BOB-Router: A New Buffering-Aware Global Router with Over-the-Block Routing Resources Optimization

Yilin Zhang[1], Salim Chowdhury[2] and David Z. Pan[1]
[1] Department of ECE, University of Texas at Austin, Austin, TX, USA
[2] Oracle, Austin, TX, USA

***Abstract*—In this paper, we propose a new global router, BOB-Router, endowed with the ability to use over-the-block routing resources to the greatest extent in addition to traditional routing concepts of minimizing wirelength, via count and overflow. In previous global routing formulations, the routing resources over the IP blocks were either dealt as routing blockages leading to a significant waste, or simply treated in the same way as outside-the-block routing resources, which violates the slew constraints and thus fail buffering. Utilizing over-the-block routing resources could dramatically improve the routing solution, yet requires special attention, since the slew, affected by different RC on different metal layers, must be constrained by buffering and is easily violated. Moreover, even all nets are slew-legalized, the routing solution could still suffer from heavy congestion problem. For the first time, BOB-Router tries to solve the over-the-block global routing problem through minimizing overflows, wirelength and via count simultaneously without violating slew constraints. BOB-Router generates a slew-legalized initial solution followed by a Lagrangian-multiplier-based pricing phase and RC-constrained A* search to help explore new buffering-aware topologies on all metal layers. Our experimental results show that BOB-Router completely satisfies the slew constraints and significantly outperforms the obstacle-avoiding global routers in terms of wirelength, via count and overflows.**

I. Introduction

As semiconductor technology keeps scaling into deeper sub-micron domain, interconnect delay becomes more critical in determining chip performance. Routing is one of the most important stages regarding performance of chip interconnection. The CEDA-sponsored ISPD Global Routing Contests [2] and [3] attract attention from dozens of academic and industrial participants. Inspired by the competitions, many high-performance global routers are published, including but not limited to, FastRoute 3.0 [22], FastRoute 4.0 [20], BoxRouter 2.0 [7], NTUgr [6], NTHU-Route [10], NTHU-Route 2.0 [5], GRIP [19], FGR [18], MaizeRouter [16], Archer [17] and NCTU-GR [9].

Those global routers can be roughly divided into two categories: sequential and concurrent algorithms. Sequential works [22], [20], [6], [10], [5], [16], [17], [9] route the nets based on heuristic rip-up and reroute (RNR) techniques, which tend to run 2D global routing followed by layer assignment. On the other hand, works such as [19] and [18] directly address the problem by running a full 3D global routing.

Meanwhile, extensively using IP-blocks to shorten turnaround time nowadays packs SOC designs with IP blocks or macros. To avoid routing over those blocks, obstacle-avoiding rectilinear Steiner minimum tree (OA-RSMT) problem has been actively studied over the years (e.g. [4], [12]–[14]). However, completely avoiding those routing areas will result in significant underutilization of high-level metal layers which is the key to save power and close timing. To tackle that issue, new ideas of intelligently utilizing part of, instead of completely avoiding, the over-the-block routing resources with buffering awareness are proposed by [21] and [11] as BOB-RSMT [21] problem, as well as studied as scenic constraints in [15].

Since the guidance from two ISPD Global Routing Contests are similar, most published modern routers aim at the same problem: minimizing wirelength and via count in addition to alleviating congestion. However, the global routing problem has never been touched upon to not only consider wirelength, vias and overflows, but also properly use over-the-block routing resources. Studying this new problem is essential as to shorten the design cycle and improve the chip quality. If over-the-block routing resources are treated the same as that for outside-the-block, long nets over the block will fail buffering, leading to additional manual work; whereas if over-the-block routing resources are totally avoided, less remaining routing resources will significantly deteriorate the quality of the routing solution.

Our key contributions include:

1) We study the over-the-block global routing problem for the first time, providing global routing solution with overflows, wirelength, via count and buffering-awareness considered simultaneously.
2) We improve BOB-RSMT algorithm [21] by addressing its two limitations. Then we apply modified BOB-RSMT algorithm for our initial legal inside-tree generation.
3) For any block with overflow, in each iteration we evolve new topologies from inside trees confined within that block, with less cost associated with congestion, wirelength and via count,
4) We conduct Lagrangian-multipliers-based cost function to reflect the weighted impact from all generated topologies. It turns out that topologies with less cost will have more impact on determining the cost of covered edges.
5) An RC-constrained A* search is proposed to help incrementally evolve new topologies with minimum cost while meeting slew constraints.

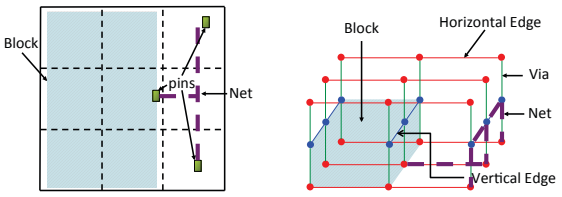
The rest of paper is organized as follow. We first introduce basic concepts of inside trees, slew model and problem formulation in Section II. Our over-the-block routing algorithm will be presented in Section III, which includes three subsections.

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

Section III-A discusses how to modify BOB-RSMT to generate initial legal inside topologies. Section III-B illustrates the process of incrementally evolving new topologies according to Lagrangian-multiplier-based cost function and RC-constrained A* search. Experimental results are shown and analyzed in Section IV, followed by conclusions in Section V.

II. Preliminaries

A. Basic over-the-block concepts



(a) (b)

Fig. 1. 3D grid-graph G of three metal layers with each one divided into 3*3 global routing bins

In global routing, the chip is partitioned into rectangular global routing bins where a 3D grid-graph $G = (V, E)$ is used to model the multi-layer design. As depicted in Fig. 1(a) and Fig. 1(b), each global routing bin is a vertex $v \in V$. The boundary between two adjacent global routing bins on the same layer is modeled as an edge $e \in E$ with a capacity c_e reflecting the maximum routing resources between the cells.

After placement, the chip is packed with IP blocks or macros which occupy the low metal layers and forbid buffer insertion over the IP blocks. In our formulation, we set $B = \{b_1, b_2, \ldots, b_m\}$ as the set of blocks. Each block is modeled by a box in the 3D grid-graph G as the shadowed part in Fig. 1.

A set of multi-terminal nets $N = \{n_1, n_2, \ldots, n_k\}$ is required to be connected in the 3D graph G. The tree topology of each net n_i will enter and leave the blocks in the graph, which divides the whole tree topology into a set of outside trees TO_i and a set of inside trees TI_i.

For any inside tree t, the leaf nodes of t are on the boundaries of a block. Among all leaf nodes, one must be driving the signal and others are receiving. We name these leaf nodes that receive signals *escaping points* (EP), and the set of escaping points for t is $EP^t = \{EP_1^t, EP_2^t, \ldots, EP_{|EP^t|}^t\}$.

We use the same model as in [21] to check if any inside tree satisfies slew constraints. In our formulation, every inside tree is forced to be legal (i.e. satisfy slew constraints).

B. Problem Formulation

Matrices including wirelength, via cost and total overflow (TOF) are used to evaluate our routing solution. TOF is preferred to be zero since slightly overflowed global routing can still make detail routing considerably more difficult.

Our proposed buffering-aware global router will connect each net in N with the target of minimizing total wirelength in addition to reducing TOF. Over-the-block trees have to satisfy the slew constraints which ensure that every topology has feasible buffering solutions.

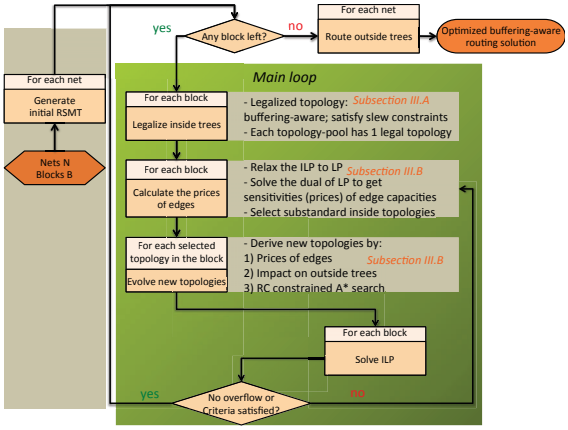


Fig. 2. Overall flow of BOB-Router

III. BOB-Router Algorithms

The overall flow of BOB-Router approach is depicted in Fig.2. The procedures in the "Main loop" frame consists of routing algorithm for inside trees while the rest part is composed of initial legal RSMT generation along with routing for outside trees.

In the BOB-Router problem formulation, any inside tree has to satisfy slew constraints to accomodate buffering. Due to this extra requirement, routing for inside trees becomes more challenging than that for outside trees. Our BOB-Router will route inside trees ahead of outside trees, as we algorithmically emphasize on inside-tree routing which prefers topologies with least downside or even betterment on the cost of outside-tree routing.

To avoid simultaneously coping with wirelength, via count, overflow and slew constraints in inside-tree-routing problem, we decouple the slew constraints by legalizing all topologies first and making sure every following step during the entire inside-tree routing will not violate the slew constraints. This decoupling process includes two steps. First, since the initial inside trees could violate slew constraints, we apply an EP-movement-based legalization procedure modified from [21] to legalize any illegal inside topology with minimum wirelength penalty. Second, during the "evolve new topologies" step (shown in Fig.2) in the inside-tree routing, we use an RC-constrained A* search to ensure that each operation during new topology evolution will not break the slew constraints.

A. Generate Legal Initial Topologies

We modify BOB-RSMT algorithm in [21] to generate legal initial inside trees. We will first briefly review BOB-RSMT approach, and then study two insufficiencies of BOB-RSMT. Lastly, we will present our modified BOB-RSMT (BOB-RSMT-m) to provide upgraded initial legal inside trees.

In BOB-RSMT, an initial RSMT is generated for each net by FLUTE [8]. Yet for any generated RSMT, it might violate

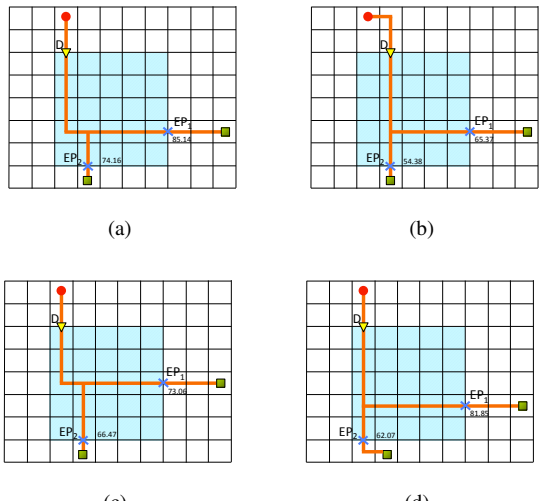


Fig. 3. Best move selection (a) shows an illegal inside tree. (b), (c) and (d) exhibit and evaluate the best single-unit move from the driver, EP_1 and EP_2 respectively.

slew constraints and become illegal. First of all, in order to legalize any illegal inside tree t in the illegal RSMT, all illegal escaping points (EP) of t will be sorted according to their slew violations. Next, in every iteration the first illegal escaping point EP_1^t with the worst slew violation based on sorting will be legalized. To legalize slew for EP_1^t, each escaping point from $\{EP_1^t, EP_2^t, \ldots EP_{|EP^t|}^t\}$ may slide to a different position by taking a combination of primitives with inside Steiner nodes updated as well. These primitives include parallel sliding which shifts the connected branch, perpendicular sliding which stretches the connected branch and EP merging with one of the neighboring EP. ILP is applied to select the final location for each EP from its *possible point set*. Finally, the location of EP_1^t will be fixed and next iteration is launched to legalize the next illegal escaping point EP_2^t. This process will be iterated until all escaping points satisfy slew constraints.

BOB-RSMT approach efficiently generates a topology satisfying slew constraints, however, it has two limitations. First, movement of the driver for an inside tree is not considered. Second, when two branches at the opposite end of the driver move simultaneously, slew improvement may be underestimated.

To address those two limitations, we keep the optimization primitives but replace ILP with a greedy approach. Instead of evaluating each possible point and applying ILP to selection, we assess every *single-unit move* from all EPs and the driver, and select the best move. For example Fig.3(a) presents an illegal inside tree, while Fig.3(b), Fig.3(c) and Fig.3(d) give the resulted topologies assuming the best moves from the driver, EP_1 and EP_2 are selected respectively. In order to directly show amount of slew improvement in Fig.3, we set one for unit R and unit C, along with zero for buffer-output resistance, buffer-input capacitance and output slew in our slew model. As we can see, the best slew improvement occurs in Fig.3(b), where the driver reduces worst slew by 19.88 slew units with single-unit move. However, in BOB-RSMT, this solution cannot be found since the move of the driver is not considered.

The other benefit from proposed BOB-RSMT-m is that it accurately catches the slew difference when multiple branches at the opposite end of the driver moving simultaneously. In this rare case, BOB-RSMT approach might disregard slew improvement from antenna clearance and overestimate slew improvement from branch sliding. As is shown in Fig.4, moving EP_1 to the right by one unit (Fig.4(b)) can improve the worst slew for 9.89 slew units while moving EP_2 in the same way (Fig.4(c)) will improve the worst slew for 3.30 slew units. If the ILP in BOB-RSMT is applied to choose both slides, the total improvement will be summed up to 13.19 slew units in an incorrect way. The actual improvement in Fig.4(e) is 11.98 slew units which consists of the slew improvement from moving EP_1 to the right by one unit and removing of antenna segment circled in 4(d).

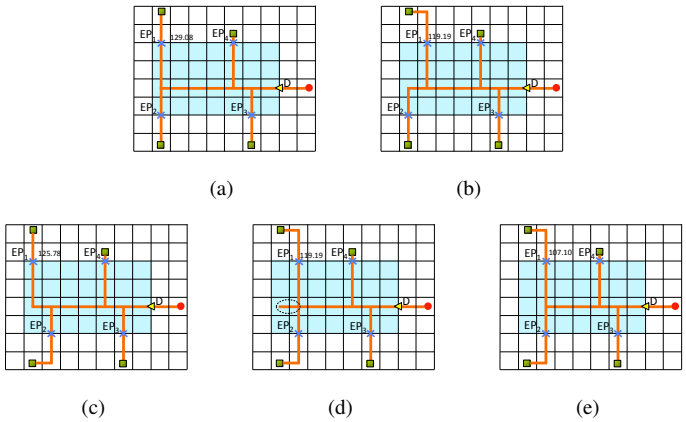


Fig. 4. Slew calculation method in BOB-RSMT and BOB-RSMT-m. (a) shows an illegal inside tree. (b) and (c) exhibit and evaluate the best single-unit move from EP_1 and EP_2 respectively. (d) and (e) illustrates if both EP_1 and EP_2 move.

B. Evolving Legal Congestion-Aware Min-Cost Topologies

The initial-inside-tree legalization guarantees one legalized topology for each inside tree. Placing these legal topologies simultaneously within each associated block could cause congestion problem. To resolve this issue, our approach uses the generated legalized topologies as seeds, giving birth to more legal congestion-aware topologies with less cost than current topologies. Finally, one topology will be chosen for each inside tree to achieve least overflow and cost.

We keep our topologies in Steiner tree structures instead of decomposing them into 2-pin nets in that (1) Steiner tree structures have more flexibility with unfixed Steiner points while 2-pin nets have to connect specified end points; (2) Steiner tree structure allows for tracking non-linear slew calculation over the entire tree, which is improbable for decomposed 2-pin nets. In normal global routing problem, it is non-trivial regarding how to come up with congestion-aware Steiner tree topologies with minimum cost. However, the Steiner tree topologies we demand for our inside trees have to satisfy additional slew constraints.

1) Formulations: First, we build an ILP formulation to describe the routing problem in each block. The ILP formulation contains no slew constraints, as every topology presented in the ILP formulation is legal.

$$\text{min.} \sum_{i=1}^{n} \sum_{t \in \zeta_i} X_{it} W_{it} + M \sum_{i=1}^{n} S_i \tag{1}$$

$$\text{s.t. } \forall i, S_i + \sum_{t \in \zeta_i} X_{it} = 1 \tag{1a}$$

$$\sum_{i=1}^{n} \sum_{t \in \zeta_i} X_{ite} <= C_e \quad \forall e \prec b, b \; is \; block \tag{1b}$$

$$X_{it} \in \{0,1\} \qquad \forall i \in \{1,2,\ldots,n\} \qquad \forall t \in \zeta_i$$

$$S_i \in \{0,1\} \qquad \forall i \in \{1,2,\ldots,n\}$$

$TI = \{T_1, T_2, \ldots, T_n\}$ is the set of inside trees within block b, and ζ_i in the formulation is the collection of all topologies for T_i. For each Steiner tree topology $t \in \zeta_i$, parameter W_{it} represents the overall cost of the topology, including both wirelength and vias. Variable S_i denotes the routability of T_i; if S_i is positive, the inside tree T_i cannot be routed with available Steiner tree topologies. $\sum_{i=1}^{n} \sum_{t \in \zeta_i} X_{ite}$ contains the amount of routing resources demanded on every edge e, which is required to maintain under the edge capacity vector C_e.

In order to minimize overflow, MS_i is used in the objective function to penalize any unroutable inside tree T_i. Parameter M is a predefined large number which is greater than the wirelength of any possible Steiner tree in the chip. Solving ILP formulation (1) guarantees no inside-overflow and minimizes total cost with maximum number of routed inside trees.

The ILP formulation has the following two purposes in our approach: i) to select one topology for each inside tree to check if overflow-free solution could be achieved at the end of each iteration, ii) the dual problem of relaxed LP could provide cost of each edge.

2) Pricing the Edges: Before solving the ILP and fixing the topologies in current iteration, more Steiner tree topologies, instead of the initial one solely, are wanted to effectuate least TOF and cost routing solution. We use sensitivity analysis on the edge capacity constraints to price each edge, which provides a guidance for the evolution of new Steiner tree topologies from current topologies.

Different edges on different layers have various values in the routing since some of them are in congested area while some are not. We calculate *price* to describe the potential overflow on each edge. To obtain the prices for edges, we first relax the ILP formulation (1) into an LP formulation by relaxing binary variables $\{X_{ij}\}$.

The relaxation on binary variables $\{X_{ij}\}$ splits the constraint of choosing only one topology for each inside tree into a set of fractional numbers indicating several potentially preferred topologies. A topology avoiding congested area and costing less wirelength and vias will be preferred and assigned positive X_{ij} which depends on the quality of the topology.

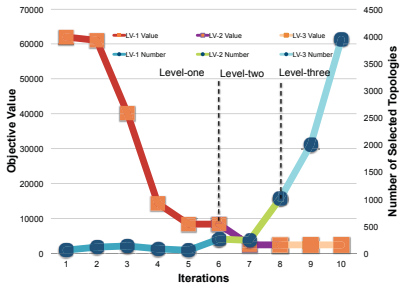


Fig. 5. Progression of objective value and number of selected "to-be-evolved" topologies over optimization rounds for one block on ADAPTEC1

The price of each edge comes from the dual of this LP formulation exhibited in (2). The variable λ_i is the Lagrange multiplier associated with relaxed topology-selection constraint (1a) for T_i and ρ_e is the Lagrange multiplier associated with relaxed capacity constraint (1b) for edge e. According to complementary slackness theorem, for optimal primal variables $X_i^*, i \in \{1,2,\ldots,n\}$ and optimal dual variable ρ_e^*, there exists $\rho_e^* * (\sum_{i=1}^{n} \sum_{t \in \zeta_i} X_{ite} - C_e) = 0$. When the ρ_e^* is positive, $\sum_{i=1}^{n} \sum_{t \in \zeta_i} X_{ite} - C_e = 0$ will be true, which means corresponding edge e has no "leftovers" in capacity. If the primal optimal solution exists, according to strong duality, the optimal dual variable ρ_e^* reflects how much improvement on the objective value we can make if the capacity of edge e increases by one. Therefore, we use the optimal dual variable ρ_e^* as the price for edge e. Compared with history-based cost in other routers, our price is more comprehensive because it considers all topologies we have and weights them according to their worth optimally.

$$\text{max.} \sum_{i=1}^{n} (-\lambda_i) + \sum_{e \prec b} (-\rho_e) c_e \tag{2}$$

$$\text{s.t. } \lambda_i + \sum_{e \prec t} \rho_e + W_{it} >= 0 \tag{2a}$$

$$\lambda_i >= -M \qquad \forall i \in \{1,2,\ldots,n\}$$

$$\rho_e >= 0 \qquad \forall e \prec b$$

3) Three-level Topology Selection: If we evolve new topologies for each existing topology, the size of our topology pools will dramatically increase without corresponding speed of TOF mitigation. Therefore, we control the number of new evolved topologies in each iteration by only considering the most costly topologies.

We use a dynamic three-level topology-selection approach to determine certain topologies for evolution in each iteration. Only if current stage fails to further improve TOF, will next stage be launched. The following level selects topologies in a more broad way which enables further TOF reduction.

- Level-one: After we find all inside trees with positive S_i, all topologies associated with these unroutable inside trees will be evolved.
- Level-two: If evolution of topologies from level-one is unable to keep optimizing the LP formulation, we as-

semble an inside-tree-routing solution by selecting the topology with largest X_{ij} for each inside tree. Then the overflow of each edge could be counted. In addition to the topologies from level-one, any topology containing overflowed edge(s) will be added.

- Level-three: If the topology evolution in level-two fails to keep optimizing the LP formulation, we evolve topologies covering edges with positive price in addition.

We gradually loosen our requirement for topology evolution, pushing the optimization with control over the number of processed topologies. Fig.5 evaluates the three-level topology-selection during optimization iterations for one single block on ADAPTEC1. It shows that the first iterations in level-one increases the size of topologies slowly. As optimization halts during any iteration, next level will be launched to reduce TOF.

4) RC-constrained A Search:* After pricing and topology selection in every optimization iteration, we evolve new topologies with slew-aware rip-up and reroute. The pricy part will be ripped up and an RC-constrained A* search algorithm is applied to reroute disconnected parts without violating the slew constraints.

For any selected topology t, we find all wires with non-zero price, and sort them by their prices in descending order. After sorting, we sequentially rip-up and reroute each wire. For one wire w on t, signaling from U to V, we remove w from t first and notate the remaining part as $t\backslash w$. Then we calculate RC_p and C_p for each point $p \in t\backslash w$. RC_p and C_p are the maximum allowed RC and C connected to p without violation to the slew constraints. The maximum possible RC and C for all points on $t\backslash w$ will be:

$$RC^{max} = \max\{RC_p, p \in \{t\backslash w\}\} \quad (3)$$

$$C^{max} = \max\{C_p, p \in \{t\backslash w\}\} \quad (4)$$

Afterwards, an RC-constrained A* search is applied to reconnect V to the remaining part $t\backslash w$. We will only accept connections to point p with RC and C less than RC_p and C_p respectively. During RC-constrained A* search, any search path with RC exceeding RC^{max} or C exceeding C^{max} will be pruned away. The cost of each edge e in our A* search is the price of e plus one. The heuristic cost function we use is the 3-D Manhattan distance to the nearest point in $t\backslash w$, which clearly is a lower bound. This RC-constrained A* search guarantees least cost solution under slew constraints.

C. Outside-tree Routing

After topologies of inside trees are fixed, capacities of all edges within blocks are set to zero before blockage-avoiding outside-tree routing, which will be solved by existing academic routers.

IV. Experimental Results

BOB-Router has been implemented in the C++ programming language. All experiments are conducted on an Intel Core 3.0GHz Linux machine with 16GB memory. We use 3D global routing benchmarks adaptec1 $\sim$ 4 and bigblue1 $\sim$ 4 from ISPD 2007 and 2008 Global Routing Contests for our experiments. Benchmarks from global routing contests are not annotated with blockage information explicitly. As far as we know, the block porosity information in the global routing benchmarks are derived from fixed macros in certain placement benchmarks. Owing to abutting blocks, it is arduous to directly retrieve geometric information of porosity areas from the routing benchmarks. Instead, we find the corresponding placement benchmarks, from which we are able to extract fixed macro geometric information. Besides, we remove nets containing pins inside blocks, which is beyond our formulation.

The wire resistance and capacitance for each metal layer are derived from ITRS [1], and we use 70ps as our maximal allowed slew.

We first evaluate the slew violation for each benchmark. Table II calibrates the slew numbers for all inside trees after RSMT topologies are generated by FLUTE and applied with a simple-layer-assignment heuristic. The heuristic will assign all inside trees on the lowest allowable pair of layers first. Then for all inside trees with slew violation, we will bring them to higher pair of metal layers according to extents of slew violations. From Table II, we can see that some benchmarks with no slew problem initially, such as bigblue2, may encounter slew problem because it is possible that most of inside trees have been promoted to the highest pair of metal layers.

TABLE II
SLEW DISTRIBUTION OF INSIDE TREES

Benchmarks	# nets	# inside trees	max slew	average slew
adaptec1	219794	57852	1713.8	36.9
adaptec2	260159	34769	494.4	28.5
adaptec3	466295	105137	23785.5	141.6
adaptec4	515304	86199	3986.7	65.8
bigblue1	282974	18763	380.1	22.1
bigblue2	576816	117259	69.9	4.0
bigblue3	1122340	79659	2025.1	22.1
bigblue4	2228930	234692	631.1	5.0

Since we eliminate all slew violation during initialization and keep slew under constraints, our final routing solution will not suffer from any slew problem. In Fig.6, we compare the slew distribution of inside trees from Table II with final routing solution for benchmark adpatec1. Initially, we observe the existence of inside trees with worst slew up to 1714ps. But after the benchmark is processed by BOB-Router, no inside tree has slew more than 70ps which is the maximum allowed slew rate in our slew constraints. The number of inside trees with slew between 60ps to 70ps is dramatically increased as most nets with slew violations originally are legalized to be just under 70ps.

If one router is not able to properly use the over-the-block routing resources, the safest way without breaking slew constraints thus involving manual work is to avoid the blocks completely by setting over-the-block routing capacity as zero (or large penalty). We compare our proposed BOB-Router with an obstacle-avoiding router (OA-Router) in terms of wirelength, via count and TOF. We modify NTHU-Router 2.0 [5] to be the OA-Router and the solver for outside-the-

TABLE I
COMPARISONS BETWEEN OUR PROPOSED BOB-ROUTER AND OA-ROUTER

Bench	over-the-block				outside-the-block				overall				OA-Router			
-marks	WL	Vias	TOF	cpu(s)	WL	Vias	TOF	cpu(s)	WL	Vias	TOF	cpu(s)	WL	Vias	TOF	cpu(s)
adaptec1	431886	138207	0	5690	2733837	1344218	199565	1421	3165723	1482425	199565	7111	3317320	1724765	450300	3463
adaptec2	261957	57838	265	4523	2615068	1258131	28847	1038	2877025	1315969	29112	5561	3371453	1836853	107498	4577
adaptec3	1235721	154123	1333	100210	8355049	2849048	639049	16527	9590770	3003171	640382	116737	10100613	3740726	1276779	18845
adaptec4	836840	105953	0	32718	8831370	2580484	329221	13202	9668210	2686437	329221	45920	11326871	3498262	438954	13455
bigblue1	98044	42090	0	55	3248498	1367350	22612	1637	3346542	1409440	22612	1692	3637249	1967568	70853	2232
bigblue2	258699	350385	0	520	3730497	2985365	3795	1131	3989196	3335750	3795	1651	4799773	3800398	5145	1346
bigblue3	522841	141885	0	2119	7800699	3847139	15148	2621	8323540	3989024	15148	4740	8961863	5267470	83416	8603
bigblue4	575639	731836	0	303	9358521	7489968	5266	2266	9934160	8221804	5266	2569	12363167	10444398	27939	5784
average	0.08	0.06	0.00	0.51	0.92	0.94	1.00	0.49	1.00	1.00	1.00	1.00	1.13	1.28	3.07	1.00

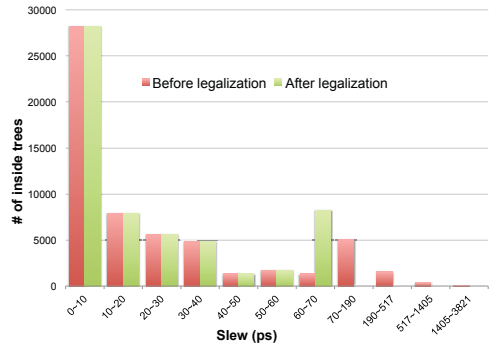


Fig. 6. Slew distribution of all inside trees in adpatec1 initially and finally. Each y coordinates number of inside trees with slew in the slot between current and previous x

block routing for its good performance. The results are shown in Table I. From the last row in the table, we can see that BOB-Router pushes about 8% of wirelength and 6% via count to the over-the-block part on average. The TOF of over-the-block routing is zero for most benchmarks. By using over-the-block routing resources, BOB-Router achieves about only 33% TOF, 88% wirelength and 78% via count of the OA-Router. We think more decrease of via count than wirelength is partially because BOB-Router performs full 3D routing for over-the-block part without layer assignment. Average runtime of BOB-Router is same with OA-Router. However, we notice that BOB-Router spends more time on bigger and tougher benchmarks, such as adaptec3, because more iterations and topologies are required.

V. CONCLUSION

In the past few years, traditional global routing has been extensively studied, which in turn makes it hard even to improve performance by 1%. We propose a new formulation of global routing problem from a different perspective. Solving this new BOB-Routing problem could keep shortening design cycle and improving routing quality. With our proposed approach, we can generate slew-violation-free solution with 66% less TOF, 12% less wirelength and 22% less via count compared with the obstacle-avoiding approach.

VI. ACKNOWLEDGMENT

This work is supported in part by NSF, SRC and Oracle.

REFERENCES

[1] 2012 Overall Roadmap Technology Characteristics (ORTC) Tables.
[2] ISPD 2007 Global Routing Contest and Benchmark Suite. http://archive.sigda.org/ispd2007/contest.html.
[3] ISPD 2008 Global Routing Contest and Benchmark Suite. http://archive.sigda.org/ispd2008/contests/ispd08rc.html.
[4] G. Ajwani, C. Chu, and W. Mak. FOARS: FLUTE Based Obstacle-Avoiding Rectilinear Steiner Tree Construction. In *Proc. ISPD*, pages 194–204, 2010.
[5] Y. Chang, Y. Lee, J. Gao, W. Wu, and T. Wang. NTHU-Route 2.0: A Robust Global Router for Modern Designs. In *IEEE TCAD*, volume 29(12), pages 1931–1944, 2010.
[6] H. Chen, C. Hsu, and Y. Chang. High-Performance Global Routing with Fast Overflow Reduction. In *Proc. ASPDAC*, pages 582–587, 2009.
[7] M. Cho, K. Lu, K. Yuan, and D. Z. Pan. BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router. In *Proc. ICCAD*, pages 503–508, 2007.
[8] C. Chu and Y. Wong. FLUTE: Fast Loopup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. *IEEE TCAD*, 27(1):70–83, 2008.
[9] K. Dai, W. Liu, and Y. Li. Efficient Simulated Evolution Based Rerouting and Congestion-Relaxed Layer Assignment on 3-D Global Routing. In *Proc. ASPDAC*, pages 570–575, 2009.
[10] J. Gao, P. Wu, and T. Wang. A New Global Router for Modern Designs. In *Proc. ASPDAC*, pages 232–237, 2008.
[11] T. Huang and E. F.Y. Young. Construction of rectilinear Steiner minimum trees with slew constraints over obstacles. In *Proc. ICCAD*, pages 144–151, 2012.
[12] T. Huang and Evangeline F.Y. Young. An Exact Algorithm for the construction of Rectilinear Steiner Minimum Trees among Complex Obstacles. In *Proc. DAC*, pages 164–169, 2011.
[13] L. Li, Z. Qian, and Evangeline F.Y. Young. Generation of Optimal Obstacle-avoiding Rectilinear Steiner Minimum Tree. In *Proc. ICCAD*, pages 21–25, 2009.
[14] L. Li and Evangeline F.Y. Young. Obstacle-avoiding Rectilinear Steiner Tree Construction. In *Proc. ICCAD*, pages 523–528, 2008.
[15] W. Liu, Y. Wei, C. N. Sze, C. J. Alpert, Z. Li, Y. Li, and N. Viswanathan. Routing Congestion Estimation with Real Design Constraints. In *Proc. DAC*, 2013.
[16] M. D. Moffitt. MaizeRouter: engineering an effective global router. In *Proc. ASPDAC*, pages 226–231, 2008.
[17] M. Mustafa Ozdal and M. D. F. Wong. Archer: a history-driven global routing algorithm. In *Proc. ICCAD*, pages 488–495, 2007.
[18] J. A. Roy and I. L. Markov. High-Performance Routing at the Nanometer Scale. In *IEEE TCAD*, volume 27(6), pages 1066–1077, 2008.
[19] T. Wu, A. Davoodi, and J. T. Linderoth. GRIP: Scalable 3D Global Routing Using Integer Programming. In *Proc. DAC*, pages 320–325, 2009.
[20] Y. Xu, Y. Zhang, and C. Chu. FastRoute 4.0: Global Router with Efficient Via Minimization. In *Proc. ASPDAC*, pages 576–581, 2009.
[21] Y. Zhang, A. Chakraborty, S. Chowdhury, and D. Z. Pan. Reclaiming Over-the-IP-Block Routing Resources With Buffering-Aware Rectilinear Steiner Minimum Tree Construction. In *Proc. ICCAD*, pages 137–143, 2012.
[22] Y. Zhang, Y. Xu, and C. Chu. FastRoute3.0: A Fast and High Quality Global Router Based on Virtual Capacity. In *Proc. ICCAD*, pages 344–349, 2008.