Negotiation-Based Track Assignment Considering Local Nets*

Man-Pan Wong[1], Wen-Hao Liu[2], and Ting-Chi Wang[1]
[1]Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[2]Block Implementation, ICD, Cadence Design Systems, Austin, TX, USA
vastskymeteor@hotmail.com, whliu@cadence.com, tcwang@cs.nthu.edu.tw

Abstract– **Routability has become a very challenging issue in a modern VLSI design flow. Many works use global routing to estimate the routability in the early design stages. However, global routing cannot accurately capture local congestion, so it is hard to detect the detailed routability issue. To more accurately estimate the detailed-routing routability, this paper presents a track-assignment-based routability estimator. In this work, wire segments called *iroutes* are extracted from a global routing result, and then the proposed negotiation-based algorithm assigns these iroutes to proper tracks and minimizes the overlaps between the iroutes. Based on the assignment result, we can judge which regions may have critical routability issues by seeing where more overlaps reside.**

1 Introduction

As technology node advances, more and more issues like via spacing, timing, and reliability need to be considered during the routing phase. As a result, routing has become one of the most challenging phases in VLSI physical design flow. To reduce the complexity of the routing problem, the routing problem is typically solved by three stages: ***global routing***, ***track assignment***, and ***detailed routing***. The global routing stage partitions the routing region into an array of global cells (g-cells), and determines that each global net needs to route through which g-cells. Next, track assignment treats each straight wire, which passes through one or more complete g-cells, as an ***iroute***, and then assigns each iroute to a routing track within its corresponding g-cells. Finally, detailed routing realizes the interconnections between pins and iroutes for global nets, and also constructs routing trees for local nets.

In order to fix the routability issue in the early design stages, routability estimation become an important problem. Recently, the works of [1-3] predict the routing difficulty of a design based on the congestion and wiring information reported by global routers. However, the studies of [4-6] indicate that there is a big mismatch between global routing and detailed routing because global routing ignores many detailed routing issues like pin access, local net wiring, and via rules. To capture these issues, track assignment is able to extract more detailed routability information compared to global routing.

This paper is the first study to employ track assignment to do better routability estimation following global routing. In track assignment stage, an iroute overlapping with another iroute or blockages is treated as a violation. The goal of existing track assignment works [7-14] is to maximize the number of assigned iroutes to generate a violation-free routing draft for detailed routing. Therefore, some iroutes may not be assigned by the existing track assignment algorithms. However, when track assignment is adopted to do routability estimation, it needs to report where violations are, and the criticality of the violations. Thus, the objective of this work is to assign every iroute to a track such that the amount of resultant violations is as small as possible.

Because a modern physical design flow needs to frequently estimate the routability of a design in different stages, the runtime of routability estimation is a critical issue. However, traditional track assignment is a one-time job in a physical design flow, so existing track assignment algorithms typically spend longer runtime to pursue higher solution quality. As a result, existing algorithms are not fast enough for routability estimation. The paper of [7] is the first work employing track assignment to guide detailed routing, and it adopts a weighted bipartite matching algorithm to solve the track assignment problem. The runtime of the weighted bipartite matching algorithm increases dramatically when the problem size increases. In addition, the works of [8-10], [11-13], and [14] respectively extend the track assignment to consider the issues of timing, yield, and double-patterning. However, with consideration of these issues, the works of [8-14] use more time-consuming algorithms like integer linear programming to solve the track assignment problem. As a result, their runtime is also unaffordable for massive routability estimation calls.

This paper proposes a negotiation-based track assignment algorithm called NTA for fast routability estimation. NTA consists of two stages: the initial assignment stage uses a greedy algorithm to efficiently identify an initial track assignment result in which iroute overlapping is allowed, and then the overlap reduction stage iteratively rips up and re-assigns iroutes to eliminate iroute overlapping as much as possible. The following advantages make NTA estimate routability more accurately than global routing.

- Different from global routing that only routes global nets, NTA takes both global nets and local nets into account.
- Because more iroute overlapping in a region implies that detailed routing needs to spend more effort to fix short violations in this region, NTA can indicate the locations of hard-to-route regions by reporting where iroute overlapping resides.
- The runtime and accuracy of NTA are easily tunable. If designers want to estimate routability very quickly by NTA, they can reduce the number of rip-up and re-assignment iterations in the overlap reduction stage of NTA. In contrast, if the overlap reduction stage runs more iterations, NTA can more accurately report which regions are truly hard-to-route.

The rest of this paper is organized as follows. Section 2 introduces the track assignment problem addressed in this work. Sections 3 and 4 respectively detail the algorithms used in the initial track assignment and overlap reduction stages of NTA. Section 5 shows the experimental results. Finally, the conclusions are drawn in Section 6.

2 Preliminaries

Section 2.1 will introduce how this work extracts iroutes from a global routing result, and then Section 2.2 defines the track assignment problem for the iroutes. Finally, Section 2.3 details the evaluation metrics for the assignment results.

2.1 IRoute Extraction

In order to capture more accurate routability information, the definitions of iroutes in this work and previous works are different. In previous works [7-14], only a straight wire that fully passes one or multiple g-cells is treated as an iroute. For example, Fig. 1(a) shows a global routing result with two nets. Net n_1 is a global net whose global routing path is highlighted by a blue box, and net n_2 is a local net because its pins are all in a single g-cell. In previous works, only wire segment w_1 shown in Fig. 1(b) will be treated as an iroute, and other wire segments will be ignored. As a result, a lot of routing information will be lost.

In this work, iroutes are extracted from both global and local nets. Given a global routing result of a global net, a straight wire from a g-cell's center to another g-cell's center is treated as an iroute. Therefore, this work extracts wire segments w_2 and w_3 from net n_1 to be iroutes (see Fig. 1(c)).

*This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant No. MOST 104-2220-E-007-014.

978-1-4673-9569-4/16/$31.00 ©2016 IEEE

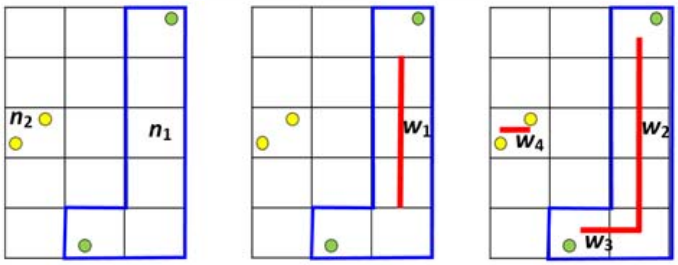


Fig. 1. (a) Global routing results with two nets; (b) an iroute extracted by previous works; (c) iroutes extracted by this work.

Since a local net is typically ignored by global routing, there is no routing result associated with the local net. To capture the impact of a local net on track assignment, the routing topology of the local net needs to be known. To this end, this work constructs two rectilinear Steiner trees, namely single vertical-trunk Steiner tree and single horizontal-trunk Steiner tree for each local net. The single vertical-trunk (horizontal-trunk) Steiner tree is built by first constructing the vertical trunk (horizontal trunk) and then connecting all the pins of the local net to the trunk by horizontal (vertical) line segments. The x-coordinate (y-coordinate) of the vertical (horizontal) trunk is determined by the median of the x-coordinates of all pins, and the top and bottom y-coordinates (the left and right x-coordinates) of the vertical (horizontal) trunk are respectively set to be the largest and smallest y-coordinates (x-coordinates) of all pins. Fig. 2 shows an example of the single vertical-trunk Steiner tree and the single horizontal-trunk Steiner tree for a local net of five pins. After the two Steiner trees are built, their lengths are calculated and compared. The one with a smaller tree length is then chosen and its associated trunk becomes the iroute for the net. Note that the location of the chosen trunk is determined during the construction of the corresponding Steiner tree, but it will become variable in our track assignment problem. In Fig. 1(c), w_4 denotes an iroute extracted from a local net.

2.2 Problem Formulation

Let ***N*** denote the set of all nets (including local nets), where each net contains a set of pins and a set of iroutes. Let ***P*** denote the set of all panels, where each panel consists of the set of all g-cells in a row or column of a routing layer. For each panel ***p***, there is a set ***T(p)*** of tracks and a set ***I(p)*** of iroutes associated with it. Routing blockages are also considered, where a blockage may partially or completely occupy one or more tracks.

For each panel p, our track assignment problem asks to assign each iroute in $I(p)$ to a track in $T(p)$. Different from the previous track assignment works [7-14] that may only assign a sub-set of iroutes in $I(p)$ to generate an overlapping-free track assignment result, this work assigns all the iroutes to tracks so that we can estimate where critical routability issue may happen based on the overlap information among iroutes. The objective of this work is to minimize *the total overlap cost*, *the total blockage cost*, and *the total wirelength cost* induced by the resultant track assignment solution.

2.3 Evaluation Metrics

This section will introduce how to compute the overlap, blockage, and wirelength costs to evaluate the quality of a track assignment result.

2.3.1 Overlap Cost and Blockage Cost

Each iroute corresponds to an interval. If two iroutes are assigned to the same track and their intervals overlap, they are said to be ***overlapping iroutes***. The overlap cost induced by assigned iroutes is calculated as follows.

For each track, if it has no overlapping iroutes, its overlap cost is zero; otherwise its overlap cost is determined by overlapping iroutes. A track with m assigned iroutes can be represented by a set of constituent intervals, where each interval is a common sub-interval of 0, 1, ..., or m assigned iroutes. For each interval that is a sub-interval of k ($2 \leqq k \leqq m$) assigned iroutes, its overlap cost is defined to be the product of its interval length and k. For the rest of the intervals (i.e., the intervals that are sub-intervals of 0 or 1 assigned iroute), their overlap costs are all 0. The overlap cost of a track is the sum of the overlap costs of all its constituent intervals. The overlap cost of a panel is the sum of the overlap costs of all its tracks, and the sum of the overlap costs of all panels is the total overlap cost of a track assignment solution.

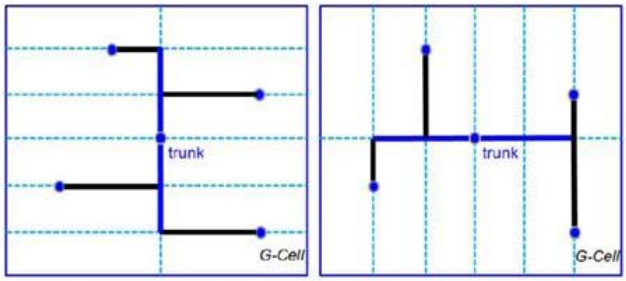


Fig. 2. Example of (a) a single vertical-trunk Steiner tree, and (b) a single horizontal-trunk Steiner tree.

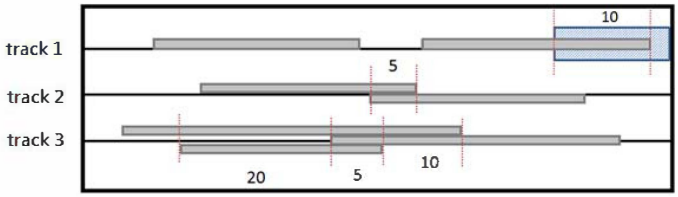


Fig. 3. A track assignment result for a panel with three tracks, seven iroutes, and one blockage.

If an iroute is assigned to a track and overlaps with one or more blockages that partially or completely occupy the track, the blockage cost of the iroute is defined to be the sum of the overlap length between the iroute and every involved blockage. The blockage cost of each track is the sum of the blockage costs of all its assigned iroutes, and the blockage cost of each panel is the sum of the blockage costs of all its tracks. The total blockage cost of a track assignment solution is then calculated by summing up the blockage costs of all the panels.

Fig. 3 shows a track assignment result for a panel that has 3 tracks and 7 iroutes. There is also a blockage (i.e., the blue rectangle) that occupies a part of track 1. The numbers in Fig. 3 denote the overlap lengths either between iroutes or between an iroute and a blockage. According to the above-mentioned cost definitions, the overlap and blockage costs of each track and the panel are calculated as follows:

track 1: overlap cost: 0, blockage cost: 10
track 2: overlap cost: 5*2 = 10, blockage cost: 0
track 3: overlap cost: 20*2+5*3+10*2 = 75, blockage cost: 0
panel: overlap cost: 0+10+75 = 85, blockage cost: 10+0+0 = 10

2.3.2 Wirelength Cost

Each iroute or pin is a ***net component***, and all net components of a net will be connected later during detailed routing. A carefully selected track for each iroute can help for minimizing the wirelength of each net and finding a better solution in detailed routing. In this work, we use a minimum spanning tree (MST) approach to compute the wirelength cost, because the MST length of a net has high correlation to its routed wirelength as indicated in [15]. We use Fig. 4 to explain how to calculate the wirelength cost for a net from a track assignment result. Once the wirelength cost of each net is known, they will be summed up to get the total wirelength of the track assignment solution.

Fig. 4(a) shows a net that has three assigned iroutes and two pins. The top horizontal panel has iroute ir_1 and pin p_1, while the bottom horizontal panel has iroute ir_3 and pin p_2. The vertical panel that has

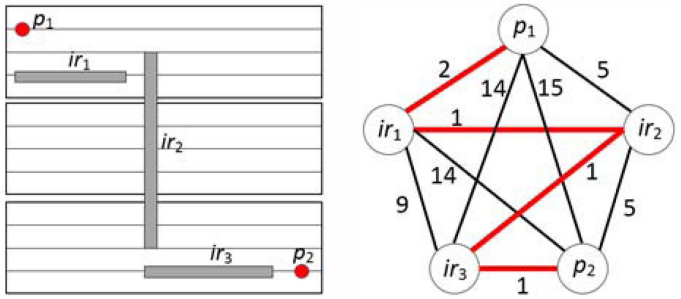


Fig. 4. An example of wirelength cost calculation.

iroute ir_2 is not shown here (note that the middle horizontal panel is given but it contains no net components for the net). To calculate the wirelength cost of the net, a weighted complete graph is built first, where each vertex denotes a net component, and the weight on each edge denotes the shortest Manhattan distance between two corresponding net components. Then an MST is found from the graph to connect all net components. Finally, the length of the MST is used as the wirelength cost for the net. For the net given in Fig. 4(a), Fig. 4(b) shows the corresponding complete graph and MST (whose tree edges are marked with red thick line segments).

3 Initial Assignment

This work presents a negotiation-based track assignment algorithm called NTA, which consists of the initial assignment stage and the overlap reduction stage. The initial assignment stage uses a greedy algorithm to efficiently identify an initial track assignment result in which overlapping among iroutes is allowed, and then the overlap reduction stage adopts a negotiation scheme to iteratively rip up and re-assign iroutes for eliminating iroute overlapping as much as possible. Both stages work in a panel-by-panel manner. The initial assignment stage is the focus of this section, while the overlap reduction stage is detailed in Section 4.

Algorithm 1 shows the pseudo code of the initial assignment stage for processing a panel p. At first, line 1 sorts all the iroutes in $I(p)$. Based on the sorted order, the loop from lines 2 to 15 assigns each iroute one-by-one to a track that induces the minimum cost. For each iroute, lines 4 to 13 compute the cost for each track and then line 14 assigns the iroute to the track with minimum cost. More details about the cost computation are described in the next three sub-sections.

Because the assignment order of iroutes will impact the assignment result, we have tried different orders to find the best one that has the minimum cost. Finally, we found assigning iroutes from long to short can get the best result, so Algorithm 1 traverses iroutes based on their lengths in a non-increasing order.

3.1 Calculation of Wirelength Cost

According to the sorted order, each iroute is sequentially assigned. During the course of assigning an iroute, all the tracks in the panel are candidates for assignment, and each possible track assignment will produce a wirelength cost. Line 5 of Algorithm 1 computes the wirelength cost when iroute *ir* is assigned to track *t*. However, to save the runtime, we do not use the MST approach as introduced in Section 2.3.2 to get the wirelength cost. Instead, we first find the shortest Manhattan distance between iroute *ir* and each of the net components that are of the same net as *ir*. Note that among these net components, if some iroutes have not assigned tracks yet, those iroutes are ignored by the distance calculation. Namely, only the iroutes with known locations and the pins are involved in the distance calculation. Then the smallest one among these distance values is used as the wirelength cost for this possible track assignment. Using this approximate approach to compute wirelength cost is much faster than the MST approach, and the quality of the track assignment results only slightly degrades.

Algorithm 1 : Initial Assignment for a Panel
Input: Panel p, iroute set $I(p)$, and track set $T(p)$
1. Sort $I(p)$ based on their lengths in a non-increasing order
2. **foreach** iroute *ir* in $I(p)$ traversed by their sorted order
3. minCost = ∞
4. **foreach** track *t* in $T(p)$
5. wlCost = CalculateWirelengthCost(*ir*, *t*)
6. overlapCost = CalculateOverlapCost(*ir*, *t*)
7. blkCost = CalculateBlocakgeCost(*ir*, *t*)
8. cost =wlCost+α*overlapCost + β*blkCost
9. **if** (cost < minCost) **then**
10. minCost = cost
11. *minCostTrack* = *t*
12. **end if**
13. **end foreach**
14. Assign *ir* to *minCostTrack*
15. **end foreach**

3.2 Calculation of Overlap Cost and Blockage Cost

Recall that our track assignment problem allows overlapping between two assigned iroutes or between an assigned iroute and a blockage to happen on a track. Therefore to minimize the overlap cost and blockage cost, when an iroute is being considered to be assigned to a track, how much the overlap cost and blockage cost will increase for this track should be known. Lines 6 and 7 of Algorithm 1 respectively compute the increases of the overlap cost and blockage cost for track *t* by the method introduced in Section 2.3.1.

3.3 Overall Cost Function

To minimize the overlap cost, blockage cost and wirelength cost simultaneously is a hard task, because we have experimentally observed that when one cost gets improved, it is likely that the others become worsened. As a result, how to reach a good balance among these three costs is of the utmost importance. To achieve this goal, line 8 of Algorithm 1 uses Eq.(1), a weighted sum of the three costs, to evaluate how good it is when assigning an iroute to a track.

$$\text{wlCost} + \boldsymbol{\alpha}\text{*overlapCost} + \boldsymbol{\beta}\text{*blkCost} \quad (1)$$

In Eq. (1), wlCost, overlapCost, and blkCost denote the wirelength cost, overlap cost, and blockage cost, respectively. In addition, α and β are user-defined parameters. In the current implementation of the initial assignment stage, α and β are respectively set to be 0.1 and a very large number. The rationale for setting β as a very large number is to avoid the generation of overlapping with blockages as much as possible, because wires crossing blockages is a critical routability issue. The very large number for β makes this stage generate a result with the minimal blockage cost. On the other hand, by setting α to be 0.1, we can make the initial assignment stage put more emphasis on reduction of the wirelength cost than the overlap cost. As a result, the initial assignment stage is likely to produce a track assignment solution that has shorter net connections but a higher overlap cost. Fortunately, the issue of overlap cost will be handled in the overlap reduction stage.

4 Overlap Reduction

As stated in the previous section, the initial assignment stage will produce a track assignment result that has a lower wirelength cost but a higher overlap cost. Therefore, to find a good balance between the two costs, the overlap reduction stage adopts a negotiation scheme to reduce the overlap cost by carefully trading the wirelength cost.

Algorithm 2 shows the pseudo code of the overlap reduction stage for processing a panel p. Line 1 computes the cost for each iroute. The loop from lines 2 to 8 iteratively rips up the maximum-cost iroute, and

***Algorithm* 2** : Overlap Reduction for a Panel
Input: Panel p, assigned iroute set $I(p)$, track set $T(p)$
1. CalculateIrouteCost()
2. **repeat**
3. r_{max} = SelectMaxCostIRoute($I(p)$)
4. RipupIRoute(r_{max})
5. t_{min} = FindMinCostTrack(r_{max}, $T(p)$)
6. Assign r_{max} to t_{min}
7. Update history cost for t_{min}
8. **until** (Termination condition is triggered)

re-assigns the iroute to the track with the minimum assignment cost, until the termination condition is met. In the loop, line 3 selects the maximum-cost iroute denoted by r_{max} among $I(p)$. Line 4 rips up r_{max} from its original track and updates the cost for each remaining iroute on that track after r_{max} is removed. Line 5 tentatively assigns r_{max} to each track in $T(p)$ and then identifies the track t_{min} that has the minimum assignment cost. Line 6 assigns r_{max} to track t_{min} and updates the cost for each iroute (including r_{max}) on t_{min}. Finally, line 7 updates the ***history cost*** for t_{min}, which is considered during the cost computation for iroute selection and re-assignment. The details about how to select and re-assign iroutes will be respectively introduced in Sections 4.1 and 4.2, and Section 4.3 will detail how to update the history cost for a track.

The rip-up and re-assignment process continues until either the iteration count meets a user defined value m or the improvement rate of the total assignment cost is lower than 5 percent. In this work, we set m to 2*$|I(p)|$, where $|I(p)|$ denotes the number of iroutes in panel p.

4.1 Iroute Selection for Re-assignment

Line 3 of Algorithm 2 selects an iroute with the maximum cost for re-assignment by using the following cost function:

$$\text{overlapCost}(ir, t) + \text{historyCost}(ir, t) \qquad (2)$$

The OverlapCost(ir, t) denotes how much overlap cost will be reduced if the iroute ir is removed from its originally residing track t, and is computed according to the definition of overlap cost introduced in Section 2.3.1. The historyCost(ir, t) denotes the sum of the history costs of all unit-intervals covered by the iroute ir on t. Each unit-interval has one-unit length, and ir covers a set of continuous unit-intervals on t. The history cost of each unit-interval on t is initially set to be 0 at the beginning of the overlap reducing stage, and it will be increased by 1 every time right after an iroute is assigned to t and causes overlapping with another iroute for the unit-interval. More details of how to update the history cost for a unit-interval will be given in Section 4.3. Note that, because the blockage cost is already minimized in the initial assignment stage, no blockage cost can be further improved here. Thus, Eq. (2) does not consider the blockage cost for iroute selection for re-assignment.

Fig. 5(a) gives an example to illustrate the rip-up process for an iroute. It shows a track assignment result of a panel from the initial assignment stage, where all unit-intervals on each track have their history costs initialized as 0. It is clear to see that iroute a has the maximum cost because it induces the maximum reduction on the overlap cost if got removed, among all iroutes in the panel. After removing iroute a, the costs of the remaining iroutes b and c will be updated; the new overlap costs for them will become 0.

By experiments, we found our greedy iroute selection strategy may always select a same set of iroutes such that the assignment solution falls into local optimum. To avoid repeatedly ripping up and re-assigning a same set of iroutes, once an iroute gets ripped up and re-assigned, the iroute will be frozen for f iterations before it becomes a candidate again for the iroute selection. In our implementation, we set f to be 20.

4.2 Iroute Re-assignment

The re-assignment step finds a track with the minimum cost for assigning an iroute by using the following cost function:

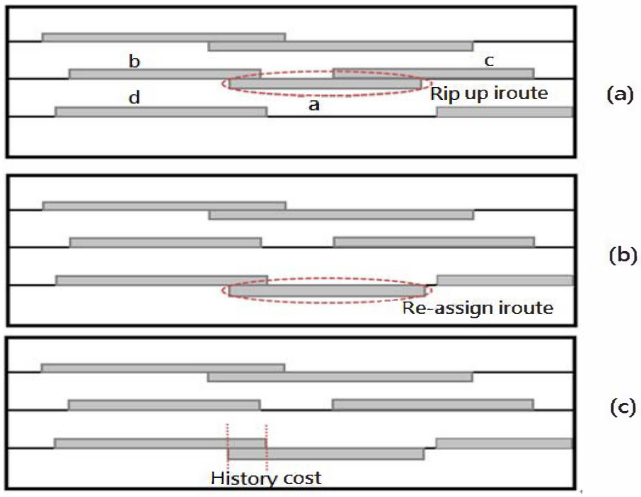


Fig. 5. Example of negotiation-based track assignment.

$$0.1*\text{wlCost}(ir, t) + \alpha_1 *\text{overlapCost}(ir, t) + \beta*\text{blkCost}(ir, t) + \text{historyCost}(ir, t) \qquad (3)$$

The wlCost(ir, t) denotes the wirelength cost of the net associated with ir when ir is assigned to track t. The overlapCost(ir, t) and blkCost(ir, t) respectively denote how much overlap cost and blockage cost on t will increase when ir is assigned to t. The wirelength, overlap and blockage costs are computed in the same ways as those in the initial assignment stage. The historyCost(ir, t) denotes the sum of the history costs of all unit-intervals covered by ir when ir is assigned to t.

There are two user-specified parameter α_1 and β in Eq. (3). The parameter α_1 is set to 0.1 at the beginning of the overlap reduction stage, and its value is increased by 0.1 after every $|I(p)|/k$ iterations, where $|I(p)|$ is the number of iroutes in the panel p and k is set to 10 in our implementation. Since we attempt to improve the overlap cost on the premise of less impact on the wirelength cost at early iterations, we put more emphasis on reducing the overlap cost in the middle and late iterations, and thus the value of α_1 increases gradually. The parameter β is a very large constant to avoid the increase of the blockage cost in the overlap reduction stage.

Let us continue to use the example in Fig. 5 as an example to illustrate the re-assignment step. Fig. 5(b) shows that the bottom track has the minimum increase in the overlap cost and thus has the minimum overlap cost if got assigned for iroute a, among all tracks in the panel; hence iroute a is assigned to the bottom track (assuming the wirelength cost is negligible). As shown in Fig. 5(c), after this assignment, iroute a overlaps with another iroute on the bottom track, and therefore the overlap costs of both iroutes need to be updated. In addition, the unit-intervals covered by the overlapping area have their history costs increased by 1.

4.3 History Cost

The history cost of a unit-interval is used to count how many times of the past iterations the unit-interval has overlap induced by iroute re-assignment. The history cost of a unit-interval is initialized as 0 at the beginning of the overlap reduction stage, and is increased by 1 every time after an iroute is assigned to the unit-interval and introduces an overlap.

Fig. 6 shows an example to illustrate how to update the history cost for a track. The top row shows the history cost of each unit-interval for the track that has five iroutes, two of which overlap and the other three of which overlap. The bottom row shows the resultant history cost of each unit-interval after another iroute is re-assigned to the track. As can be seen from the figure, the newly assigned iroute causes overlapping with the original five iroutes as indicated by the two violation intervals.

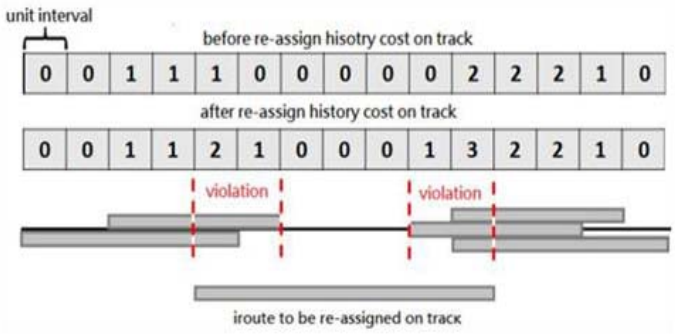
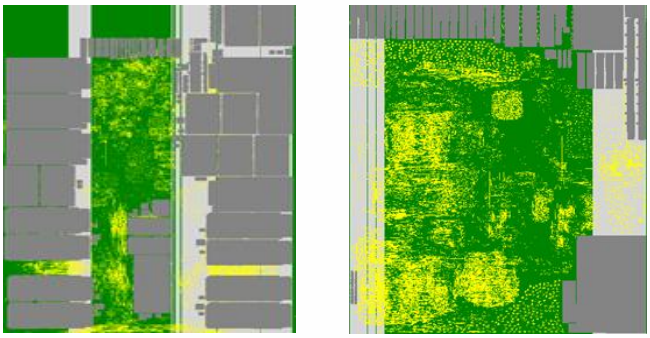


Fig. 6. History cost calculation.



(a) (b)

Fig. 7. Iroute overlapping map of (a) sb11 and (b) sb12.

TABLE. 1. COMPARISON OF WIRELENGTH COST (WL), OVERLAP COST (OC), AND BLOCKAGE COST (BC) BETWEEN THE INITIAL ASSIGNMENT STAGE AND OVERLAP REDUCTIONS STAGE.

Circuit	Initial Assignment Stage			Overlap Reduction Stage		
	WL (10^6)	OC (10^6)	BC (10^6)	WL (10^6)	OC (10^6)	BC (10^6)
sb2	25.31	24.29	0.34	26.88	5.55	0.34
sb3	24.04	11.20	0.33	25.37	3.91	0.33
sb6	24.96	11.24	0.23	26.79	3.28	0.23
sb7	37.75	13.33	0.21	40.39	3.06	0.21
sb9	20.85	7.72	0.15	22.68	1.96	0.15
sb11	22.24	8.06	0.22	23.79	1.62	0.22
sb12	36.13	12.88	0.04	39.13	1.63	0.04
sb14	15.48	6.57	0.26	16.27	1.65	0.26
sb16	16.65	8.18	0.14	17.68	2.52	0.14
sb19	12.29	4.38	0.09	13.34	1.01	0.09
Ratio	1	1	1	1.07	0.72	1

TABLE. 3. COMPARISON BETWEEN THE TRACK ASSIGNMENT RESULTS WITH AND WITHOUT LOCAL NET CONSIDERATION.

Circuit	With Local Nets			Without Local Nets		
	WL (10^6)	OC (10^6)	BC (10^6)	WL (10^6)	OC (10^6)	BC (10^6)
sb2	26.88	5.55	0.34	23.08	4.64	0.26
sb3	25.37	3.91	0.33	21.81	2.56	0.24
sb6	26.79	3.28	0.23	22.32	1.98	0.20
sb7	40.39	3.06	0.21	34.19	1.60	0.19
sb9	22.68	1.96	0.15	18.36	1.06	0.13
sb11	23.79	1.62	0.22	19.83	0.85	0.18
sb12	39.13	1.63	0.04	32.95	0.88	0.03
sb14	16.27	1.65	0.26	14.08	0.96	0.22
sb16	17.68	2.52	0.14	14.86	1.50	0.13
sb19	13.34	1.01	0.09	10.88	0.42	0.08
Ratio	1	1	1	0.84	0.581	0.849

The history cost of each unit-interval within the violation intervals is increased by 1.

From our observation, the congestion of some tracks, in terms of the amount of iroute overlapping, may be relatively higher than the others in a panel. As a result, tracks with higher congestion tend to have larger history costs for their constituent unit-intervals. By taking history cost into account during the calculation of the cost of an iroute and the cost of a track, the overlap reduction stage will try to move iroutes from more congested tracks to less congested tracks.

5 Experimental Results

The proposed NTA is implemented in C/C++ language on a hexa-core 2.4 GHz Intel Xeon-based linux server with a 20GB memory. To generate global routing results to be the test cases for NTA, we adopt a global router NCTUgr [16] to route the placement results of NTUplace4 [17] that were released by the DAC12 placement contest [18]. The global routing result of each test case is overflow-free, but as shall be seen soon, no overlap-free track assignment result can be found by NTA and a prior work.

5.1 Comparison between the Initial Assignment Stage and the Overlap Reduction Stage

Table 1 shows the experimental results of the wirelength cost (WL), the overlap cost (OC), and the blockage cost (BC) obtained by the initial assignment stage and the overlap reduction stage of NTA. Table 1 reveals that the overlap reduction stage significantly improves the overlap cost by 28% while increasing the wirelength cost by 7% on average. Meanwhile, the blockage cost remains unchanged after the overlap reduction stage.

5.2 Comparison between a Prior Work and NTA

This section compares NTA with an existing track assignment algorithm called WBM that is based on weighted bipartite matching [7]. Because the binary code of WBM is not available to us, we implement it by ourselves. The objective of WBM is to maximize the number of assigned iroutes without introducing any overlap, so some iroutes are not assigned by WBM. Therefore, in order to make an apple-to-apple comparison, we add a post assignment stage after WBM to assign the remaining iroutes to tracks, and name the overall approach WBM-TA. The post assignment stage is similar to our initial assignment stage, except that it starts with a track assignment solution produced by WBM.

Table 2 compares the track assignment results produced by NTA and WBM-TA. Both NTA and WBM-TA generate the same blockage cost for each test case, but NTA can respectively achieve 8.9% shorter wirelength and 75.3% lower overlap cost on average than WBM-TA. According to our observation, many long iroutes are not assigned by WBM, and cause more serious iroute overlapping and much less flexibilities for the post assignment stage to assign them. Because NTA can better minimize iroute overlapping, most easy-to-solve overlaps can be resolved by NTA and only the hard-to-solve overlaps remain. Thus, NTA can more accurately indicate which regions are truly hard-to-route.

Table 2 indicates that NTA using a single core is averagely 2.23X faster than WBM-TA; this demonstrates that NTA is more suitable to serve fast routability estimation. Moreover, the algorithm of NTA can be easily parallelized. Each panel can be concurrently processed in both the initial assignment stage and overlap reduction stage by assigning each panel to a various core. Our experiment reveals that NTA can respectively get 1.92X, 3.83X, and 7.22X average speedup when it is run on 2, 4, and 8 cores.

5.3 Routability Estimation from Track Assignment

Table 3 compares the assignment results of NTA with local net consideration and without local net consideration. From the results of overlap cost and blockage cost, we can tell that they increase greatly after adding local nets, which signifies that global routing does not utilize local nets to estimate routability, making the estimation

TABLE. 4. COMPARISON OF THE TRACK ASSIGNMENT RESULTS BETWEEN NTUPLACE4'S AND SIMPLR'S PLACED CIRCUITS.

Circuit	NTUplace4			SimPLR		
	WL (10^6)	OC (10^6)	BC (10^6)	WL (10^6)	OC (10^6)	BC (10^6)
sb2	26.88	5.55	0.34	31.04	7.01	0.38
sb3	25.37	3.91	0.33	27.60	5.29	0.35
sb6	26.79	3.28	0.23	27.93	3.92	0.25
sb7	40.39	3.06	0.21	42.08	4.09	0.27
sb9	22.68	1.96	0.15	23.79	2.64	0.16
sb11	23.79	1.62	0.22	24.78	2.56	0.28
sb12	39.13	1.63	0.04	41.13	1.88	0.08
sb14	16.27	1.65	0.26	17.32	2.28	0.28
sb16	17.68	2.52	0.14	18.72	3.35	0.22
sb19	13.34	1.01	0.09	14.12	1.30	0.13
Ratio	1	1	1	1.065	1.322	1.334

unreliable. Consequently, it is desired to consider local nets in track assignment, which results in a more accurate prediction of congested regions.

To see the impact of different placers on routability, we also use NTA to compare the routability of the placement solutions obtained by NTUplace4 [17] and SimPLR [19]. Both sets of placed circuits have overflow-free global routing results produced by NCTUgr [16]. This means the routability of placement solutions obtained by NTUplace4 and SimPLR has no big difference from the view point of global routing. However, from Table 4, it can be seen that by running NTA on the two sets of test cases, different results of wirelength cost, overlap cost and blockage cost are obtained. These track assignment results are helpful for designers to see where congested regions are and where violations happen. For instance, Fig. 7 shows the iroute overlapping map of sb11 and sb12 placed by NTUplace4, in which the yellow regions highlight the locations of iroute overlapping that have higher probability to cause the routability issue.

6 Conclusions

Because global routing cannot capture local congestion and many routing rules, it is hard to accurately detect the detailed routability issue. In order to overcome this shortage, this paper proposed a negotiation-based track assignment approach (NTA) which can estimate the routability from a closer view to detailed routing. Consequently, a designer can have a better idea of where routing congestions happen and how severe they are in a circuit through the observation of the track assignment solution found by NTA.

REFERENCES

[1] H. Shojaei *et al.*, "Congestion analysis for global routing via integer programming," in *Proc. ICCAD*, pp. 256-262, 2011.

[2] W.-H. Liu *et al.*, "A fast maze-free routing congestion estimator with hybrid unilateral monotonic routing," in *Proc. ICCAD*, pp. 713-719, 2012.

[3] W.-H. Liu *et al.*, "Routing congestion estimation with real design constraints," in *Proc. DAC,* 2013.

[4] H. Shojaei *et al.* , "Planning for local net congestion in global routing," in *Proc. ISPD*, pp. 85 – 92, 2013.

[5] W.-H. Liu *et al.* "Case study for placement solutions in ispd11 and dac12 routability-driven placement contests," in *Proc. ISPD*, pp. 114 – 119, 2013.

[6] V. Yutsis *et al.*, "ISPD 2014 benchmarks with sub-45nm technology rules for detailed-routing-driven placement," in *Proc. ISPD*, pp. 161 – 168, 2014.

[7] S. Batterywala *et al.*, "Track assignment: a desirable intermediate step between global routing and detailed routing," in *Proc. ICCAD*, pp 59-66, 2002.

[8] 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. ISPD*, pp 61-68, 2000.

[9] D. Wu *et al.*, "Timing driven track routing considering coupling capacitance," in *Proc. ASP-DAC*, pp 1156-1159, 2005.

[10] Y.-N. Chang *et al.*, "Non-slicing floorplanning based crosstalk reduction on gridless track assignment for a gridless routing system with fast pseudo-tile extraction," in *Proc. ISPD*, pp. 134-141, 2008.

[11] M. Cho *et al.*, "Track routing and optimization for yield," *IEEE TCAD*, 27(5), pp. 872 – 882, 2008.

[12] X. Gao and L. Macchiarulo, "Track routing optimizing timing and yield," in *Proc. ASP-DAC*, pp 627 – 632, 2011.

[13] Y.-W. Lee *et al.*, "Minimizing critical area on gridless wire ordering, sizing and spacing," *Journal of Information Science and Engineering*, pp. 157 – 177, 2014.

[14] B.-T. Lai *et al.*, "Native-conflict-avoiding track routing for double patterning technology," in *Proc. SOCC*, pp. 381 – 386, 2012.

[15] D. J.-H. Huang and A. B. Kahng, "Partitioning-based standard-cell global placement with an exact objective," in *Proc. of ISPD*, pp 18-25, 1997.

[16] W.-H. Liu *et al.*, "NCTU-GR 2.0: multithreaded collision-aware global routing with bounded-length maze routing," *IEEE TCAD*, 32(5), pp. 709-722, 2013.

[17] M.-K. Hsu *et al.*, "Routability-driven placement for hierarchical mixed-size circuit designs," in *Proc. DAC*, 2013.

[18] N. Viswanathan *et al.*, "The DAC 2012 routability-driven placement contest and benchmark suite," in *Proc. DAC*, 2012.

[19] J. Hu *et al.*, "Taming the complexity of coordinated place and route," in *Proc. DAC*, 2013.

TABLE. 2. COMPARISON OF WIRELENGTH COST (WL), OVERLAP COST (OC), AND BLOCKAGE COST (BC) BETWEEN WBM-TA AND NTA.

Circuit	WBM-TA				NTA			
	WL (10^6)	OC (10^6)	BC (10^6)	time (s)	WL (10^6)	OC (10^6)	BC (10^6)	time (s)
sb2	29.15	24.29	0.34	543.19	26.88	5.55	0.34	217.79
sb3	29.02	11.20	0.33	549.59	25.37	3.91	0.33	206.36
sb6	29.73	11.24	0.23	517.28	26.79	3.28	0.23	205.85
sb7	43.98	13.33	0.21	758.54	40.39	3.06	0.21	305.96
sb9	24.63	7.72	0.15	327.80	22.68	1.96	0.15	182.00
sb11	26.06	8.06	0.22	353.90	23.79	1.62	0.22	128.16
sb12	41.06	12.88	0.04	591.56	39.13	1.63	0.04	314.61
sb14	18.31	6.57	0.26	183.85	16.27	1.65	0.26	92.12
sb16	19.57	8.18	0.14	219.49	17.68	2.52	0.14	81.81
sb19	14.64	4.38	0.09	162.74	13.34	1.01	0.09	157.70
Ratio	1	1	1	1	0.911	0.247	1	0.448