Simultaneous Timing Driven Tree Surgery in Routing with Machine Learning-based Acceleration

Peishan Tu, Chak-Wa Pui, Evangeline F. Y. Young
Department of Computer Science and Engineering
The Chinese University of Hong Kong, NT, Hong Kong
pstu,cwpui,fyyoung@cse.cuhk.edu.hk

ABSTRACT

In global routing, both timing and routability are critical criterions to measure the performance of a design. However, these two objectives naturally conflict with each other during routing. In this paper, a tree surgery technique is presented to adjust routing tree topologies in global routing to fix timing. We formulate the problem as a quadratic program (QP), which adjusts routing topologies of all the nets from a global perspective and takes congestion into consideration to trade off timing and routability objectives. We also apply machine learning-based techniques to accelerate our algorithm, which offers a fast and effective way to solve the problem. Experimental results on ICCAD 2015 benchmarks show that our algorithms can achieve 10.12% timing improvement with no significant degradation in routability and wirelength. With machine learning-based acceleration (MLA), our results can be obtained in almost negligible runtime.

CCS CONCEPTS

• **Hardware → Wire routing**;

ACM Reference Format:
Peishan Tu, Chak-Wa Pui, Evangeline F. Y. Young. 2018. Simultaneous Timing Driven Tree Surgery in Routing with Machine Learning-based Acceleration. In *GLSVLSI '18: 2018 Great Lakes Symposium on VLSI, May 23–25, 2018, Chicago, IL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3194554.3194556

1 INTRODUCTION

As design complexity increases, achieving timing closure becomes even more challenging. Modern designs demand improved routing strategies to meet timing requirements [12].

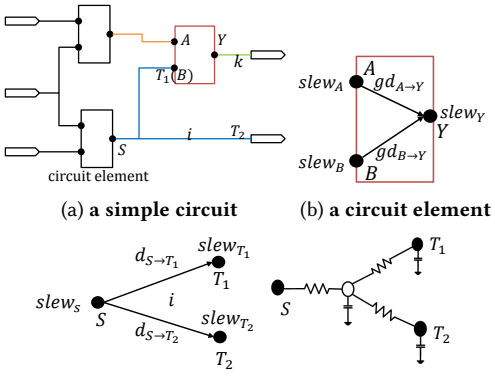
Since global routing plays an important role in both the placement and routing phases, there are numerous previous works on global routing. NCTUgr [15] began with rectilinear minimum spanning tree (RMST) topologies and utilized the rectilinear steiner minimum tree (RSMT) topologies to guide the following monotonic routing and negotiation-based rip-up and reroute. NTHURoute [2] also decomposed nets to two-pin nets based on routing tree topologies. Next, it utilized a rip-up and reroute approach, based on the congested region identification, to further improve routability. FastRoute [18] first built routing trees for all the nets and then adjusted the routing edges to reduce congestion. It then performed multi-source multi-pin maze routing and 3-bend routing with an adaptive cost function. MaizeRouter [17] initialized tree construction by FLUTE [6], which is an approach to build RSMT. It then shifted and retracted the edge to optimize congestion. Followed by layer assignment, the maze routing was performed. Overall, these global routers achieved good performance in terms of wirelength and routing congestion. However, timing is not considered in these routers.

Besides congestion driven routers mentioned above, some works optimized timing and congestion together. The work [7] proposed a global routing approach to incorporate both congestion and timing optimization. However, it took tens of seconds to handle at most 400 nets, which may not be fast enough for current designs. In addition, it targeted at small number of module nets instead of large number of standard cell nets. The unified timing and congestion optimizing (UTACO) [11] algorithm adopted a shadow price mechanism which considered timing and congestion as the sum of the price. It first built the minimum wire length steiner tree and performed rip-up and reroute by optimizing the price of congestion and timing. However, it modeled the delay of each net individually which may neglect interaction among adjacent nets because actual gate delay is affected by values from upstream and downstream nets. The work [5] targeted at optimizing Chemical-Mechanical Polishing(CMP) and timing in global routing besides congestion. It modeled timing by a guide of wire density. However, the gate model it considered is the lumped resistance model. The work [19] proposed a routing algorithm that considered timing optimization, buffer insertion and power reduction. It first constructed minimum steiner trees and additional detoured trees and buffered trees are then built to reduce congestion and timing. Next, it formulated an ILP to decrease power consumption. However, the buffer tree construction was time consuming and only wire delay was optimized without gate delay considered.

In most global routers, a tree topology will be assigned to each net, better timing can be achieved by considering timing in construction of the routing tree topologies. Several algorithms [1] [3] are proposed to build timing aware routing trees to achieve good performance on balancing net and gate delay. However, most of them adopt simple lumped resistance model as their gate delay model, which is inaccurate and inadequate for modern designs. Moreover, modern gate delay model requires that tree topologies should not be optimized individually. Hence, in timing aware global

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK14206015).
GLSVLSI '18, May 23–25, 2018, Chicago, IL, USA

ACM ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194556



(a) **a simple circuit** (b) **a circuit element**

(c) **interconnection** (d) **RC model**

Figure 1: Delay in the circuit.

routers, methods are needed to capture delay more accurately and to consider the topologies of all the nets simultaneously.

In this work, we propose an algorithm to adjust the tree topologies of all the nets to fit timing from a global perspective and consider routing congestion simultaneously. Our contributions are summarized as follows:

- To optimize the tree topologies globally, a QP is formulated to determine how to adjust the most critical sink connection to optimize timing and congestion.
- We study various circuit properties and identified those that contribute to timing. Later, these features will be used to accelerate the QP-based tree surgery technique by a machine learning-based technique.
- Experimental results show that we can improve timing of the design significantly with small increase in routing congestion.

The remaining of this paper is organized as follows. Section 2 introduces basic knowledge in timing analysis. Section 3 defines our tree surgery technique (TST) in global route and its formulation. Section 4 contains details of our algorithms, including quadratic programming based TST and machine learning-based acceleration technique. Section 5 shows the experimental results and we finally conclude the work in Section 6.

2 PRELIMINARY

A simple circuit structure is shown in Figure 1(a). It consists of circuit elements, IO pins and interconnections. The circuit elements can be combinational logic elements or sequential elements. When signals travel from the primary inputs to the primary outputs of the circuit, the circuit elements and their interconnections will have delays which affect the performance of the circuit.

2.1 Delay and Slew Calculation

A circuit element shown in Figure 1(b) is extracted from Figure 1(a) which is marked red. It consists of input A, input B and output Y, which is the source gate of net k. Based on a nonlinear delay model (NLDM), gate delay $gd_{A\to Y}$ is estimated based on a 2-dimensional table with inputs slew $slew_A$ and capacitance cap_k. Generally, given specific index values x and y, gate delay can be estimated by Equation (1). Assuming $x_1 < x < x_2$ and $y_1 < y < y_2$, solutions of the bilinear interpolation can be computed as in Equation (1) and the coefficients can be obtained by Equation (2) using z_{11}, z_{12}, z_{21} and z_{22}. Details of the calculation are shown in [8].

$$L(x, y) = a_0 + a_1 \cdot x + a_2 \cdot y + a_3 \cdot x \cdot y \tag{1}$$

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_1 & y_2 & x_1y_2 \\ 1 & x_2 & y_1 & x_2y_1 \\ 1 & x_2 & y_2 & x_2y_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{bmatrix}. \tag{2}$$

Interconnect delay is calculated according to Elmore delay. We make use of the slew calculation as proposed by [8].

2.2 Timing Analysis

Generally, timing analysis is propagated from the primary input to the output to obtain the actual arrival time (aat) and from the output to the input to obtain the required arrival time (rat). We quantify the timing of a circuit at each node by the term slack which is computed by $slack = rat - aat$. Statistic Timing Analysis (STA) [8] is always performed to check the timing of the design.

3 PROBLEM FORMULATION

Given the placement of a design, nets are routed and timing information are obtained by STA. Our objective is to maximize the total negative slack (TNS) by adjusting routing topologies, which is called TST. In our work, TST tries to reconnect critical sinks to maximize TNS by reducing wire delay and gate delay on the critical path, which is formulated as Equation (3).

$$\begin{aligned} \max \quad & \text{TNS}, \\ s.t. \quad & x_i \in \{0, 1\} \quad \forall i \in N_c, \end{aligned} \tag{3}$$

where x_i denotes whether the most critical sink of each net $i \in N_c$ is reconnected and N_c is a set of net such that any net $i \in N_c$ with P_i sinks should satisfy the following constraints:

$$j_x^i = \underset{l \in P_i, \text{slack}_l < 0}{\arg\min} \; slack_l, \tag{4}$$

$$\text{parent}[j_x^i] \neq s_i, \tag{5}$$

where j_x^i is the sink in net i whose slack is the most negative and the parent of j_x^i is not the source s_i. If the slack of every sink in net i is positive, the routing topology of net i will remain unchanged. Equation (5) requires that the sink j_x^i with the worst negative slack is not connected to the root. It may then be possible to connect it to the root to improve the timing.

The explicit formulation of our objective is explained in the following section.

4 TREE SURGERY TECHNIQUE (TST)

TST is an approach to modify the tree structure, such as reconnection. TST is first formulated as a QP to maximize the total negative slack while congestion is also considered in the formulation. We then extract circuit properties which can influence timing and a machine learning based acceleration method is further proposed to speed up the QP-based TST. The notations of variables in this section are listed in Table 1.

4.1 QP-based TST (QPTST)

4.1.1 Timing Optimization. In order to achieve timing closure, STA is used to detect the timing problem of a design. It measures

Table 1: Variable notations in Section 4.1.

s_i	the source of net i
j_x^i	the sink of net i whose slack is negative and the worst
l	the input pin of the gate of s_i which affects actual arrival time of s_i
j	the net of input pin l
$L(cap_i, slew_l)$	gate delay $gd_{l\to S}$ from input l to source node S of net i
$d^o_{s_i\to j_x^i}$	delay from s_i to j_x^i before reconnection on net i
$d_{s_i\to j_x^i}$	delay from s_i to j_x^i after reconnection on net i
$\Delta d_{s_i\to j_x^i}$	delay difference before and after reconnection on net i, $d^o_{s_i\to j_x^i} - d_{s_i\to j_x^i}$
cap_i^o	lumped capacitance of node i before deciding whether performing reconnection on net i
cap_i	lumped capacitance of node i after deciding whether performing reconnection on net i, $cap_i^o - \Delta cap_i x_i$
Δcap_i	capacitance changed when reconnection is performed on net i
$slew_l^o$	slew of pin l before deciding whether performing reconnection on net j
$slew_l$	slew of pin l after deciding whether performing reconnection on net j, $slew_l^o - \Delta slew_l x_j$
$\Delta slew_l$	slew changed when reconnection is performed on net j
ΔL_i	gate delay difference of source s in net i considering reconnection of net i and net j

slack ($slack_{po} = rat_{po} - at_{po}$) at each timing end point $po \in PO$, where PO is a set of primary outputs and register data ports. In STA, timing failure can be detected if the slack of a timing end point is negative ($slack_{po} < 0$). In order to reduce timing failure, our objective is to maximize TNS at critical timing endpoints, which is formulated in Equation (6). PO_n denotes the set of timing end points with negative slack.

$$\max \sum_{po\in PO_n} rat_{po} - aat_{po}. \tag{6}$$

The negative slacks of PO_n is mainly related to the nets with negative slack sinks, which are called critical nets N_c. Hence, in this work, we will improve the slack of critical nets instead of directly optimizing the slack on the primary outputs, which is shown in Equation (7).

$$\max \sum_{i\in N_c} \sum_{l\in P_i^c} rat_l - aat_l, \tag{7}$$

where P_i^c is the set of critical sinks of net i. Since simultaneously optimizing all the critical sinks of a net is hard to achieve and may cause congestion, we further simplify the problem such that only the most critical sink of each critical net will be considered and formulate it as Equation (8).

$$\max \sum_{i\in N_c} rat_{j_x^i} - aat_{j_x^i}. \tag{8}$$

Since the slack value on one critical timing path is the same, for each net, optimizing the slack of the source is equivalent to optimize the slack of the most critical sink. Hence, Equation (8) can be transformed into Equation (9).

$$\begin{aligned} &\max \sum_{i\in N_c} rat_{s_i} - aat_{s_i} \\ =&\max \sum_{i\in N_c} rat_{j_x^i} - d_{s_i\to j_x^i} - aat_l - gd_{l\to s_i} \end{aligned}, \tag{9}$$

where $d_{s_i\to j_x^i}$ is the net delay from s_i to j_x^i, $gd_{l\to s_i}$ is the gate delay and the input pin l of the gate containing node s_i determines the actual arrival time of source s_i. By assuming $rat_{j_x^i}$ and aat_l are constant, we can further simplify the problem as Equation (10).

$$\begin{aligned} &\min \sum_{i\in N_c} d_{s_i\to j_x^i} + gd_{l\to s_i} \\ =&\min \sum_{i\in N_c} d_{s_i\to j_x^i} + L(cap_i, slew_l) \end{aligned}, \tag{10}$$

where gate delay can be represented as $L(cap_i, slew_l)$.

In this work, we minimize delay $d_{s_i\to j_x^i}$ in Equation (10) by reconnecting the critical sink j_x^i directly to its source s_i. However, reconnecting all the nets N_c will increase total capacitance cap_i of each net i due to longer wirelength, which will increase the gate delay $L(cap_i, slew_l)$. In order to maximize the delay reduction $d_{s_i\to j_x^i} + L(cap_i, Slew_l)$, the set of net that will be reconnected is found by Equation (11).

$$\begin{aligned} &\max \sum_{i=1}^{n} (\beta \cdot \Delta L_i + \Delta d_{s_i\to j_x^i} \cdot x_i), \\ &s.t. \quad x_i = \{0, 1\} \quad \forall i \in N_c, \end{aligned} \tag{11}$$

where x_i is a binary variable indicating whether net i is reconstructed. β is a user defined parameter. $\Delta d_{s_i\to j_x^i}$ is the difference of interconnect delay on the path from the critical sink j_x^i to its source s_i before and after reconnecting it to the root, which is computed by Equation (12).

$$\Delta d_{s_i\to j_x^i} = d^o_{s_i\to j_x^i} - d_{s_i\to j_x^i}. \tag{12}$$

ΔL_i implies how much gate delay at node s_i can be reduced, which is computed by Equation (13).

$$\begin{aligned} \Delta L_i =& L(cap_i^o, slew_l^o) - L(cap_i, slew_l) \\ =& a_1 \cdot (cap_i^o - cap_i) + a_2 \cdot (slew_l^o - slew_l), \\ &+ a_3 \cdot (cap_i^o \cdot slew_l^o - cap_i \cdot slew_l) \end{aligned} \tag{13}$$

where $L(cap_i^o, slew_l^o)$ and $L(cap_i, slew_l)$ are the gate delay before and after reconnection respectively. The value of ΔL_i is determined by reconnection of net i and net j, where net j influences the input slew at node l. It is easy to see that ΔL_i can be rewritten in the form of summation of terms with x_i and x_j as in Equation (14), where a_0, a_1, a_2 and a_3 can be obtained as shown in Section 2.1.

$$\begin{aligned} \Delta L_i =& (a_1 + a_3 \cdot slew_l^o) \cdot \Delta cap_i \cdot x_i \\ &+ (a_2 + a_3 \cdot cap_i^o) \cdot \Delta slew_l \cdot x_j \\ &- a_3 \cdot \Delta cap_i \cdot \Delta slew_l \cdot x_i \cdot x_j \end{aligned} \tag{14}$$

With Equation (11) and Equation (14), we can formulate a QP to determine which net to be reconnected such that the total negative slacks is optimized.

4.1.2 Congestion Optimization. When we improve the timing by reconnecting sinks to their sources, routing congestion may be increased. Hence, routability needs to be considered when we optimize timing by reconnection. The general idea is to avoid reconnecting the critical sink which may go through congested routing regions.

More specifically, a penalty factor of each critical sink is obtained and such penalty will be added to the objective function in order

to consider routability. The penalty factor of each critical sink is calculated from the overflow values of its source, which can be obtained after global routing. To honor our original tree topologies, both the steiner points and pins of each tree are treated as pins in the global router. How we calculate the overflow penalty po_i of critical sink i is illustrated by an example given in Figure 2, where a, b, c are the possible locations of the critical sink and s is the source. For each critical sink i, po_i can be obtained from the overflow values of its source, which are o_{e_u}, o_{e_r}, o_{e_d} and o_{e_l} in this example. There are two kinds of situations as follows:

(1) The routing grid of the critical sink is either horizontal or vertical to the one of its source, such as critical sinks a, c.
(2) Otherwise, such as critical sinks b.

For the first situation, po_i is equal to the overflow of the edge cut through by the connection between the sink and source. For the other situation, po_i is equal to the maximum overflow of the edges cut through by the bounding box of the sink and source. In Figure 2, the overflow penalties of a, b and c are o_{e_u}, $\max(o_{e_u}, o_{e_r})$ and o_{e_r} respectively.

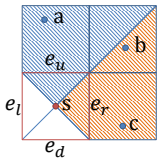


Figure 2: An example of how to calculate potential routing overflow.

By adding overflow penalty po_i into the objective function as shown in Equation (15), we can optimize timing and congestion simultaneously.

$$\max \sum_{i=1}^{n} (\beta \cdot \Delta L_i + (\Delta d_{s \to j_x^i} + \alpha \cdot po_i) \cdot x_i) \quad (15)$$
$$s.t. \quad x_i = \{0, 1\} \quad \forall i \in N_c$$

4.2 Machine Learning-based Acceleration (MLA)

In this section, we first study how the circuit properties will affect the reconnection decisions. For example, the critical sink may have a large detour to the source in the original tree topology and the slew of the critical sink will be improved a lot after reconnection. We will select some of these properties as features and use a classification approach to speed up QPTST. In the following parts, we assume each net i has n sinks and a source s_i and sink j_x^i is the most critical one.

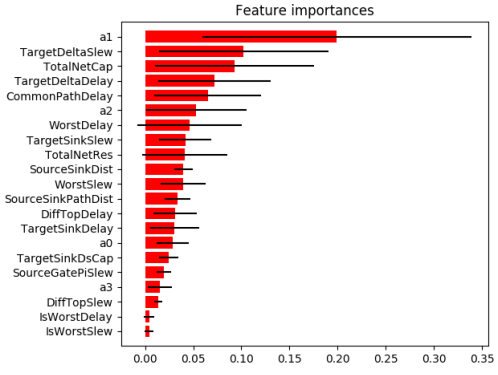


Figure 3: Feature Importances.

The circuit properties we study can be categorized into three types: (1) slew and delay related features as shown in Table 2, (2) distance and length related features as shown in Table 3, and (3) physics related features as shown in Table 4.

Since the ranges of the values of the extracted features $\boldsymbol{f}$ vary a lot, Equation (16) is used for normalization.

$$f' = \frac{f - min(\boldsymbol{f})}{max(\boldsymbol{f}) - min(\boldsymbol{f})}, \quad (16)$$

where $max(\boldsymbol{f})$ and $min(\boldsymbol{f})$ are obtained in the training set.

Large number of features may cause inefficiency and overfitting. Hence, we need to reduce the number of features and features are selected according to their importance. The importances of features are obtained by a machine learning model. As shown in Figure 3, features are ranked according to their importance produced by Random Forest (RF) [14].

After feature selection and preprocessing, we formulate our MLA problem as a classification problem. To be specific, a binary classifier is applied to each critical net to decide whether it should be reconnected and the results of QPTST are used to label the data. Accuracy of the classifier is based on the results of QPTST. We use the RF as our classification model and the top 15 important circuit properties shown in Figure 3 are selected as final features. If congestion is also considered, factor po_i of each net i is added to our features during classification.

Table 2: Delay And Slew Related Features.

DiffTopSlew (*DiffTopDelay*)	The difference of sink j_x^i slew (delay) and the largest slew (delay) of net i except slew (delay) of sink j_x^i.
IsWorstSlew (*IsWorstDelay*)	Whether slew (delay) of sink j_x^i is worst in net i.
WorstSlew (*WorstDelay*)	Value of the worst slew (delay) of net i.
TargetSinkSlew (*TargetSinkDelay*)	The slew (delay) of sink j_x^i.
TargetDeltaSlew (*TargetDeltaDelay*)	The difference slew (delay) of sink j_x^i before and after connecting to root.
CommonPathDelay	The delay accumulated on branches which connects s to j_x^i path.

Table 3: Distance And Length Related Features.

SourceSinkPathDist	The path length from source to the sink j_x^i on the routing tree.
SourceSinkDist	The Manhattan distance between the position of source s and sink j.

Table 4: Physics Related Features.

TotalNetCap	The total capacitance of net i.
TotalNetRes	The total net resistance.
a_0,a_1,a_2 and a_3	The coefficients of lookup table function of source s.

5 EXPERIMENTAL RESULTS

In the experiments, the benchmarks of the contest in ICCAD 2015[13] are used and these benchmarks provide timing information. OpenTimer [9] is used for STA. Our work is implemented in C++ and tested on a 2.1 GHz Intel Linux machine with a 64 GB memory. IBM ILOG CPLEX V12.7.0 [10] is used to solve the QP.

5.1 QPTST Results

5.1.1 Timing Results. The results of QPTST is shown in Table 6. Evaluation is performed by Opentimer [9] and our results provide

Table 5: ICCAD 2015 Benchmarks Information

Designs	#nodes	#nets	clock periods (ns)
superblue10	1876103	1898119	10
superblue1	1209716	1215710	9
superblue16	981559	999902	5.5
superblue18	768068	771542	7
superblue3	1213253	1224979	10
superblue4	795645	802513	6
superblue5	1086888	1100825	9
superblue7	1931639	1933945	5.5

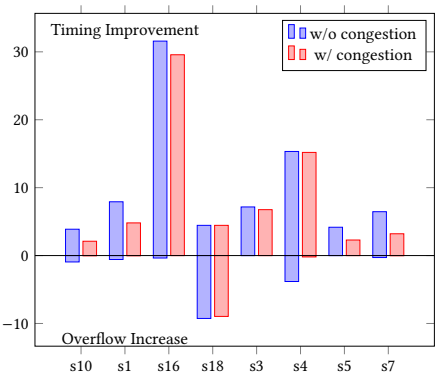


Figure 4: Performance analysis on timing and routing congestion.

interconnection information to it. FLUTE Baseline shows timing results when all the nets are constructed by FLUTE, which does not optimize timing. Direct Connection is the experiment that the most critical sink of every net is reconnected to its source. It can shorten path length of all the nets but will increase wirelength a lot. QPTST is our result and congestion-aware QPTST is the algorithm described in Section 4.1.2. β is set to 2500. QPTST takes 27.6s while congestion-aware QPTST takes 30.71s on average. r_wns and r_tns denote WNS and TNS improvement over the FLUTE baseline. r_stwl and r_d denote tree length improvement over the FLUTE baseline and direct connection. Direct Connection improves timing by 1.10% on worst negative slack (WNS) and 6.81% on total negative slack (TNS). However, it increases 18.74% wirelength as expected. QPTST obtains 2.05% and 10.12% improvement on WNS and TNS. The wirelength is only worse by −0.55%. If we also consider congestion in the objective function, timing is not as good as QPTST but can still get 1.74% and 8.55% improvement on WNS and TNS. The wirelength is improved compared with QPTST. All the results of our algorithms achieve better wirelength and timing compared with the FLUTE baseline and direct connection.

Besides using FLUTE as baseline, we also performed QPTST experiments on timing driven routing tree constructed by the PD method mentioned in [1]. Shown in Table 7, our algorithm achieves 1.7% and 7.11% improvement on WNS and TNS with loss of 0.39% wirelength. It shows that our algorithm is efficient on both non-timing aware trees and timing driven routing.

5.1.2 Congestion Results. Nets are decomposed to two-pin nets by FLUTE first and NCTUgr [16] is performed to measure congestion. As shown in Table 8, r denotes the improvement compared with FLUTE based net decomposition. The overflow of QPTST and congestion aware QPTST increases 1.91% and 1.15% respectively. Wirelength increases 1.29% and 0.79% respectively. In congestion aware TST, α is set to 1000. Our algorithm QPTST can improve timing around 10% but congestion is increased by 1.91%.

5.1.3 Analysis. Figure 4 shows the analysis of performance of our algorithms on timing and congestion. The part above 0 of the chart is timing improvement and the other part is plotted as increase of overflow. It is obvious that our algorithms can achieve significantly timing improvement with small increase in congestion.

5.2 Machine Learning-based Acceleration Results

As mentioned in Section 4.2, top 15 features are selected in our experiments. Data from 4 benchmarks (superblue4, superblue16, superblue18 and superblue7) with scaled features are fed into cross validation and our machine learning model is trained by random forest method. In addition to designs listed in Table 5, we also adopt placement results generated by top 3 contestants in [13]. Therefore, test benchmark sets (superblue1, superblue3, superblue5 and superblue10) with each of 7 placement results are tested. Benchmarks (superblue4, superblue16, superblue18 and superblue7) with untrained placement results from contestants are also tested.

The average evaluation of each benchmark is shown in the Table 9. We can see from the table that runtime is reduced a lot. Moreover, we achieves a relative high accuracy classification rate. The final timing and wirelength improvement are also listed in Table 9. Classification Over Base shows timing results compared with baseline and Classification Over QP shows the comparison with the results of QPTST. The timing and wirelength quality are very compatible compared with our QPTST results.

6 CONCLUSION

Timing is a critical issue for the design optimization and it is hard to improve timing without increasing routing congestion. In this work, we formulate the tree surgery problem as a QP, which optimizes gate delay and net delay with adjacent nets considered. In order to enhance routability, congestion is also optimized in our algorithm. To speed up the process, a machine learning-based algorithm is proposed and features related to timing optimization are extracted from the design. In the experimental results, our algorithms can achieve high quality of timing improvement.

REFERENCES

[1] Charles J Alpert, Andrew B Kahng, CN Sze, and Qinke Wang. 2006. Timing-driven steiner trees are (practically) free. In *Proceedings of the 43rd annual Design Automation Conference*. ACM, 389–392.
[2] Yen-Jung Chang, Yu-Ting Lee, Jhih-Rong Gao, Pei-Ci Wu, and Ting-Chi Wang. 2010. NTHU-route 2.0: a robust global router for modern designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 12 (2010), 1931–1944.
[3] Genjie Chen, Peishan Tu, and Evangeline FY Young. 2017. SALT: Provably Good Routing Topology by a Novel Steiner Shallow-Light Tree Algorithm. In *Proceedings of the 2017 IEEE/ACM international conference on Computer-aided design*. ACM.
[4] Minsik Cho and David Z Pan. 2007. BoxRouter: a new global router based on box expansion and progressive ILP. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 12 (2007), 2130–2143.
[5] Minsik Cho, David Z Pan, Hua Xiang, and Ruchir Puri. 2006. Wire density driven global routing for CMP variation and timing. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 487–492.
[6] Chris Chu and Yiu-Chung Wong. 2008. FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 1 (2008), 70–83.

Table 6: Experimental Results of Tree Surgery Technique.

Benchmarks	FLUTE Baseline**						Direct Connection*			QPTST					Congestion Aware QPTST				
	WNS	r_wns	TNS	r_tns	stWL	r_stwl	r_wns	r_tns	r_stwl	r_wns	r_tns	r_stwl	r_d	CPU(s)	r_wns	r_tns	r_stwl	r_d	CPU(s)
superblue10	-1.65	1.00	-33.10	1.00	2.05	1.00	0.47%	2.47%	-13.92%	0.92%	3.88%	-0.79%	11.52%	69.73	0.00%	2.11%	-0.48%	11.80%	77.03
superblue1	-0.50	1.00	-0.46	1.00	0.96	1.00	-0.26%	3.20%	-19.76%	1.76%	7.92%	-0.38%	16.18%	15.60	1.78%	4.81%	-0.37%	16.19%	25.05
superblue16	-0.46	1.00	-0.76	1.00	0.93	1.00	3.58%	25.18%	-14.74%	3.94%	31.58%	-0.38%	12.52%	6.42	3.94%	29.57%	-0.36%	12.54%	15.21
superblue18	-0.46	1.00	-1.03	1.00	0.58	1.00	-0.75%	2.10%	-23.30%	2.27%	4.45%	-0.18%	18.75%	17.93	2.27%	4.45%	-0.18%	18.75%	13.01
superblue3	-1.01	1.00	-1.50	1.00	1.14	1.00	4.82%	5.79%	-18.87%	5.61%	7.16%	-0.11%	15.78%	6.12	5.37%	6.76%	-0.09%	15.80%	15.10
superblue4	-0.62	1.00	-3.47	1.00	0.71	1.00	0.90%	10.81%	-18.51%	1.60%	15.33%	-1.79%	14.10%	48.95	0.47%	15.19%	-1.58%	14.29%	57.36
superblue5	-2.57	1.00	-6.95	1.00	1.08	1.00	0.07%	1.42%	-17.24%	0.32%	4.17%	-0.62%	14.18%	11.93	0.12%	2.29%	-0.27%	14.48%	22.02
superblue7	-1.51	1.00	-1.84	1.00	1.40	1.00	0.00%	3.54%	-23.56%	0.00%	6.46%	-0.13%	18.96%	44.13	0.00%	3.21%	-0.08%	19.00%	20.91
Average	-1.10	1.00	-6.14	1.00	1.11	1.00	1.10%	6.81%	-18.74%	2.05%	10.12%	-0.55%	15.25%	27.60	1.74%	8.55%	-0.43%	15.35%	30.71

*Direct Connection: directly connect the critical sinks to the source for all nets.

**WNS is in $10^4 ps$. TNS is in $10^6 ps$. stWL is in $10^8 um$.

Table 7: Comparisons Between PD Based Tree Construction and QPTST.

Benchmarks	PD Baseline						Direct Connection			QPTST				
	WNS	r_wns	TNS	r_tns	stWL	r_stwl	r_wns	r_tns	r_stwl	r_wns	r_tns	r_stwl	r_d	CPU(s)
superblue10	-1.66	1.00	-33.14	1.00	2.12	1.00	4.65%	2.78%	-10.97%	5.12%	4.12%	-0.60%	9.34%	57.87
superblue1	-0.50	1.00	-0.47	1.00	1.01	1.00	-0.86%	1.53%	-14.78%	0.90%	5.46%	-0.24%	12.67%	14.14
superblue16	-0.46	1.00	-0.69	1.00	0.96	1.00	2.41%	11.21%	-12.12%	3.10%	19.44%	-0.28%	10.56%	6.01
superblue18	-0.45	1.00	-1.04	1.00	0.63	1.00	-0.74%	-0.29%	-18.12%	1.10%	2.41%	-0.14%	15.23%	4.43
superblue3	-1.01	1.00	-1.51	1.00	1.22	1.00	2.02%	2.90%	-14.80%	2.71%	4.15%	-0.07%	12.84%	5.91
superblue4	-0.63	1.00	-3.54	1.00	0.75	1.00	-0.63%	9.66%	-14.50%	-0.02%	13.20%	-1.29%	11.53%	39.80
superblue5	-2.57	1.00	-6.95	1.00	1.11	1.00	-0.18%	0.12%	-13.43%	0.10%	2.91%	-0.45%	11.44%	12.08
superblue7	-1.52	1.00	-1.81	1.00	1.51	1.00	0.00%	1.06%	-18.69%	0.56%	5.24%	-0.09%	15.67%	11.46
Average	-1.10	1.00	-6.14	1.00	1.16	1.00	0.83%	3.62%	-14.68%	1.70%	7.11%	-0.39%	12.41%	18.96

Table 8: Comparisons Before and After Considering Congestion in QPTST.

	FLUTE Based Net Decomposition				QPTST				Congestion Aware QPTST			
Benchmarks	WL	r	Overflow	r	WL	r	Overflow	r	WL	r	Overflow	r
superblue10	1.71	1.00	7696.32	1.00	1.72	-1.69%	7768.53	-0.94%	1.71	-1.03%	7697.78	-0.02%
superblue16	0.98	1.00	684.93	1.00	0.98	-0.15%	688.89	-0.58%	0.98	0.17%	685.04	-0.02%
superblue18	.63	1.00	74.81	1.00	0.63	0.37%	75.08	-0.36%	0.63	0.55%	74.81	0.00%
superblue1	0.78	1.00	3297.61	1.00	0.82	-3.16%	3603.07	-9.26%	0.82	-2.46%	3593.22	-8.96%
superblue3	1.07	1.00	2538.63	1.00	1.07	-0.83%	2538.63	0.00%	1.07	-0.83%	2538.63	0.00%
superblue4	0.77	1.00	749.92	1.00	0.78	-2.24%	778.55	-3.82%	0.77	-0.18%	751.39	-0.20%
superblue5	0.73	1.00	3060.91	1.00	0.73	-1.72%	3060.80	0.00%	0.73	-1.72%	3060.80	0.00%
superblue7	.37	1.00	3972.55	1.00	1.38	-0.93%	3984.21	-0.29%	1.37	-0.81%	3972.87	-0.01%
Average	1.01	1.00	2759.46	1.00	1.01	-1.29%	2812.22	-1.91%	1.01	-0.79%	2796.82	-1.15%

Table 9: Experimental Results of Machine Learning Acceleration (MLA).

Benchmarks	Classification Accuracy						Classification Over Base			Classification Over QP		
	TP	FP	FN	TN	ACC	CPU(s)	r_wns	r_tns	r_wl	r_wns	r_tns	r_wl
superblue18	2047.17	63.00	41.33	1483.67	97.13%	1.21	0.09%	3.89%	-0.18%	-0.01%	0.01%	0.00%
superblue16	3037.67	122.50	96.33	1833.33	95.53%	1.54	5.24%	29.54%	-0.32%	0.10%	0.14%	0.00%
superblue7	3392.20	119.80	157.00	1769.00	94.91%	1.59	0.00%	5.49%	-0.13%	0.00%	0.41%	0.00%
superblue4	15374.83	92.83	151.33	12844.83	99.14%	6.39	1.55%	13.87%	-1.78%	-0.02%	0.11%	0.00%
superblue1	4651.14	798.71	1201.43	4932.86	82.74%	2.98	1.71%	6.10%	-0.38%	0.42%	1.28%	-0.06%
superblue3	1587.14	310.00	288.57	1211.43	82.38%	1.13	3.67%	5.32%	-0.10%	0.37%	1.26%	-0.05%
superblue5	5552.86	707.14	1182.57	3803.57	83.19%	2.97	0.25%	3.40%	-0.57%	0.05%	0.06%	-0.01%
superblue10	20330.29	2107.29	1974.43	9563.00	87.99%	7.73	0.73%	3.71%	-0.76%	0.07%	0.55%	-0.08%

[7] Jiang Hu and Sachin S Sapatnekar. 2000. A timing-constrained algorithm for simultaneous global routing of multiple nets. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*. IEEE Press, 99–103.

[8] Jin Hu, Greg Schaeffer, and Vibhor Garg. 2015. TAU 2015 contest on incremental timing analysis. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*. IEEE, 882–889.

[9] Tsung-Wei Huang and Martin DF Wong. 2015. OpenTimer: A high-performance timing analysis tool. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 895–902.

[10] IBM. 2017. CPLEX. (2017). https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[11] Tong Jing, Xianlong Hong, Haiyun Bao, Yici Cai, Jingyu Xu, Chungkuan Cheng, and Jun Gu. 2003. UTACO: a unified timing and congestion optimizing algorithm for standard cell global routing. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*. ACM, 834–839.

[12] Andrew B Kahng. 2015. New game, new goal posts: A recent history of timing closure. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 4.

[13] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. 2015. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 921–926.

[14] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[15] Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao. 2013. NCTU-GR 2.0: multithreaded collision-aware global routing with bounded-length maze routing. *IEEE Transactions on computer-aided design of integrated circuits and systems* 32, 5 (2013), 709–722.

[16] Wen-Hao Liu, Cheng-Kok Koh, and Yih-Lang Li. 2013. Optimization of placement solutions for routability. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 1–9.

[17] Michael D Moffitt. 2008. MaizeRouter: Engineering an effective global router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 11 (2008), 2017–2026.

[18] Min Pan, Yue Xu, Yanheng Zhang, and Chris Chu. 2012. FastRoute: An efficient and high-quality global router. *VLSI Design* 2012 (2012), 14.

[19] Ahmed Youssef, Zhen Yang, Mohab Anis, Shawki Areibi, Anthony Vannelli, and Mohamed Elmasry. 2010. A power-efficient multipin ILP-based routing technique. *IEEE Transactions on Circuits and Systems I: Regular Papers* 57, 1 (2010), 225–235.