulate graph models for real circuit, such as wire conflict graph and adjacency graph.

The initial positions of wire segments are given by a global router. At this step, wires are grouped into panels which are horizontal rows of grids. Wire segments in the same panel are packed together vertically, so they are highly overlapping. The routing resource inside each panel is a fixed number of available tracks. The task is to assign these wire segments within the same panel to different tracks such that no wire conflict occurs and all bounds on performance degradation are satisfied.

3.1 Graph Construction

3.1.1 Conflict Graph (Interval Graph)

A conflict graph G(V,E) can be created for the wire segments. Each wire segment is represented by a vertex $v \in V$, and an edge (u, v) between vertex u and v means that wire u and wire v have overlap in span. This is illustrated in Fig. 8 below.

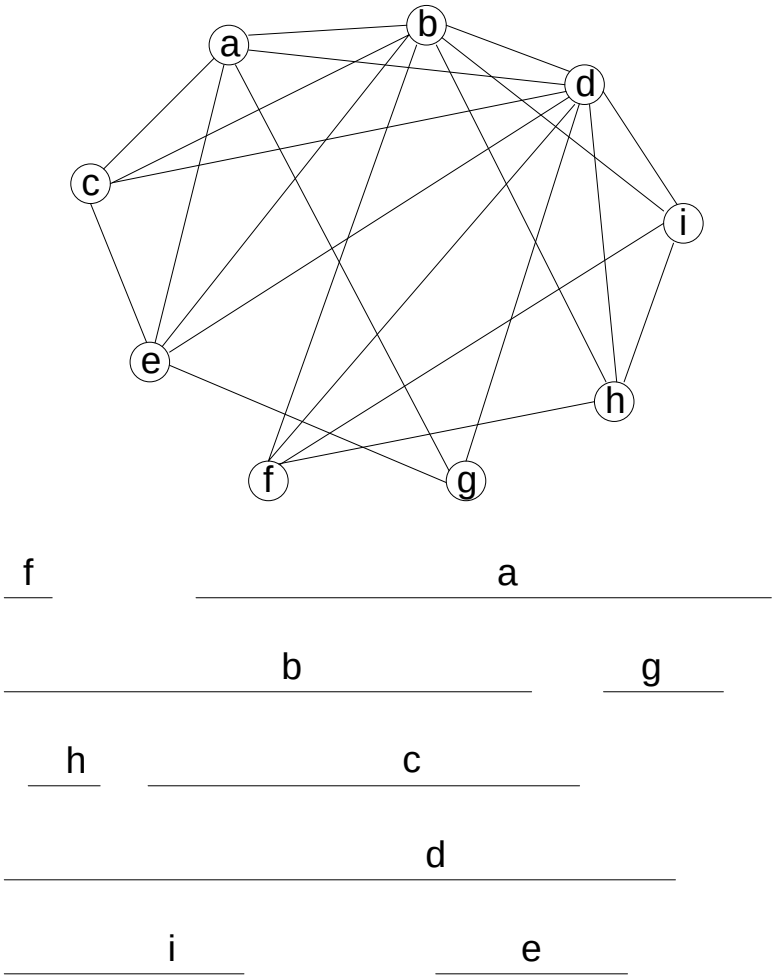

Fig. 8. Conflict graph for 9 wires.

3.1.2 Clique

A clique [16] in an undirected graph is a sub-graph which is complete. There is an edge connecting every pair of vertices in the clique. The maximum clique of a graph is a clique with the largest size (the most vertices). Fig. 9 shows a maximum clique in the conflict graph given in Fig. 8. In our case, the vertices in a clique are wires that have overlap in span with each other. No pairs of them can share the same track. Thus the size

of the maximum clique actually represents the minimum number of tracks needed in order to assign all wires in an overlapping-free manner.

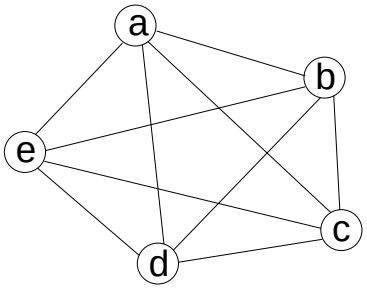

Fig. 9. A maximum clique inside the conflict graph in Fig. 8. Wire *a*, *b*, *c*, *d* and *e* have overlap in span with each other, so they cannot be assigned to a same track. To assign the 9 wires without wire conflict, a minimum number of 5 tracks are needed.

3.1.3 Adjacency Graph

To add crosstalk information into the graph, we introduce a new type of conflict graph, i.e., adjacency graph. This graph is developed based on the clique in the original conflict graph, but with weight for all the edges. We know that wires in a clique have overlap in span and must be assigned to different tracks, so any pairs of those wires are potential neighbors. That is to say, they could be assigned to adjacent tracks and there could be crosstalk between them. To limit the delay degradation caused by crosstalk, we should take control of the potential adjacencies and forbid those that can cause too much delay increment.

We set a weighting factor w_{ij} for edge $e(i, j)$ connecting vertex i and j. The value of w_{ij} is the sum of delay increment of the two wires caused by each other when they are assigned to adjacent parallel tracks:

$$w_{ij} = d_{ij} + d_{ji} \quad (9)$$

where d_{ij} is the delay increment of wire i caused by its neighbor wire j, and d_{ji} is the delay increment of wire j caused by wire i.

After assigning the weighting factor to edges in the clique, we need to check if there are any forbidden adjacencies. For each edge *e(i, j)*, we check if $d_{ij} \leq B_i$ and $d_{ji} \leq B_j$, and we remove edge *e(i, j)* if any of the two inequalities is violated. Because each wire in this clique can have two neighbors (neighbor in track above or below) at the same time, we also consider the case that the sum of delay increment caused by the two neighbors exceeds the bound. The details of this method are discussed in section 3.2. The graph in Fig. 9 is reduced to the one shown in Fig. 10 by the edge removing step. Note that if there are more available tracks than what is needed, i.e., the size of the maximum clique, we can add dummy vertices into the reduced graph. Each dummy vertex represents a spare track, i.e., a virtual wire and it connects to all the other vertices via zero weighted edges in the graph because an empty track can be neighbor of any wire with no crosstalk.

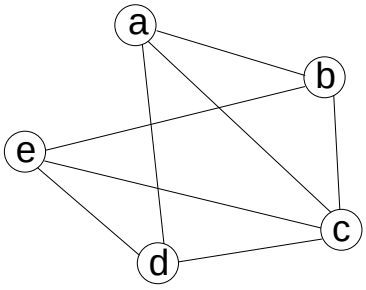

Fig. 10. The remaining graph after removing edges from the clique in Fig. 9.

Next heuristic is to find a Hamiltonian path inside the reduced graph, which gives the order of tracks that the wires are assigned to. For example in Fig. 10, a possible Hamiltonian path is a→b→e→c→d, which corresponds to the track assignment that wire *a* is assigned to track 1, wire *b* is assigned to track 2, etc. When finding the Hamiltonian path we also need to consider the weights of edges, since we want to find a minimum weight path to limit the total crosstalk, and leave a less noisy environment for all the unassigned wires. In our case there is no specific requirement on deciding the beginning and ending point of the path, so we can make use of the algorithms for Traveling Salesman Problem (TSP) [17] to find a circle, and cut one edge to form a path.

3.1.4 Extended Bipartite Graph

After assigning the wires corresponding to the vertices of the maximum clique, we apply a greedy algorithm to assign the rest of the wire segments. Note that the "Hamiltonian path" was not applied to assigning the rest of wires, because some tracks are already occupied, and when considering delay increment we need to take into account both assigned and unassigned wire segments. That is to say, delay increment of each wire is changing dynamically with different choices of assignment. We repeat the step of finding maximum clique among the rest of wires until all wires are assigned, and construct an extended bipartite graph for each one of the cliques.

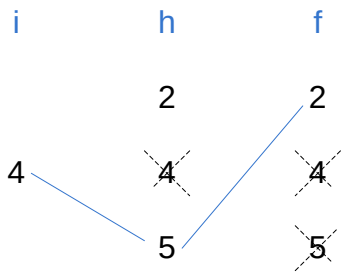

Fig. 11. Assigning a clique of wire segments: *i, h* and *f*. Starting from the one with the least number of track candidates, find a legal path from left to right without causing any bound violation.

For example in Fig. 11, there are three wire segments *i, h, f* left to be assigned. The numbers listed under each wire segment are the indices of tracks that are available for this particular segment. The choices of tracks are decided by the current assigned wire segments. The rule is that newly assigned wire segments should not conflict with previously assigned wires, and the new assignment should not cause any bound violations. In the case shown in Fig. 11, wire *i* has only one track candidate, i.e., it can only be assigned to track 4, and wire *h* and *f* both have three available track options. If any one of *h* and *f* is assigned first and occupies track 4, wire *i* would have no valid candidate tracks. A similar condition is discussed in [1], but instead of taking their look-ahead heuristic, we do a sorting for all the wire segments before assigning them. We sort the wire segments according to their number of candidate tracks. Segments with less track options are given higher priority such that less flexible variable is fixed first, resulting in a greater overall flexibility among the whole system of variables. So in Fig. 11, wire *i* is given highest priority since it has only one choice of tracks. After putting *i* in track 4, we immediately update the track options for wire *h* which will be assigned next. Wire *h* cannot be assigned to track 4 anymore because wire *i* has taken the place.

So we remove track 4 from its track candidates. Then we randomly choose from track 2 and 5 for wire *h* and check whether it is a legal assignment. If assigning wire *h* to track 5 does not cause any bound violations we will continue the same procedure to assign the next wire segment. If such an assignment causes bound violation, we remove this track candidate and try its fellow candidates. In the case that all track candidates are visited but no one can satisfy the bound restriction, we go back one step to reassign the previous wire segment to another legal track candidate. The current amount of delay increment of both previous wire segment and the segment to be assigned should be updated, as well as the occupation information of track candidates.

The path from the left end to the right end represents the order of tracks that the corresponding wire segments are assigned to. When a path is found, the assignment of the current clique is done. All these wires will be marked as assigned and there will be a new record for each track storing the name of wires that occupy this track. We can continue to find the next maximum clique in the set of unassigned wires, and apply the same procedure of assignment until all wires are properly assigned.

3.2 Flowchart and Detailed Algorithms

This heuristic algorithm solves track assignment problem for wires in a single layer. It is targets at a whole panel at a time, and repeats until all panels are processed. The whole procedure can be divided into two steps: assigning the largest clique and assigning the rest of wires. The first step is transformed into a minimum weight Hamiltonian path

problem, and the second step is implemented in a greedy way as interpreted in the previous section. The flow chart below shows the outline of this algorithm.

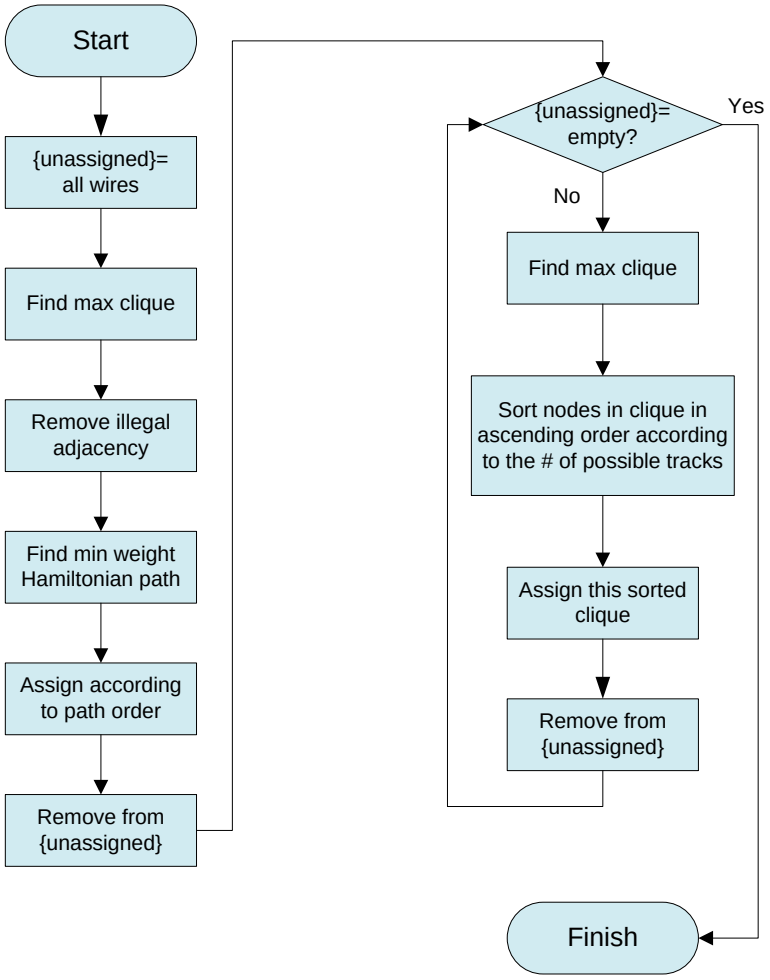

Fig. 12. Flow chart of the heuristic algorithm.

3.2.1 Step 1: Assign the Maximum Clique

The left half of the flow chart shown in Fig. 12 describes the first step of the algorithm. We define an array *unassigned* to store the indices of wire segments that have not yet been assigned. Initially it contains all the wire segments that are generated by

global routing. Once a segment is fixed, its index is removed from *unassigned*. The algorithm comes to an end when the array *unassigned* is empty.

Find Maximum Clique. The information of wire position is obtained from a global router, the coalesCgrip [18]. From its output we can read in the initial position of all wire segments for each panel *p*, and calculate the potential delay increment matrix *D*. The overlap matrix *O* is also created, and a conflict graph CG(*V, E*) is built where each vertex $v_i \in V$ corresponds to a wire segment in panel *p*. There is an edge connecting vertex v_i and v_j if wire *i* and wire *j* overlap in span.

The target is to find the maximum clique in this conflict graph, and the first step is to find all cliques. An off-the-shelf algorithm '*segment tree*' [19] is used to find all cliques. Each wire is seen as an interval, with a head node (left end) and a rear node (right end). Considering all wires located in a panel, sort the corresponding nodes according to their horizontal coordinates. Then the set of all cliques can be obtained by sweeping through all the nodes from left to right. For each node, if it is a head node then its corresponding wire is added to the current clique; otherwise the wire is removed from the current clique.

Fig. 13 illustrates this procedure. Suppose there are five wires numbered from 1 to 5, and their head and rear nodes are sorted and indexed from 0 to 9. The blue dashed line sweeps from the left to the right, adding or removing wires from cliques. Each time it comes across a node, a new clique is created. In this particular case the cliques found are: {1}, {1,3}, {1,2,3}, {1,2,3,5}, {1,3,5}, {1,3,4,5}, {1,4,5}, {1,4} and {4}. Thus the cliques with maximum size are {1,2,3,5} and {1,3,4,5}.

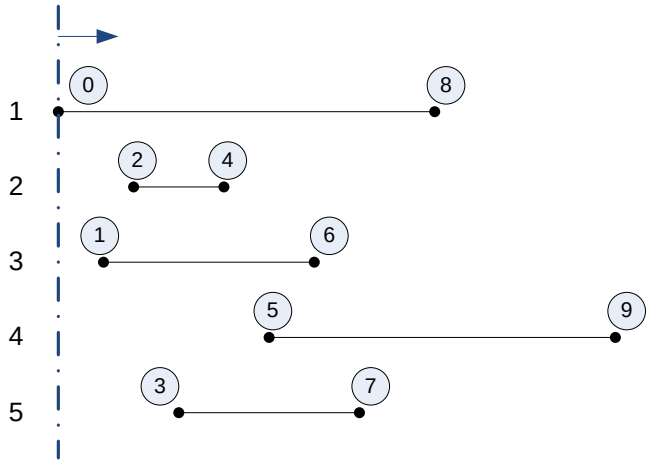

Fig. 13. Illustration of the *segment tree* method.

When all cliques are found, they are sorted in descending order of their sizes. If several cliques are about the same size, the one with biggest total wire length comes first. We choose to assign longer wires first because longer wires have relatively less chances to be assigned legally if not given priority.

Having the maximum clique chosen, we convert it to an adjacency graph by adding weighting factor into the clique, as described in section 3.1.3.

Remove Illegal Adjacency. The max clique is a complete graph, implying that no pair of wires in this clique can be assigned to the same track. It also implies the possibility for any pair of wires to be assigned to neighboring tracks. However, some of these adjacencies are illegal since they might cause violations of the delay increment bounds of certain wires. Two kinds of illegal adjacencies are considered in this algorithm, and an edge removing approach is performed accordingly.

i) for each vertex v_i check all edges e(v_i, v_j) connecting to it:

if $d_{ij} > B_i$, remove edge e(v_i, v_j), where B_i is the delay increment bound of wire i.

ii) for each vertex v_i check every pair of edges e(v_i, v_j) and e(v_i, v_k):

if $d_{ij} + d_{ik} > B_i$, remove one of these two edges.

In the actual implementation, violations of type i) are processed prior to those of type ii), because they are 'hard' violations. After removing all violations of type i), vertices which still have violations of type ii) are added into a set *S*. Removal of type ii) illegal adjacencies is done inside set *S*.

When handling the second type of illegal adjacency, generally the edge with larger weight is removed. For example, if vertex v_i is in set *S* and $w_{ik} > w_{ij}$ then edge e(v_i, v_k) is chosen to be removed. However, if vertex v_j is also an element of *S* while vertex v_k is not, then edge e(v_i, v_j) is removed with priority regardless of its weight.

Each time an edge e(v_i, v_j) is removed, we check vertices v_i and v_j to see if they still have type ii) bound violation. If any one of them no longer has bound violation, it is removed from set *S*. This procedure is repeated until *S* is empty.

Find Minimum Weight Hamiltonian Path. The problem is now to find a minimum weight Hamiltonian path in the edge removed graph. If the number of tracks is greater than the size of max clique, spare tracks are added to the graph as new vertices with zero weight edges connecting to all existing vertices. Since there are no specified starting and ending points, the problem is actually equivalent to the Traveling Salesman Problem. An existing heuristic *nearest insertion* [20] is applied here. It finds and returns a circle if there is any, thus a path can be formed by breaking any one of the edges. An example of a Hamiltonian path (colored in blue) is given in Fig. 14.

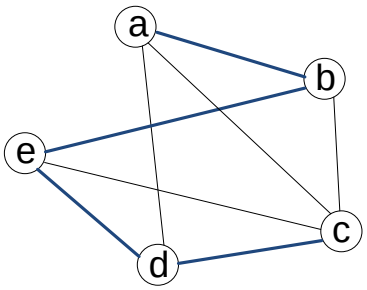

Fig. 14. An example of Hamiltonian path a-b-e-d-c.

In the test benches (IBM superblue 4 for example), the maximum clique size is 80 which is not too big for this heuristic to give a result in reasonable time. The wires in the max clique are then assigned to tracks according to the order of the vertices along the path.

3.2.2 Step 2: Assign the Rest of Wires

The procedure of step 1 cannot be reused for step 2, because most of the tracks are already occupied and the delay increment between wires changes dynamically when assigning the rest of the wires. Nevertheless, part of step 1 is still applicable for step 2.

Firstly the interval tree heuristic is performed to find the max clique among all unassigned wires. For each wire in the max clique, we count the number of candidate tracks (tracks that this wire can be assigned to without causing overlap), and then sort these wires in an ascending order according to this number. That is to say, wires with less track options are considered first.

Track assignment is then performed starting from the very beginning of the sorted clique, until the whole clique is assigned. For a particular wire, a candidate track is selected randomly from the set of candidate tracks belonging to this wire, and this track

is marked as *tried*. We try to assign the wire to this track, and check if this assignment is legal (if it will cause any bound violations). This attempt is repeated until a legal assignment is found. Then the track is marked as *occupied* so that other wires in the same clique cannot be assigned to the same track later. If all candidates are *tried* but none of the attempts is legal, we trace one step back and reassign the previous wire to a different track.

Same procedure is performed for the max clique found in each step, until all the wires in panel p are assigned. A whole circuit may contain hundreds of panels, and all of those panels are independent in our case. The algorithm can be reused for each one of the panels, and it comes to an end when all panels are done.

Theoretically, if the problem case has a feasible solution (all wire segments can be assigned without any bound violation), the heuristic algorithm should be able to find it. However, in real cases it is hard to create a problem which has a feasible solution for sure. This is because there are often millions of wire segments on a single circuit, and it is impossible to set proper bounds for all of the wires. In our project we set the bounds randomly in a certain range. So there are situations when not all wire segments can be legally assigned. To deal with those situations, our algorithm can also accept tolerable violations when there is no way to avoid them.

3.2.3 Delay Calculation and Bound Setting

The method of calculating delay increment caused by crosstalk is given in previous chapter, and here we describe the calculation in detail. For any wire segment *i*, the relative position between *i* and its neighbor *j* can be any case shown in Fig. 15. Suppose

the overlapping length is l_o, the upstream length from the source point of *i* to the overlap starting point is l_{up}, and accordingly the downstream length after the overlapping part is l_{down}. In Fig. 15 (A) the neighbor wire *j* is much longer than wire *i* and the overlapping length is the length of *i*. Thus the delay increment of *i* caused by *j* can be written as

$$d_{ij} = r \cdot l_o \cdot \frac{c \cdot l_o}{2} \tag{10}$$

where *r* is the unit length resistance and *c* is the unit length coupling capacitance. In case (B) the overlapping length is only part of the wire length. The delay increment is expressed as

$$d_{ij} = r \cdot l_{up} \cdot c \cdot l_o + r \cdot l_o \cdot \frac{c \cdot l_o}{2}. \tag{11}$$

In case (C) the expression is the same as (11) because the downstream length has no effect on delay increment. Similarly the expression in case (D) is the same as in case (A) regardless of the downstream length.

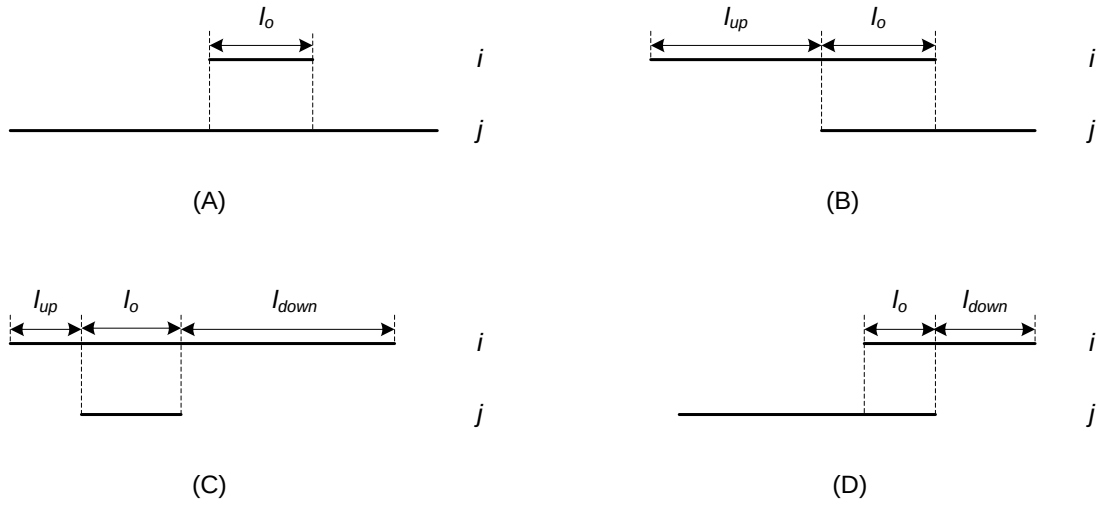

Fig. 15. Cases of relative positions of two neighboring wires.

Note that formula (10) is a special case of (11) with $l_{up}= 0$. Therefore formula (11) is a general expression of the delay increment in all cases. Since the horizontal coordinates of all wire segments are known, we can easily calculate l_{up} and l_o, and thereafter get the delay increment matrix D.

Another fact is that the delay increment reaches the maximum when the entire wire overlaps with a neighbor wire, i.e., $l_o = l_i$. This can be proven by transforming formula (11) into

$$\mathrm{d_{ij}} = \mathrm{r} \cdot \mathrm{c} \cdot \mathrm{l_o} \cdot (\mathrm{l_i} - \frac{\mathrm{l_o}}{2} - \mathrm{l_{down}}) \tag{12}$$

by replacing l_{up} with $l_i - l_{down}$. It is obvious that d_{ij} can reach the maximum only when $l_{down} = 0$. The formula then becomes

$$\mathrm{d_{ij}} = \mathrm{r} \cdot \mathrm{c} \cdot \mathrm{l_o} \cdot (\mathrm{l_i} - \frac{\mathrm{l_o}}{2}) \tag{13}$$

This is a concave function with curve shown in Fig. 16. The function value reaches the maximum $d_{ij} = \frac{rcl_i^2}{2}$ when and only when $l_o = l_i$.

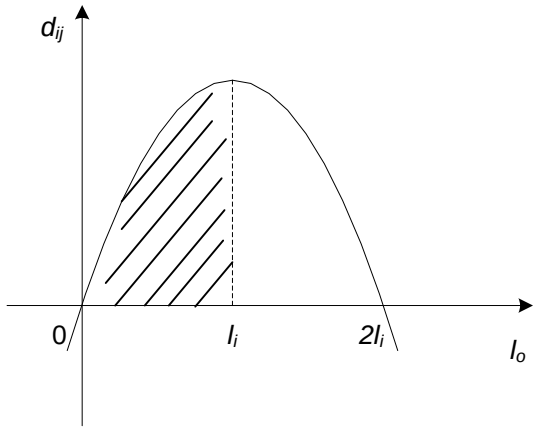

Fig. 16. Function curve of d_{ij}. Feasible domain is $0 \le l_o \le l_i$, where d_{ij} is monotonically increasing. Maximum value is reached when $l_o = l_i$.

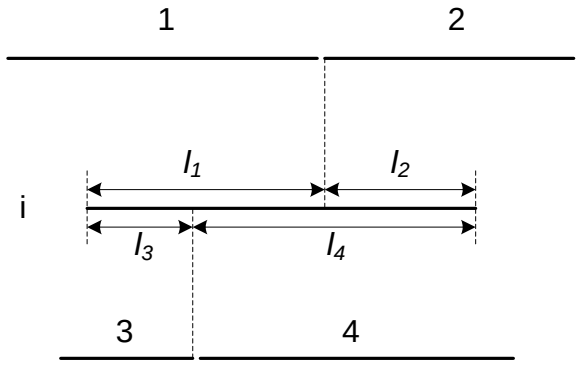

Fig. 17. An example of a fully overlapped wire segment.

To set proper bounds for each wire segment, we need to know the maximum total delay increment a wire segment can have. We know that for a certain wire *i*, its delay increment is $d_i = \sum_j d_{ij}$, where *j* represents any neighbor wire in the track above or below wire *i*. We also know from above analysis that delay increment is maximal when the entire length is overlapped. We take the case shown in Fig. 17 as an example. Suppose wire segment *i* is surrounded by four neighboring wires 1, 2, 3 and 4. The corresponding overlapping lengths are l_1, l_2, l_3 and l_4. The total delay increment of wire segment *i* can be written as

$$d_i = \frac{rcl_1^2}{2} + rcl_1 l_2 + \frac{rcl_2^2}{2} + \frac{rcl_3^2}{2} + rcl_3 l_4 + \frac{rcl_4^2}{2}$$

$$= \frac{rc}{2}(l_1 + l_2)^2 + \frac{rc}{2}(l_3 + l_4)^2$$

$$= \frac{rc}{2} \cdot 2{l_i}^2 = rcl_i^2 \qquad (14)$$

where l_i is the length of wire *i*. Formula (14) gives an upper bound of the delay increment a wire can have: $d_{max} = rcl^2$. According to this we set the delay increment

bound for every wire segment to be a random number in the range $(0, d_{max})$. However, it is meaningless to have a bound which is too small, because in this case there would be no feasible solution to the track assignment problem. So we should set reasonable bounds for our test benches. To make it convenient for comparison, we set three cases of ranges: $(0.25d_{max}, d_{max})$, $(0.5d_{max}, d_{max})$ and $(0.75d_{max}, d_{max})$. We will compare the results of these cases in the next section.

4. EXPERIMENTAL RESULTS

We carried out several experiments to evaluate the efficiency of our algorithm. Our test cases are from the IBM ISPD 2011 contest. The test benches we used are circuits Superblue1, Superblue 4 and Superblue 10. We employ global router coalesCgrip and feed the original test bench to the router. The router then generates a global routing output containing the initial horizontal positions of wires. In this output file wires are grouped into panels (about 500 to 900 panels for a particular test bench), where inside each panel wires are given the same vertical coordinate. That is to say, initially all the wires in one panel are overlapping with each other. Each panel has around 700 to 3000 wires, and panels are independent with each other. Thus we can do track assignment panel by panel, dealing with thousands of wires at a time instead of millions as a whole.

We parse the output file of the global router and create an input file for our program. The max clique size in our test benches is 80, and we set the number of available tracks to be 85, a little bigger than the max size. This is to leave some space for multiple feasible solutions, and at the same time do not make the problem too easy to solve. Parameters on resistance and capacitance as well as delay increment bounds are set according to section 3.2.3. Results of different test benches are compared according to the number of wires having bound violations, and the amount of violations.

Fig. 18 shows the results of our algorithm for circuit Superblue4, which contains 523,388 wire segments. Bars in different colors represent different bound ranges as described in section 3.2.3. Fig. 19 shows the results for circuit Superblue1, which

contains 722,904 wire segments. Fig. 20 is for circuit Superblue10, containing 928,278 wire segments. From these three figures we can see that our algorithm performs very well in controlling bound violations. It can achieve zero violation when the bounds are not too tight. Even if bounds are tight, it can generate a solution with very low bound violation rate.

Fig. 21 gives an inside look of a part of a certain panel after track assignment. It shows the positions of real wire segments, and we can see clearly that there is no wire conflict. This figure is generated by MATLAB by reading the output file of our algorithm.

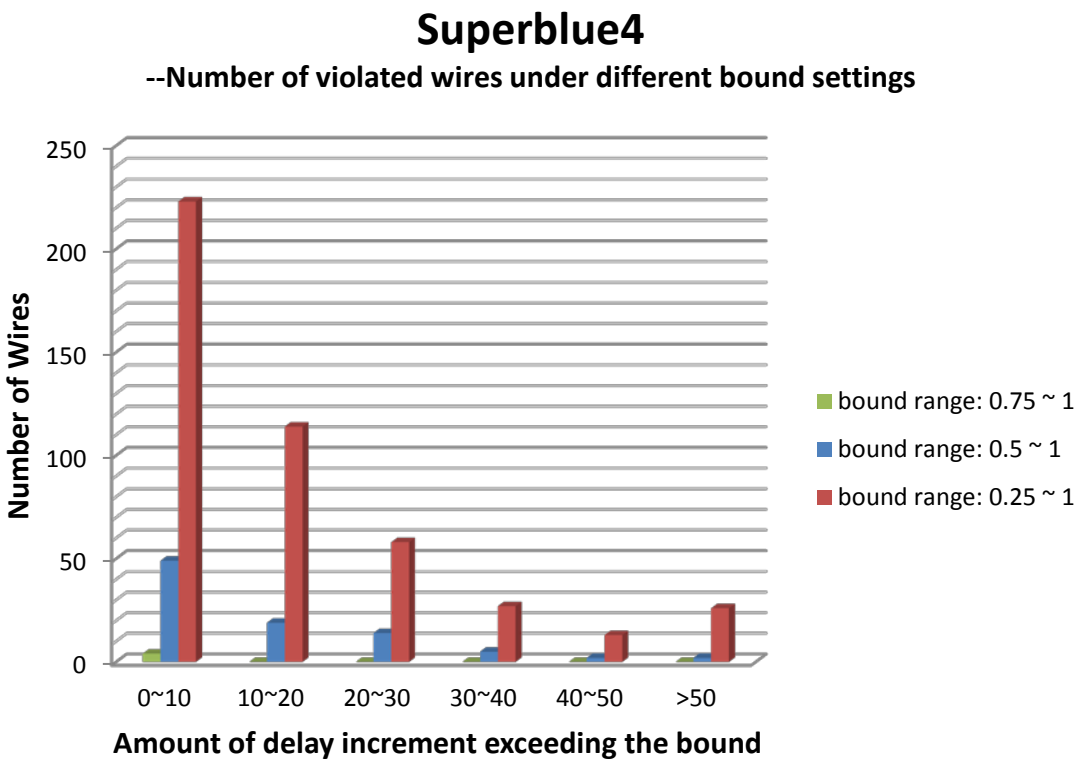

Fig. 18. Number of violated wires in test case Superblue4.

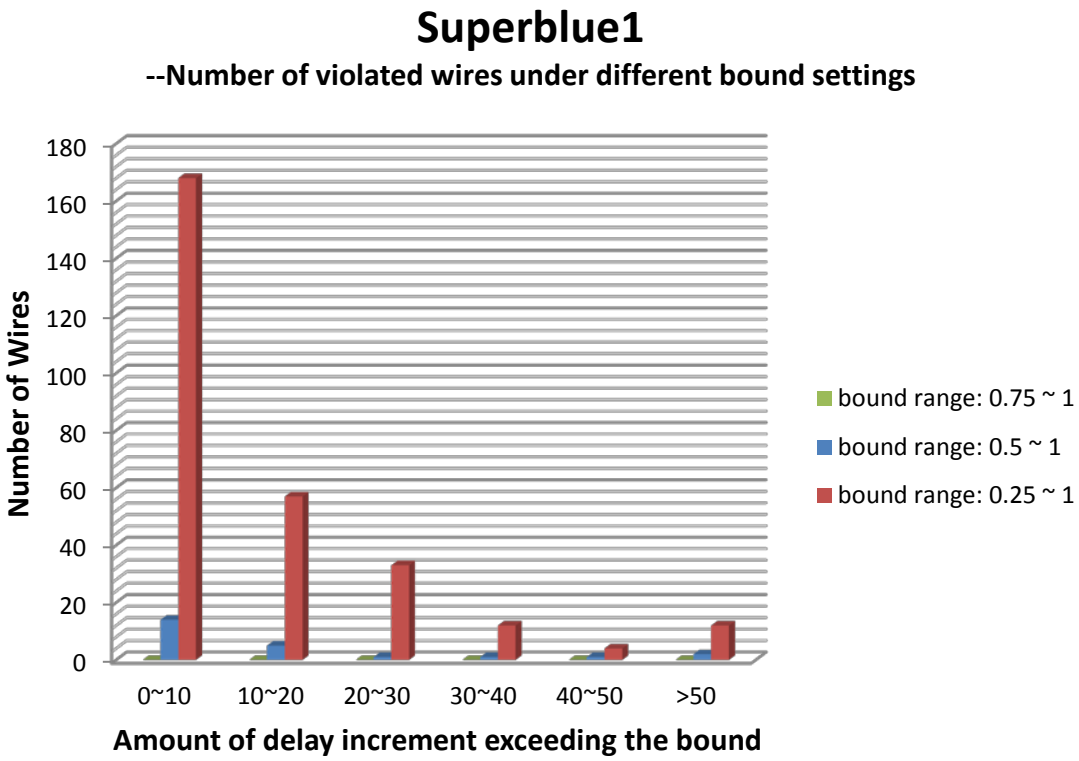

Fig. 19. Number of violated wires in test case Superblue1.

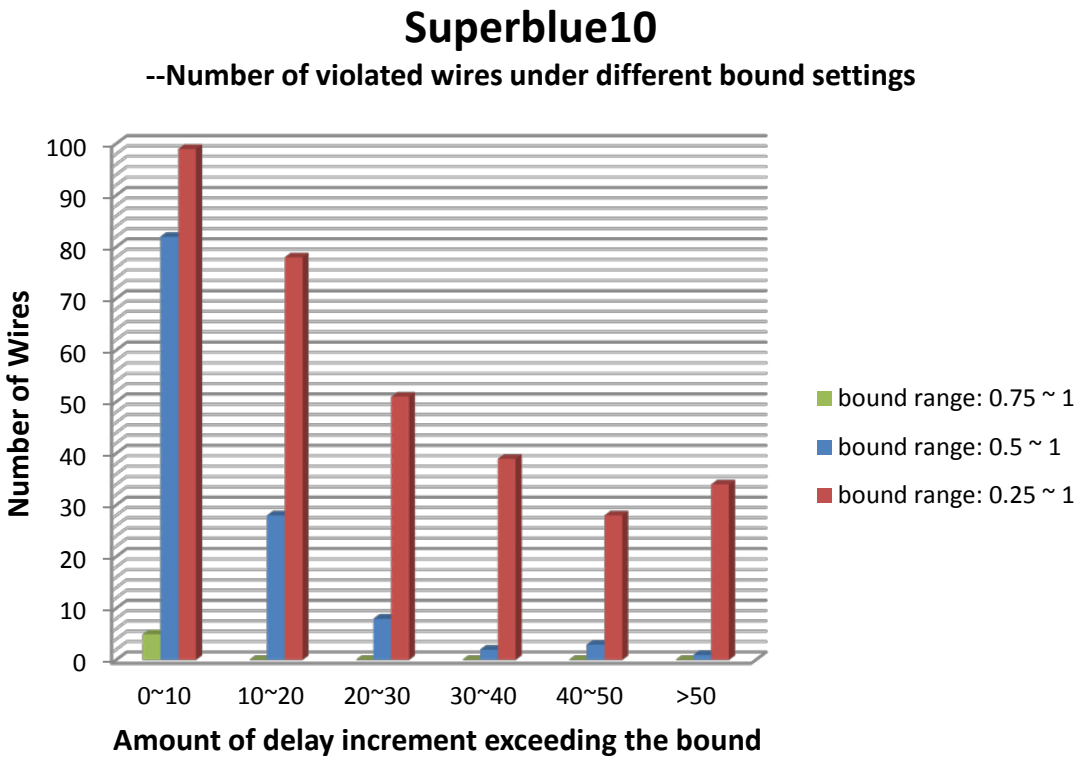

Fig. 20. Number of violated wires in test case Superblue10.

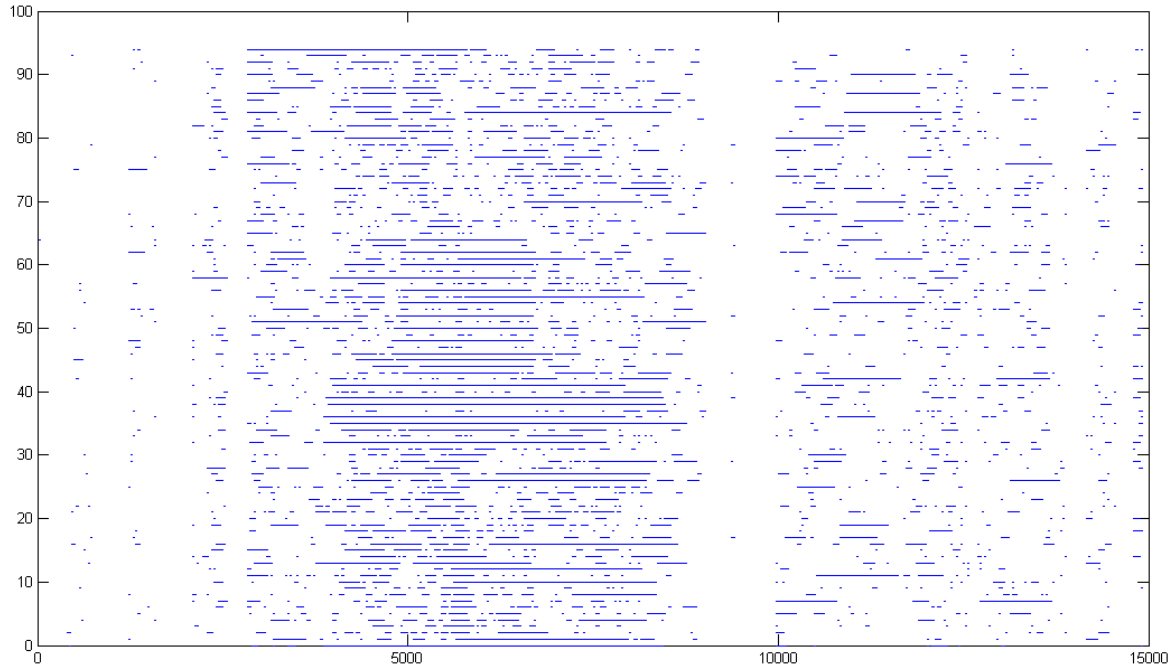

Fig. 21. Part of a panel after track assignment.

We also perform two other track assignment methods for comparison. One method recursively finds min-weight Hamiltonian paths for cliques in the set of unassigned wires, and assigns wires according to the order of vertices in path. The idea comes from Di Wu's work [5]. This method is referred to as Approach 2. The other one is similar to our approach but only considers wire conflict when doing the second step, and it is named as Approach 3. Table II shows that our approach is not only fast, but also superior in eliminating bound violations.

Besides, comparison is also made between the heuristic method and the ILP method. Table III gives the results of those two methods on several test cases. The heuristic is very fast that the run time is nearly zero for the small test cases, while the ILP becomes rather slow when constraints are tight even if the problem size is not big. However, the heuristic may not find the exact violation-free solution when routing resource is very limited. Instead, it gives a solution with some tolerance on bound violations. The ILP method can guarantee a feasible solution if there exists one, at the cost of long running time.

TABLE II

COMPARISON OF THREE APPROACHES

	Test Cases	Our Approach	Approach 2	Approach 3
Number of wires violating delay increment bounds	Superblue4 (total # wires 523,388)	461	48,179	22,526
	Superblue1 (total # wires 722,904)	286	65,470	27,484
	Superblue10 (total # wires 928,278)	329	75,951	30,310
Average amount of violation per violated wire	Superblue4	17.71	238.50	443.10
	Superblue1	14.18	215.79	305.03
	Superblue10	25.79	222.84	361.93
Total delay increment in the entire circuit	Superblue4	2.84×10^7	5.16×10^7	4.91×10^7
	Superblue1	3.00×10^7	6.42×10^7	4.53×10^7
	Superblue10	3.34×10^7	8.16×10^7	5.89×10^7
Run time	Superblue4	16’17’’	18’36’’	15’38’’
	Superblue1	24’46’’	24’47’’	22’7’’
	Superblue10	26’15’’	26’10’’	22’32’’

The number of available tracks is set to be 90, and the bound range is $(0.25d_{imax}, d_{imax})$, same for all three approaches. “ ’ ” stands for minute, and “ ’’ ” stands for second.

TABLE III

COMPARISON OF HEURISTIC AND ILP METHOD

Test Cases		Heuristic		ILP	
		Number of wires violating bounds	Run time	Number of wires violating bounds	Run time
# wires: 182 # tracks: 33	bound: (0.25~1)	2	0''	0	7'44''
	bound: (0.25~0.75)	4	0''	0	2hr 38'1''
# wires: 205 # tracks: 21	bound: (0.25~1)	2	0''	0	1hr 33'55''
	bound: (0.25~0.75)	7	0''	0	15hr 10'43''
# wires: 269 # tracks: 23	bound: (0.25~1)	0	0''	0	5'9''
	bound: (0.25~0.75)	1	0''	0	1hr 48'49''
# wires: 305 # tracks: 27	bound: (0.25~1)	4	0''	0	9hr 44'5''
	bound: (0.25~0.75)	5	0''	N/A	>48hr

Test cases are randomly picked, small enough for the ILP solver to solve.

5. CONCLUSION

In this work we propose and implement a novel heuristic approach to solve constraint-based track assignment problems. Our algorithm utilizes the nearest insertion method targeting at the Traveling Salesman Problem, and applies a greedy method on an extended Bipartite graph. Different from most previous works, our approach takes into account the dynamically changing delay increment on each wire segment. This adds great difficulty to implementation, but experimental results show that our approach has achieved significant success in eliminating performance degradation induced by crosstalk.

REFERENCES

[1] S. Batterywala, N. Shenoy, W, Nicholls, and H. Zhou, "Track Assignment: A Desirable Intermediate Step Between Global Routing and Detailed Routing," *in Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 59-66, 2002.

[2] B. W. Kernighan, D. G. Schweikert, and G. Persky, "An Optimum Channel-routing Algorithm for Polycell Layouts of Integrated Circuits," *in Proc. 10th Des. Automation Workshop*, pp. 50-59, 1973.

[3] P. Heydari and M. Pedram, "Capacitive Coupling Noise in High-Speed VLSI Circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, p. 478 , 2005.

[4] A. B. Kahng, S. Muddu, and E. Sarto, "On Switching Factor Based Analysis of Coupled RC Interconnects*", in Proc. IEEE/ACM Design Automation Conference*, pp. 79-84 , 2000.

[5] D. Wu, J. Hu, M. Zhao and R. Mahapatra, "Timing Driven Track Routing Considering Coupling Capacitance," *in Proc. Asia South Pac. Des. Autom. Conf.*, p. 1156, 2005.

[6] H. Zhou and D. F. Wong, "Global routing with crosstalk constraints," IEEE *Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 11, pp. 1683-1688, 1999.

[7] H.-P. Tseng, L. Sheffer and C. Sechen, “Timing- and crosstalk- driven area routing,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 4, pp. 528-544, 2001.

[8] T. Gao and C.L. Liu, “Minimum Crosstalk Channel Routing,” *in Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 692-696, 1993.

[9] R. Kay and R. A. Rutenbar, “Wire Packing: A Strong Formulation of Crosstalk-Aware Chip-Level Track/Layer Assignment with an Efficient Integer Programming Solution,” *in Proc. Int. Symp. on Physical Design*, pp. 61-68, 2000.

[10] H. Yao, Q. Zhou, X. Hong and Y. Cai, “Crosstalk Aware Routing Resource Assignment,” *J. Comput. Sci. Technol.*, vol. 20, pp. 231-236, 2005.

[11] T. Xue, E. S. Kuh and D. Wang, “Post Global Routing Crosstalk Risk Estimation and Reduction,” *in Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 302-309, 1996.

[12] R. Li , D. Zhou and D. Du, “Satisfiability and integer programming as complementary tools,” *in Proc. of the 2004 Asia and South Pacific Design Automation Conf.*, Yokohama, Japan, January 27-30, 2004.

[13] S. A. Cook, “The complexity if theorem proving procedures,” *in Proc. of 3rd Annual ACM Symposium on the Theory of Computing*, New York, 1971, pp. 151-158.

[14] J. P. Warners, “A linear-time transformation of linear inequalities into conjunctive normal form,” *Information Processing Letters*, vol. 68, pp. 63-69, 1998.

[15] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, “PBS: A Backtrack Search Pseudo Boolean Solver and Optimizer,” *in Symposium on the Theory and Applications os Satisfiability Testing (SAT)*, pp. 346-353, 2002.

[16] R. Diestel, *Graph Theory*. Heidelberg, Germany: Springer-Verlag, 2010.

[17] D.L. Applegate, R.E. Bixby, V. Chvatal and W.J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ: Princeton University Press, 2007.

[18] H. Shojaei, A. Davoodi, and J. Linderoth, “Congestion analysis for global routing via integer programming,” *Int'l Conf. on Computer-Aided Design (ICCAD'11)*, pp. 256-262, November 2011.

[19] D. P. Mehta and S. Sahni, *Handbook of data structures and applications*, Chapman and Hall/CRC, 2005.

[20] D. J. Rosenkrantz, R. E. Stearns and P. M. Lewis, “An analysis of several heuristics for the traveling salesman problem,” *SIAM J. Comput*., vol. 6, pp. 563 - 581, 1977.

VITA

Qiong Zhao received her Bachelor of Engineering degree in communication engineering from Beijing University of Posts and Telecommunications, Beijing, China in 2009. She graduated with Master of Science degree in computer engineering from Texas A&M University in May 2012. Her research interests include VLSI Computer Aided Design algorithms for floorplanning, routing and track assignment.

She can be reached at the Department of Electrical and Computer Engineering, 238 Zachry Engineering Center, Texas A&M University, College Station, TX 77843-3128. Her email is *qiongzhao@tamu.edu*.