Applied Mathematics and Computation 255 (2015) 92–104

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

A provably tight delay-driven concurrently congestion mitigating global routing algorithm ☆

Radhamanjari Samanta [a,*], Adil I. Erzin [b,c], Soumyendu Raha [a], Yuriy V. Shamardin [b], Ivan I. Takhonov [c], Vyacheslav V. Zalyubovskiy [b]

[a] Supercomputer Education and Research Center, Indian Institute of Science, Bangalore, India
[b] Sobolev Institute of Mathematics, Siberian Branch, Russian Academy of Sciences, Novosibirsk, Russia
[c] Novosibirsk State University, Novosibirsk, Russia

ARTICLE INFO

Keywords:
Steiner tree
Elmore delay
Global routing
Gradient method

ABSTRACT

Routing is a very important step in VLSI physical design. A set of nets are routed under delay and resource constraints in multi-net global routing. In this paper a delay-driven congestion-aware global routing algorithm is developed, which is a heuristic based method to solve a multi-objective NP-hard optimization problem. The proposed delay-driven Steiner tree construction method is of $O(n^2 \log n)$ complexity, where n is the number of terminal points and it provides n-approximation solution of the critical time minimization problem for a certain class of grid graphs. The existing timing-driven method (Hu and Sapatnekar, 2002) has a complexity $O(n^4)$ and is implemented on nets with small number of sinks. Next we propose a FPTAS Gradient algorithm for minimizing the total overflow. This is a concurrent approach considering all the nets simultaneously contrary to the existing approaches of sequential rip-up and reroute. The algorithms are implemented on ISPD98 derived benchmarks and the drastic reduction of overflow is observed.

1. Introduction

The Global Routing Problem (GRP) in VLSI design is a problem of routing a set of nets (multi-net global routing) subject to limited resources and delay constraints. There are various recent approaches for solving GRP [2,5,20,15,18,23] available, but the referred methods are not timing-driven. Most of these modern routers generate Steiner trees with highly optimized wirelength and then use rip-up and reroute iteratively for reducing the congestion. But merely optimizing the wirelength and then minimizing the overflow will not produce a feasible routing because they will not necessarily meet timing at the sinks. A simple example is shown in Fig. 1 to demonstrate that optimum wirelength does not necessarily mean optimum delay and vice versa.

The computation of delay is heavily dependent on pendant subtrees. Therefore, optimizing delay and congestion is a multi-objective constraint, which is the focus of this work. In the example, Fig. 1(a) shows that, given 3 pins, 1(source), 2(sink), and 3(sink), we first draw the Hanan grid by drawing horizontal and vertical lines through them. The intersecting

☆ The reported study was partially supported by RFBR, Russia, research project No. 13-07-00139, and by Basic Science Research Program (NRF-2013R1A1A2064302) through NRF, Korea.

* Corresponding author.
E-mail addresses: samanta@ssl.serc.iisc.in, radhamanjari@gmail.com (R. Samanta), adilerzin@math.nsc.ru (A.I. Erzin), raha@serc.iisc.in (S. Raha), orlab@math.nsc.ru (Y.V. Shamardin), takhonov@gmail.com (I.I. Takhonov), slava@math.nsc.ru (V.V. Zalyubovskiy).

http://dx.doi.org/10.1016/j.amc.2014.11.062
0096-3003/

points are the generated Steiner points. Fig. 1(b) shows the minimum wire-length(WL = 3 units) tree configuration and Fig. 1(c) shows another tree configuration for the net, whose wirelength, WL = 4 units. Now, we compute the delay of the sink nodes for both the trees. In the figure, all $R_i = R$ and all $C_i = C$. Though the tree in Fig. 1(b) has minimum WL(3 units), it has $3RC$ delay whereas tree in Fig. 1(c) has 4 units of WL but $2RC$ delays at the sinks as can be seen from the following equations. For, Fig. 1(b),

$$\text{Delay at node2}, d_{12} = R_1 * C_3 + (R_1 + R_2) * C_2 \simeq 3RC;$$
$$\text{Delay at node3}, d_{13} = R_1 * C_2 + (R_1 + R_3) * C_3 \simeq 3RC;$$

and for, Fig. 1(c),

$$\text{Delay at node2}, d_{12} = (R_1 + R_2) * C_2 \simeq 2RC;$$
$$\text{Delay at node3}, d_{13} = (R_4 + R_3) * C_3 \simeq 2RC.$$

As suggested by Moffitt et al. [21], there is increasing demand of timing-driven routing algorithms and there are not many works focused in this area. There are a few global routing algorithms [14,30,29] based on MVERT [13], which consider timing but they are of complexity $O(n^4)$ (n is the number of sinks), and implemented on nets with small number of sinks. Also GRP was formulated as a multi-commodity flow Problem [12] as well. With each net a certain flow of unit size is associated. Each edge has a flow capacity. With respect to the objective function we may get a min-cost multi-commodity flow problem or concurrent multi-commodity flow problem [27,3,1,10]. Meta-heuristics to solve GRP can be found in Timber-Wolf [8] (simulated annealing), [4] (evolution algorithm), [9] (genetic algorithm) and [31] (tabu search). Fault tolerant routing method for network-on-chip is described in [17].We propose an $O(n^2 \log n)$ method [26] for constructing delay-driven Steiner trees. Another contribution of our work is a Gradient based approach for minimizing the overflow. The novelty of this algorithm is that, it considers all the nets concurrently and provides a fully polynomial time approximation scheme(FPTAS).

In Section 2, the problem is formulated, and Section 3 describes our proposed algorithm MAD (Modified Algorithm of Dijkstra) and its iterative modifications (IMAD). IMAD is applied to create minimum critical delay Steiner trees for each net. History based IMAD described in Section 5 is used to create congestion-aware trees for each net. Also we have used a router FLUTE [7] to generate another set of Steiner candidate trees for each net. Finally a gradient algorithm is used to pick one tree for each net from its candidate set of trees, such that the total congestion/overflow is minimal. The Gradient method is described in Section 4.

Since there are no recent timing-driven router available, we run MAD without Gradient on IBM/ISPD98 benchmarks and this gives us the initial congestion of the chip. Then we run IMAD with Gradient and show how effectively it reduces the congestion. The benchmarks are modified by assigning resistance, capacitance values to the wires. We show that 66.4% trees

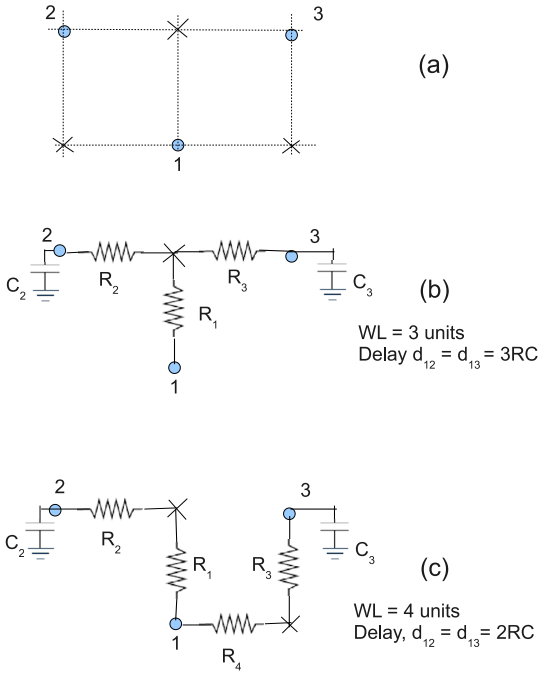

Fig. 1. Minimum wirelength does not necessarily mean minimum delay and vice versa.

picked by Gradient are generated by MAD and the rest are from FLUTE. The experimental results are shown in Section 7, and Section 8 gives the conclusion. The appendix gives the theoretical analysis of MAD.

2. Problem formulation

The problem can be formulated as a two-criteria problem as follows. In the global graph, it is required to find a set of Steiner trees, each of which connects a subset of vertices, such that the total congestion overflow, and the maximum timing slack are minimal. The approximate solution to this problem is constructed in two stages. First we construct a set of timing-driven Steiner trees for each net based on Elmore delay, and then select one tree for each net, taking into account the density of connections. Density of connection is the number of wires passing through an edge. It is used as a measure of congestion.

At the first stage, given an undirected graph $G = (V, E)$, $|V| = m$. To each edge $(i,j) \in E$ two non-negative parameters r_{ij} (*resistance*) and c_{ij} (*capacitance*) are assigned. Consider the subset of vertices $S_v = \{0, 1, \ldots, n\} \subseteq V$, where vertex 0 is the source of the graph (since the graph is undirected, it has no source and we call vertex 0 the source of signal) and nodes $S_v \setminus \{0\}$ are the *sinks* or *terminals*. We call nodes from $V \setminus S_v$ intermediate nodes. Each sink $i \in S_v$ has the *capacitance* c_i and vertex 0 has also the *resistance* r_0.

Let us consider a Steiner tree T spanning S_v and rooted in 0. We are using the Elmore delay metric. Let k be an arbitrary terminal in T and define Elmore delay as $t_k(T)$. Let us denote:

- $P_k(T)$ - path from 0 to k in T. Also $P_{uv}(T)$ denotes path from u to v in T. Argument T is omitted where it is obvious from context;
- T_j $(j \in V(T))$ is downstream subtree of T with root j; T_e $(e \in E(T))$ is downstream subtree of T rooted in the head node of arc e;
- $C(H)$ $(R(H))$ - total capacitance (resistance) of subgraph $H, C(H) = \sum_{e \in E(H)} c_e + \sum_{i \in V(H)} c_i$. Also we use notations: $C_j = C(T_j), C_e = C(T_e)$ and $R_{uv} = R(P_{uv})$.

The Elmore delay [22,25] along the arc (i,j) in T is defined as follows.

$$d_{ij} = d_{ij}(T) = r_{ij}\left(\frac{c_{ij}}{2} + C_j\right). \tag{1}$$

The delay of signal propagation from source 0 to terminal k (delay along the path $P_k(T)$) is given by

$$t_k = t_k(T) = r_0 C_0 + \sum_{(i,j) \in P_k(T)} d_{ij}. \tag{2}$$

The maximum among all the delays to terminals in T is called the *critical delay* and is denoted by $t^*(T)$. Terminal $k : t_k(T) = t^*(T)$ is a *critical terminal* in tree T. In this paper, we discuss the sub problem of finding such a tree spanning S in G that provides minimum critical delay:

$$\max_{i \in S} t_i(T) \to \min_{T}. \tag{3}$$

This problem is known to be NP-hard [16]. The following section describes the proposed approximate algorithm, Modified Algorithm of Dijkstra (MAD) for solving problem (3).

3. Algorithm MAD

Steps of the Algorithm:

Step 0. Set tree $T = (0, \emptyset)$ and delay $t_0 = 0$;
Step 1. Find $(i,j) = \arg\min_{\substack{(u,v) \in E; \\ u \in T, v \notin T}} \{t_u(T \cup \{u, v\}) + d_{uv}\}$,

where
$t_u(T \cup \{u, v\}) = t_u(T) + r_0(c_{uv} + c_v) + \sum_{e \in P_u(T)} r_e(c_{uv} + c_v)$.
Set $T = T \cup \{i,j\}$ and recalculate the delays t_k $(k \in T)$:

- If j is an intermediate vertex, then set $t_j = t_i + d_{ij}$, and the delays t_k in all the other vertices do not change;
- if j is a terminal, then cut off all the pendent subtrees not containing terminal vertices and recalculate all the delays in T by formula (2).

If not all terminals are included in T then go to Step 1.

3.1. Demonstration of MAD with an example:

Here we will show how MAD creates a rectilinear Steiner tree from a given set of points. In Fig. 2, the small circles 0, 1 and 2 are the given terminal points of a net, which are also called sinks. To generate Steiner points for rectilinear geometry, vertical and horizontal lines are drawn through nodes 0, 1 and 2 resulting the grid structure in Fig. 2. The crosses at the intersection of the lines are the candidate Steiner points. Now, we start from root node 0 with an initial trivial (the set of edges is empty) tree. We assume delay at $t_0 = 0$. To make the example brief and simple, we will not get into the details of the delay calculation using the formulas. Instead we will assume the delay values at the nodes and generate the tree. Two edges (0, 3) and (0, 5) are going out from root 0. Let us assume, $\{t_0(T \cup \{0,3\}) + d_{03}\} < \{t_0(T \cup \{0,5\}) + d_{05}\}$. Therefore edge (0, 3) is selected by MAD. Since 3 is an intermediate (Steiner) vertex, just the delay at node 3 is updated as $t_3 = t_0 + d_{03}$. Similarly in the next two iterations, edges (0, 5) and (5, 6) are added to the tree and t_5 and t_6 are calculated. Next the minimum delay edge picked up by MAD is (5, 1). But the node to be added is 1, which is a sink. Therefore, all the pendent subtrees not containing terminal vertices are to be removed from the tree. The resulting tree in this iteration is shown in Fig. 3. And now the delays of the nodes in the tree have changed because subtrees of some of the nodes have changed. Therefore, t_0, t_5 and t_1 are calculated using Eq. (2). Similarly edges (1, 7), (5, 6) and (7, 2) are added in the next three consecutive iterations and the delays of the added nodes t_7 and t_6 are calculated. Now the last node added to the tree is sink 2. Therefore, after cutting-off the redundant subtrees, the tree is as in Fig. 4. Since all the sinks are added to the tree, the iteration stops.

Time Complexity of Algorithm MAD: The generated set of Steiner points is reduced to $O(cn)$, where n is the number of sinks/terminal points and c is a small factor. The reduction is based on a clustering technique adapted from [6]. The worst-case time complexity of MAD is $O(n^2 \log n)$. The number of iteration of Step 1 is upper bounded by n. And the complexity of each Step 1 is $O(r \log r + g)$ based on Fibonacci-heap implementation, where r, the total number of nodes in the graph is $O(cn)$ and g, the number of edges is a linear function of n. Therefore, the worst-case complexity of MAD is $O(n^2 \log n)$.

The algorithm admits the following iterative modifications.

Algorithm IMAD-1 at iteration $k+1$ constructs a Steiner tree with MAD using the values of delays from the tree built at the previous iteration. At the 1st iteration, the previous tree is trivial.
Let T^k be the tree constructed by the algorithm at the k-th iteration and d_{ij}^k be delay along the arc $(i,j) \in T^k$. The algorithm uses MAD to construct T^{k+1} and at each step of MAD the edge (u, v) that minimizes the value: $t_u(T^{k+1} \cup \{u, v\}) + d_{uv}^k$ $((u, v) \in E, u \in T^{k+1}, v \notin T^{k+1})$ is attached to the tree.

Algorithm IMAD-2 at iteration $k+1$ constructs a Steiner tree with MAD using the values of delays from all the trees built at all the previous iterations. At the 1st iteration, the previous tree is trivial.
Let $\bar{d}_{ij}^k$ be the arithmetic mean of delays d_{ij}^l along the arc (i,j) in trees T^l $(l \leqslant k)$. The algorithm uses MAD to construct T^{k+1} and at each step of MAD the edge (u, v) that minimizes the value: $t_u(T^{k+1} \cup \{u, v\}) + \bar{d}_{uv}^k$ $((u, v) \in E, u \in T^{k+1}, v \notin T^{k+1})$ is attached to the tree.

The algorithms stop, when $T^{k+1} = T^l$ $l \leqslant k$, or maximum number of iterations is performed.

3.2. Preliminary lemmas

Here we introduce some denotations and preliminary lemmas which will be used later.

Remark 1. Let T be a Steiner tree spanning $S, u \in S$ be a leaf in T and P be the path connecting root 0 and u. $T \setminus P$ is a set of subtrees $\{T_{(0)}, \ldots, T_{(k)}\}$. Denote by v_i the root of $T_{(i)}, v_i \in P$. Vertices $\{v_1, \ldots, v_k\}$ split P into $k+1$ parts: $P = P_1 \cup P_2 \cup \ldots \cup P_{k+1}$, where $P_1 = P_{0v_1}, P_{k+1} = P_{v_k u}$ and $P_i = P_{v_{i-1} v_i}$ for $i = 1, \ldots, k$ (see Fig. 5). We will use these denotations in following statements.

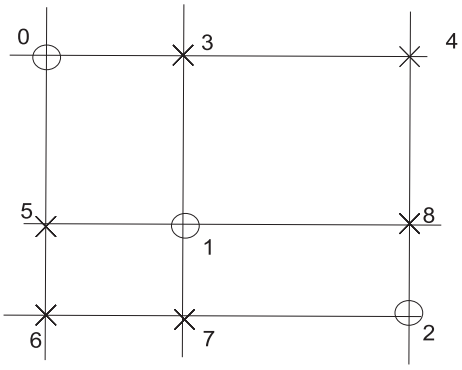

Fig. 2. Candidate Steiner points.

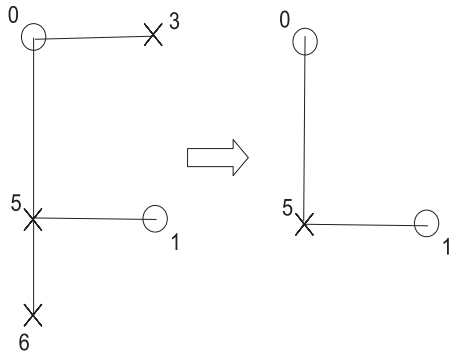

Fig. 3. Intermediate tree.

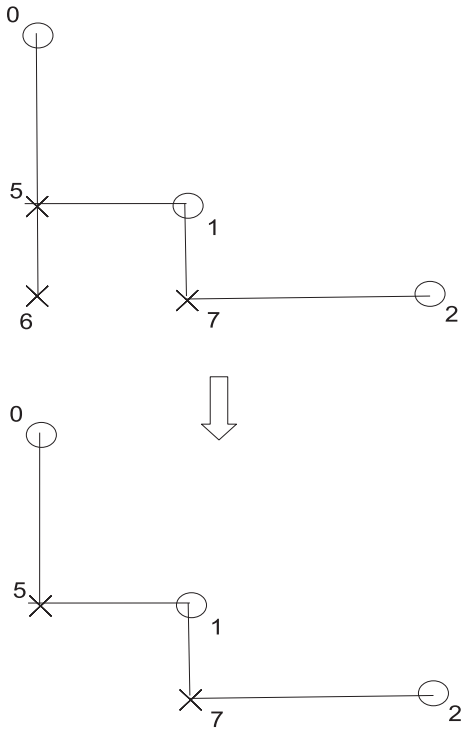

Fig. 4. Final Steiner tree generated by MAD.

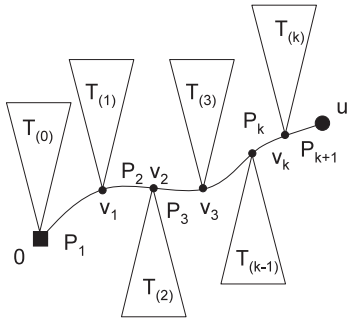

Fig. 5. Remark 1.

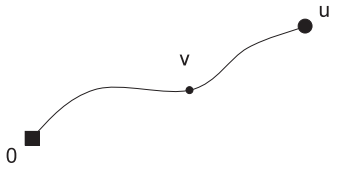

Fig. 6. Lemma 2.

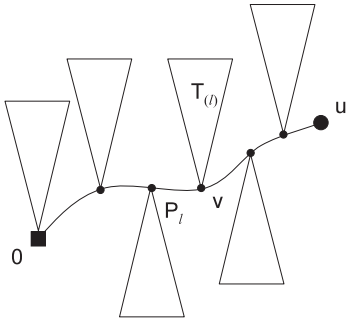

Fig. 7. Lemma 3.

Consider Steiner tree T. Let $\hat{t}_u$ be the delay along the path P_u (disregarding subtrees) and $\bar{t}_{uv}$ be the delay in vertex v in subtree $T_u \setminus T_v$.

Lemma 1. *Let T be a Steiner tree for S and $u \in S$ be a leaf in T. Then*

$$t_u(T) = \hat{t}_u + r_0 \sum_{i=0}^{k} C(T_{(i)}) + \sum_{i=1}^{k} R(P_i) \sum_{j \geq i} C(T_{(j)}),$$

where P_i and $T_{(j)}$ are defined as in Remark 1.

Proof. proof is omitted due to page limitation. Details can be obtained in [28]. □

Lemma 2. *Given the path P connecting the source vertex 0 with a terminal u and $v \in P$ (see Fig. 6). Then*

$$t_u(P) = \hat{t}_{0v} + \hat{t}_{vu} + R_{0v}(C_{vu} + c_u).$$

Proof. proof is omitted due to page limitation. Details can be obtained in [28]. □

Lemma 3. *Let T be a Steiner tree for $S, u \in S$ be a leaf in T and $v \in P_u$ (see Fig. 7). Then*

$$t_u(T) = \bar{t}_{0v} + \bar{t}_{vu} + R_{0v} C(T_v).$$

Proof. proof is omitted due to page limitation. Details can be obtained in [28]. □

4. Congestion aware tree selection

Algorithm IMAD constructs a set of timing-driven Steiner trees for each net in the global graph.

The algorithm is applied when the following inputs are provided. Logical network given as a set of nets and primary inputs with AT(Arrival Time) s and primary outputs with RT (Required Time) s, Number of layers, Specific resistance and capacitance and maximum number of channels Q_{ij} (capacity of corresponding global edge) in each layer, and Resistances and capacitances of vias.

The best trees generated by IMAD are stored as candidate tree set for that net. To increase the cardinality of the candidate set, we use FLUTE [7] along with IMAD. FLUTE is a fast and accurate method for construction of rectilinear Steiner minimal tree (RSMT). We create Steiner trees for each net applying L-routing on two pin nets decomposed from multi-pin nets

(decomposition done by FLUTE). We check the delay of the FLUTE generated trees. If the tree generated from FLUTE is new(i.e. if it was not already generated by IMAD) and has comparable delay with IMAD generated trees, then it is added to the candidate tree set of that net. We have measured comparable delay of FLUTE as 95% to 105% of delay of trees generated by IMAD. Therefore, we have several feasible Steiner trees of various types in the candidate set for each net. A gradient based method is described below which is used to pick one tree for each net from its candidate set Q^s, so that the total overflow of the edges is minimum. Consider the problem of optimal use of routing resources, i.e. the available routing tracks. We use index $e \in E$ for edge of global graph G, and index t for trees, where

$$t \in J = \bigcup_{s=1}^{S} Q^s,$$

$$\text{Set } a_{et} = \begin{cases} 1, & \text{if edge } e \in \text{the tree } t \\ 0, & \text{otherwise} \end{cases} \text{ and}$$

$$x_t = \begin{cases} 1, & \text{if tree } t \text{ is selected} \\ 0, & \text{otherwise.} \end{cases}$$

Then the problem is defined as follows:

$$f(x) = \sum_{e \in E} \left(\max \left\{ 0, \sum_{t \in J} a_{et} x_t - q_e \right\} \right)^2 \to \min_{x_t \in [0,1]}; \tag{4}$$

$$\sum_{t \in Q^s} x_t = 1, \quad s = 1, \ldots, S. \tag{5}$$

Objective (4) is the sum of penalties for capacity overflows of routing resources $q_e, e \in E$. If there is no overflow, then the penalty (4) is zero. Linear relaxation of (4) and (5) is considered. The function $f(x)$ is convex and smooth, therefore the gradient algorithm, described below can be applied.

For the reason of simplicity we rearrange the objective function:

$$\tilde{f}(y) = \sum_{e \in E} (\max \{0, y_e - q_e\})^2,$$

where $y_e = \sum_{t \in J} a_{et} x_t, e \in E$.

We use the following notations below: let $\varepsilon > 0$ be desired precision of solution and L be a lower bound of objective function of problem (4) and (5).

Algorithm GradAl.

Step 0. Initialization.

- $\tilde{x}_t := \frac{1}{|Q^s|}$ $(t \in Q^s, \quad s = 1, \ldots, S)$;
- $\tilde{y}_e := \sum_{t \in J} a_{et} \tilde{x}_t$ $(e \in E)$;
- $L := 0$.

Step 1. Iteration. Calculate the values

- $g_e = \max\{0, \tilde{y}_e - q_e\}$ $(e \in E)$;
- $t_s = \arg \min_{t \in Q^s} \sum_{e \in E} g_e a_{et}$

and set

- $\delta_t = \begin{cases} 1 - \tilde{x}_t, & t = t_s; \\ -\tilde{x}_t, & t \neq t_s; \end{cases}$ for each $t \in Q^s, s = 1, \ldots, S$;
- $z = (z_e), \quad z_e = \sum_{t \in J} a_{et} \delta_t, \ e \in E$;
- $GZ = \sum_{e \in E} g_e z_e$.

Recount the lower bound: $L := \max\{L, \tilde{f}(\tilde{y}) + 2GZ\}$.
If $\tilde{f}(\tilde{y}) - L \leqslant \varepsilon \max\{1, L\}$, then STOP. Else GOTO Step 2.

Step 2. Move to a new point.

Calculate $\tilde{p} = \min\{1, -\frac{GZ}{ZZ}\}$, GZ is defined above, $ZZ = \sum_{e\in E}(z_e)^2$, and set

- $\tilde{x}_t := \tilde{x}_t + \delta_t\tilde{p}$, $(t \in J)$;
- $\tilde{y}_e := \tilde{y}_e + z_e\tilde{p}$, $(e \in E)$.

Then GOTO Step 1.

Claim 1. The algorithm provides $(1+\varepsilon)$-approximation solution of the problem (4) and (5). Its time complexity is $O(|E||J|\varepsilon^{-1}\ln\varepsilon^{-1})$.

Proof. First demonstrate that the objective function decreases at each iteration. Since function $\tilde{f}(y)$ is convex, then

$$\tilde{f}(y) \geqslant \tilde{f}(\tilde{y}) + \bigtriangledown\tilde{f}(\tilde{y})(y-\tilde{y}) = \tilde{f}(\tilde{y}) + \sum_{e\in E} 2g_e(y_e - \tilde{y}_e),$$

where y is arbitrary feasible point. Estimate the last expression by solving the problem

$$\sum_{e\in E} g_e y_e = \sum_{e\in E} g_e \sum_{t\in J} a_{et}x_t = \sum_{t\in J}\left(\sum_{e\in E} g_e a_{et}\right)x_t \to \min_x,$$

The problem above decomposes into the independent subproblems for each net s:

$$\min_x \sum_{t\in J}\left(\sum_{e\in E} g_e a_{et}\right)x_t = \sum_{s=1}^{S}\min_{t\in Q^s}\left(\sum_{e\in E} g_e a_{et}\right) = \sum_{s=1}^{S}\sum_{e\in E} g_e a_{et_s},$$

the last equality follows from the definition of t_s. Thus,

$$\sum_{e\in E} g_e(y_e - \tilde{y}_e) \geqslant \sum_{s=1}^{S}\sum_{e\in E} g_e a_{et_s} - \sum_{e\in E} g_e\tilde{y}_e = \sum_{e\in E} g_e\left(\sum_{s=1}^{S} a_{et_s} - \sum_{t\in J} a_{et}x_t\right) = \sum_{e\in E} g_e\left(\sum_{s=1}^{S} a_{et_s}(1-x_{t_s}) - \sum_{t\in J,\ t\neq t_s} a_{et}x_t\right)$$
$$= \sum_{e\in E} g_e \sum_{t\in J} a_{et}\delta_t = GZ.$$

Therefore,

$$\tilde{f}(y) \geqslant \tilde{f}(\tilde{y}) + 2GZ,$$

$L = \tilde{f}(\tilde{y}) + 2GZ$ is a lower bound for $\tilde{f}(y)$, and if $\tilde{f}(\tilde{y}) - L \leqslant \varepsilon\max\{1, L\}$, then $\tilde{y}$ defines a $(1+\varepsilon)$-approximation. Otherwise the descent direction is defined by vectors δ and z. □

The descent direction δ is from current point $\tilde{x}$ to the integer point $x = (x_t)$, where $x_t = 1$ if $t \in \{t_1, \ldots, t_s\}$, and $x_t = 0$, otherwise. In order to find a stride parameter $p \in [0, 1]$, we can find minimum of function

$$h(p) = f(\tilde{y} + zp) = \sum_{e\in E}(\max\{0, \tilde{y}_e - q_e + z_e p\})^2,$$

but we will use simpler function

$$r(p) = \sum_{e\in E}(g_e + z_e p)^2.$$

It is easy to check that $h(0) = r(0), h'(0) = r'(0), h(p) \leqslant r(p)$ for all $p \in [0, 1]$, and function r reaches a minimum in $\tilde{p} = \min\{1, -\frac{GZ}{ZZ}\}$.

If the current point $\tilde{y}$ is not optimal, then $h'(0) < 0, r'(0) < 0$ and $\tilde{p} > 0$. Inequality $h(0) > h(\tilde{p})$ follows from the expression $h(0) = r(0) > r(\tilde{p}) \geqslant h(\tilde{p})$.

Therefore, moving into point $\tilde{x}'_t = \tilde{x}_t + \delta_t\tilde{p}$ $(t \in J)$ and $\tilde{y}'_e = \tilde{y}_e + z_e\tilde{p}$ $(e \in E)$ does not increase the objective function.

Now we prove, that each iteration has time complexity $O(|E||J|)$, and the total number of iterations is bounded by $O(\varepsilon^{-1}\ln\varepsilon^{-1})$.

Let us first find a lower bound for $\rho = r(0) - r(\tilde{p})$ for one iteration. Set $\Delta = -2GZ$. From definition of $r(p)$ and step (stride parameter) $\tilde{p}$ follows that if $\Delta \geqslant 2ZZ$, then $\rho \geqslant \Delta/2$, and if $\Delta \leqslant 2ZZ$, then $\rho \geqslant \frac{\Delta^2}{4ZZ}$. Finally

$$\rho \geqslant \min\left\{\frac{\Delta}{2}, \frac{\Delta^2}{4ZZ}\right\} = \frac{\Delta}{2}\min\left\{1, \frac{\Delta}{2ZZ}\right\}.$$

The same estimation takes place for argument $d = f(\tilde{y}) - f(\tilde{y} + z\tilde{p})$ of objective function, since $d \geqslant \rho$.

Let us now estimate the upper bound for $D = f(\tilde{y}) - L$. Define Q as the total number of iterations. Parameters of one iteration we define by upper index $l = 0, 1, \ldots, Q$. Step 0 corresponds iteration $l = 0$, then $\tilde{y}^0$ is initial point, $L^0 = 0$ and $D^0 = f(\tilde{y}^0) - L^0 = f(\tilde{y}^0)$. Define c as number which is greater than each $f(\tilde{y}^l)$, Δ^l and $ZZ^l, l = 0, 1, \ldots$, even if the process is infinite. Such number exists since the feasible set of linear relaxation problem is compact.

From equalities $f(\tilde{y}^l) = f(\tilde{y}^{l-1}) - d^l, L^l = \max\{L^{l-1}, f(\tilde{y}^{l-1}) - \Delta^l\}$ and $D^l = f(\tilde{y}^l) - L^l$ follows that $D^l = \min\{D^{l-1}, \Delta^l\} - d^l$. Taking into account the above lower bound for d^l, the definition of number c and inequality $\Delta^l \geqslant 0$, we get

$$D^l \leqslant \min\{D^{l-1}, \Delta^l\} - \frac{(\Delta^l)^2}{4c} \leqslant \max_{v \geqslant 0}\left\{\min\{D^{l-1}, v\} - \frac{v^2}{4c}\right\}.$$

Maximum of the last expression reached when $v = D^{l-1}$, then

$$D^l \leqslant D^{l-1} - \frac{(D^{l-1})^2}{4c} = \left(1 - \frac{D^{l-1}}{4c}\right)D^{l-1}.$$

Continuing this inequality using inequality $D^{l-1} > \varepsilon$, which is true at each iteration except the last one. As a result, we get

$$D^l \leqslant \left(1 - \frac{\varepsilon}{4c}\right)D^{l-1} \leqslant \left(1 - \frac{\varepsilon}{4c}\right)^l D^0 \leqslant \left(1 - \frac{\varepsilon}{4c}\right)^l c.$$

The number of iterations Q is bounded by $T(\varepsilon)$, and $\left(1 - \frac{\varepsilon}{4c}\right)^{T(\varepsilon)} c = \varepsilon$ and since $\lim_{\varepsilon \to 0} \frac{T(\varepsilon)}{\varepsilon^{-1}\ln\varepsilon^{-1}} = 4c$, the complexity do not exceed $O(\varepsilon^{-1}\ln\varepsilon^{-1})$.

5. History based MAD

To reduce the overflow further a penalty based on congestion history is added to the original cost function. Optimizing congestion and delay at the same time is a difficult task as minimizing delay can lead to higher congestion and vice versa. Therefore, a trade-off between the delay and the overflow is done in the cost function so that the delay is not increased beyond a certain range to mitigate the congestion. Penalty is added to the history based cost of all the overflowed edges.

The history based cost at $(i+1)$th iteration of an edge e is given as.

$h_e^{i+1} = \begin{cases} h_e^i + c, & \text{if edge } e \text{ has overflow} \\ h_e^i, & \text{otherwise} \end{cases}$, where h_e^i is the history based cost of edge e at ith iteration and c is a constant which can be increased to give more weightage to congestion in the cost function. The actual cost of the edge e is now calculated as $c_e = (b_e + h_e).p_e$, where b_e represents the base cost of Step 1 of algorithm MAD of section 3, which is dedicated to optimize delay. And p_e represents the current congestion penalty. This approach is called Negotiated Congestion Routing (NCR) [19]. p_e can be obtained using the following equations. Let the density of an edge, $d(e) = \frac{\text{demand(current number of wires passing through the edge)}}{\text{supply(capacity)}}$. Then the congestion penalty term p_e[24] for edge e is defined as, $p_e = \begin{cases} exp(\beta(d_e - 1)), & \text{if } d_e > 1 \\ d_e, & \text{otherwise} \end{cases}$, where value of β used is ln 5 in the experiments.

Table 1
Results of MAD with and without Gradient on Examples Derived from IBM/ISPD98 benchmarks.

Bench name	Grids	# of Nets	MAD w/o Gradient				History based mad with gradient			
			Max ovfl	Total ovfl	Delay (ps)	Run time (s)	Max ovfl	Total ovfl	Delay (ps)	Run time (s)
ibm01dd	64 × 64	11507	16	2494	23209.31	12.4	6	288	23539.25	21.2
ibm02dd	80 × 64	18429	17	2692	31201.34	18.9	4	59	31764.75	34.5
ibm03dd	80 × 64	21621	12	660	37339.23	17.3	2	3	37643.41	36.5
ibm04dd	96 × 64	26163	12	1574	54112.32	43.9	5	138	54289.78	83.5
ibm06dd	128 × 64	33354	16	2975	62911.21	47.4	5	90	63412.81	104.3
ibm07dd	192 × 64	44394	9	270	84201.69	134.2	0	0	84976.51	228.1
ibm08dd	192 × 64	47944	22	3531	91381.24	113.7	4	43	91729.24	238.7
ibm09dd	256 × 64	50393	10	1920	92817.23	189.1	4	77	93102.13	359.3
ibm10dd	256 × 64	64227	16	2483	126672.23	189.4	5	32	127134.33	435.7

Table 2
% of FLUTE generated trees selected by Gradient.

Bench name	% of FLUTE tree selected
ibm01dd	34.1
ibm02dd	37.8
ibm03dd	33.8
ibm04dd	38.3
ibm06dd	34.9
ibm07dd	27.2
ibm08dd	34.1
ibm09dd	31.7
ibm10dd	30.1
Average	33.6

6. Diversity of the generated steiner trees

The performance of the Gradient algorithm depends on the diversity of the candidate Steiner tree set. The dissimilarity between the trees will give a number of routing possibilities for each net. Using a combination of trees, the congestion can be effectively reduced. To measure the diversity of the Steiner tree pool, we use the metric described in [11]. Diversity of the pool,
$D = \frac{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} D_{ij}}{\frac{N(N-1)}{2}} = \frac{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} 1-S_{ij}}{\frac{N(N-1)}{2}} = \frac{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} 1-\frac{\mathrm{comm}(E_i E_j)}{\max(E_i E_j)}}{\frac{N(N-1)}{2}}$, where N is number of candidate Steiner trees in the pool,
S_{ij} is similarity between two trees i and j,
D_{ij} is $(1 - S_{ij})$, which is dissimilarity between two trees i and j,
$\mathrm{comm}(E_i, Ej)$ is percentage of common edges shared by tree i and j, and
$\max(E_i, Ej)$ represents maximum number of edges in either tree.

7. Experimental results

Algorithm IMAD, history based MAD and FLUTE generate a set of perspective trees for each net. For the experiment, the maximum number of candidate trees used in the pool is 7. The average diversity of the trees is 32%. A tree is chosen for each net using the gradient algorithm such that minimal residual capacity of global edges is maximal.

All algorithms are implemented in C on a quad-core AMD Opteron machine on Linux. We note that no benchmark suites are available for *delay driven* routing. Existing methods [30,29,14] consider examples too small (maximum number of nets 1294) to be meaningful for modern VLSI physical design practices. Also, being a multi-objective NP-hard problem, our delay driven routing with practical running times *reduces* overflow similar to the existing algorithms and unlike *congestion aware* only routing algorithms. We run MAD without Gradient and IMAD with Gradient on a set of examples derived from modifying the IBM/ISPD 98 benchmark suite by assigning resistance(0.016 specific resistance) and capacitance(0.47 specific capacitance) values to the wires. The modified examples are renamed with suffix "dd" to the benchmark designs.

The results are shown in Table 1. From Table 1, it can be seen how Gradient reduces the overflow efficiently. In Table 2, we show the percentage of FLUTE generated trees selected by the gradient algorithm. Average 33.6% selected trees are generated by FLUTE, i.e 66.4% selected trees are from MAD, which establishes the importance of MAD as a timing-driven router.

8. Conclusion

In this paper, we proposed a provably tight timing-driven Steiner tree construction algorithm. Also we proposed a gradient based method for minimizing the overflow. We have implemented our algorithm on modified benchmarks derived from large industry-standard benchmarks called ibm/ISPD 98 benchmarks. None of the available timing-driven global routing algorithms work on such large number of nets. The current algorithm has limitation on getting zero-overflow solutions. In future research, we are working on further reduction of the overflow while maintaining the timing at the sinks.

Appendix A. Theoretical Analysis of MAD

In this section, we investigate the performance of MAD applied to grid graphs. We call G a $M \times N$ *grid graph* if the set of vertices $V(G) = \{(m,n)|\ 0 \leqslant m \leqslant M, 0 \leqslant n \leqslant N\}$ for some M and N and $E(G) = \{(v_1, v_2)|v_1 = (m,n), v_2 = (m,n+1) \text{or} v_2 = (m+1,n)\}$. Weighted grid graph G belongs to class Γ_1 if resistances and capacitances of all the edges of one line (horizontal or vertical) are equal, i.e. given edges $e_1 = ((m_1,n_1),(m_1,n_1+1))$ and $e_2 = ((m_1,n_2),(m_1,n_2+1))$, then $r_{e_1} = r_{e_2}$ and $c_{e_1} = c_{e_2}$. The same property holds for horizontal edges.

Weighted grid graph G belongs to class Γ_2, if $G \in \Gamma_1$ and additionally for each $e \in E(G), c_e = k \cdot r_e$ holds for some k. In this section, we will show that, (a) If $G \in \Gamma_1$ is a $m \times n$ grid graph with single terminal vertex u, then MAD gives the minimum

delay path connecting the source vertex 0 and terminal u and (b) If $G \in \Gamma_2$ and contains n terminal vertices, then MAD provides n-approximation solution of problem (3).

A.1. Grid graphs with a single terminal

Here we assume graph G belongs to class Γ_1 and contains the only terminal vertex $u : S = \{0, u\}$. We scrutinize the performance of MAD on such kind of graphs and demonstrate the optimality of the path obtained.

First we show that, given two edges of a line (horizontal or vertical), the nearest to the root 0 is added to the tree first.

Claim 2. Given a graph $G \in \Gamma_1$ with single terminal vertex u. Let T be a partially constructed tree and vertices $u1(x_1, y), u2(x_2, y) \in T$ and $v1(x_1, y+1), v2(x_2, y+1) \notin T$ $(x_1 < x_2)$. Then

$$t_{v1}(T \cup \{(u1, v1)\}) + d_{(u1,v1)} < t_{v2}(T \cup \{(u2, v2)\}) + d_{(u2,v2)}.$$

Similar inequality holds for horizontal edges.

Proof. We omit this technical proof for the reason of space. □

Claim 3. Let $G \in \Gamma_1$ be a $m \times n$ grid graph with single terminal vertex u and T be a partially constructed tree. If vertex $v(x, y) \in T$ then all the vertices $w(i, j) \in T$ $(i \leqslant x, j \leqslant y)$.

Proof. proof is omitted due to page limitation. Details can be obtained in [28] □

Theorem 1. *Let $G \in \Gamma_1$ be a $m \times n$ grid graph with single terminal vertex u and P be the path constructed by MAD. Then*

$$t_u(P) = \min_{P_{0u}} t_u(P_{0u}).$$

Proof. proof is omitted due to page limitation. Details can be obtained in [28]. □

A.2. Grid graphs with $n \geqslant 2$ terminals

In this section, we consider operation of MAD on grid graphs with n terminal vertices and estimate its approximation ratio in a special case.

Claim 4. Given graph G containing n terminal vertices. Assume MAD connects terminals in the following order: $\{u_1, u_2, \ldots, u_k, \ldots, u_n\}$ and denote partially constructed tree spanning set $\{u_1, \ldots, u_k\}$ by T^k. Then

$$t^*(T^k) \leqslant \hat{t}_{u_k} + t^*(T^{k-1}).$$

Proof. proof is omitted due to page limitation. Details can be obtained in [28]. □

Remark 2. Let S be the set of terminal vertices and T be a Steiner tree for S. Claim 4 yields the following estimation for critical delay in T:

$$t^*(T) \leqslant \sum_{u \in S} \hat{t}_u.$$

Theorem 2. *Given $G \in \Gamma_2$ and $S = \{0, 1, \ldots, n\}$. Let T_{MAD} be the tree constructed by MAD and T_{opt} be an optimal tree spanning S. Then*

$$\frac{t^*(T_{MAD})}{t^*(T_{opt})} \leqslant n.$$

Proof. Use Remark 2. Let u_i be the i-th terminal attached to T_{MAD} and $T_{opt}(u_1, u_2, \ldots, u_i)$ be an optimal tree spanning set $\{u_1, u_2, \ldots, u_i\}$. As G belongs to class Γ_2 delays along all the paths connecting 0 and u_i are obviously equal. Also it is obvious that $\hat{t}_{u_i} \leqslant t^*(T_{opt}(u_1, u_2, \ldots, u_i))$. Hence,

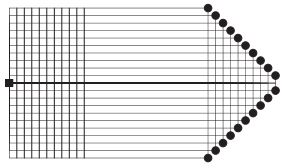

Fig. A.8. Graph G.

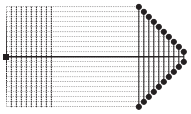

Fig. A.9. Tree T1.

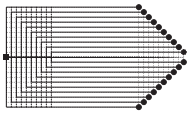

Fig. A.10. Tree T2.

$$t^*(T_{MAD}) \leqslant \sum_{i=1}^{n} t^*(T_{opt}(u_1, u_2, \ldots, u_i)) \leqslant \sum_{i=1}^{n} t^*(T_{opt}(u_1, u_2, \ldots, u_n)) = n \cdot t^*(T_{opt}(u_1, u_2, \ldots, u_n)),$$

Therefore,

$$\frac{t^*(T_{MAD})}{t^*(T_{opt})} \leqslant n.$$

□

A.3. Tightness of the bound

In this section we demonstrate that the accuracy estimate of MAD obtained in the previous section is tight. Consider grid graph $G \in \Gamma_2$ with $n = 2k$ terminals depicted in Fig. A.8 (similar example for graph with odd number of terminals can be easily constructed).

Let $r_e = c_e$ for each $e \in E(G)$, resistances of all the vertical and "short" horizontal edges equal ε and resistances of all the "long" horizontal edges equal L. Capacities of all the terminals equal zero. We specify values r_0, ε and L later.

Consider two Steiner trees: $T1$ (Fig. A.9) and $T2$ (Fig. A.10).

It is easy to prove that critical delays in trees $T1$ and $T2$ satisfy the following relations:

$$t^*(T1) = r_0 L + r_0(k^2 + 3k - 2)\varepsilon + \varepsilon^2\left(\frac{4}{3}k^3 + 2k^2 - 3k + \frac{1}{2}\right) + \varepsilon L(k^2 + 3k - 1) + \frac{L^2}{2} \tag{A.1}$$

$$t^*(T2) = 2kr_0 L + r_0(4k^2 - k - 1)\varepsilon + \varepsilon L(4k - 3) + \varepsilon^2\left(5k^2 - 6k + \frac{3}{2}\right) + \frac{L^2}{2} \tag{A.2}$$

Choose $\delta > 0$ such that $\delta \setminus L \ll 1$ and set $\varepsilon = \frac{\delta}{k^2 L}, r_0 = L^2$. In this case (A.1) and (A.2) imply:

$$\frac{t^*(T2)}{t^*(T1)} = 2k - O\left(\frac{1}{L}\right) < 2k, \tag{A.3}$$

and the ratio converges to $2k$ when $L \to \infty$. One may easily see that tree $T2$ can be constructed by MAD. Also it is obvious that $t^*(T_{opt}) \leqslant t^*(T2)$. So we have

$$\frac{t^*(T_{MAD})}{t^*(T_{opt})} \geqslant 2k - O\left(\frac{1}{L}\right), \tag{A.4}$$

and the following Remark is true.

Remark 3. Given $G \in \Gamma_2$ containing n terminal vertices. Then MAD provides n-approximation solution of problem (3).

References

[1] C. Albrecht, Global routing by new approximation algorithms for multicommodity flow, IEEE Trans. on CAD 20 (2001) 622–632.
[2] E. Bozorgzadeh, R. Kastner, M. Sarrafzadeh, Creating and exploiting flexibility in steiner trees, in: ACM/IEEE Design Automation Conference, ACM, 2001, pp. 195–198.
[3] R. Carden, C.K. Cheng, A global router using an efficient approximate multicommodity multiterminal flow algorithm, in: ACM/IEEE DAC, 1991, pp. 316–321.
[4] Y.A. Chen, Y.L. Lin, Y.C. Hsu, A new global router for asic design based on simulated evolution, in: Int. Symp. on VLSI Technology, Systems and Applications, 1989, pp. 261–265.
[5] M. Cho, D.Z. Pan, Boxrouter: a new global router based on box expansion and progressive ILP, in: Design Automation Conference, 2006, pp. 373–378.
[6] S. Chowdhury, G. Grewal, D. Banerji, Clustering hanan points to reduce VLSI interconnect routing times, in: IEEE Canadian Conference on Electrical & Computer Engineering, 2006, pp. 1223–1227.
[7] C. Chu, Y.C. Wong. Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design, in: International Symposium on Physical Design, 2005, pp. 28–35.
[8] C. Sechen, A. Sangiovanni-Vincentelli, The timberwolf placement and routing package, IEEE J. Solid-State Circuits SC-20 (1985) 510–522.
[9] H. Esbensen, A macro-cell global router based on two genetic algorithms, in: EDAC, 1994, pp. 428–433.
[10] N. Garg, J. Konemann, Faster and simpler algorithms for multicommodity flow and other fractional packing problems, in: 39th Ann. IEEE Symp. on Foundations of Computer Science, 1998, pp. 253–259.
[11] G. Grewal, X. Yu, M. Xu, Generating diverse pools of steiner trees for VLSI routing, in Canadian Conference on Electrical and Computer Engineering, 2005, pp. 677–685, 2005.
[12] X. Hong, T. Xue, J. Huang, C. Cheng, E.S. Kuh, Tiger: an efficient timing-driven global router for gate array and standard cell layout design, in: IEEE TCAD of Integrated Circuits and Systems, 2006, pp. 1323–1331.
[13] H. Hou, J. Hu, S.S. Sapatnekar, Non-Hanan routing, IEEE Trans. Comput Aided Des. Integr. Circuits Syst. 18 (4) (1999) 436–444.
[14] J. Hu, S.S. Sapatnekar, A timing-constrained simultaneous global routing algorithm, IEEE TCAD Integr, Circuits Syst. 21 (2002) 1025–1036.
[15] T.C.W.J.R. Gao, P.C. Wu, A new global router for modern designs, In: Asia and South Pacific Design Automation Conference, 2008, pp. 226–231.
[16] M.R. Kramer, J.V. Leeuwen, The complexity of wire routing and finding minimum area layouts for arbitrary VLSI circuits, Adv. Comput. Res. VLSI theory 2 (1984) 129–146.
[17] Y. Li, H. Gu, Fault tolerant routing algorithm based on the artificial potential field model in network-on-chip, Appl. Math. Comput. 217 (7) (2010) 3226–3235.
[18] M.D.F.W.M.M. Ozdal, Archer: a history-driven global routing algorithm, in: IEEE/ACM International Conference on Computer-Aided Design, 2007, pp. 488–495.
[19] L. McMurchie, C. Ebeling, Pathfinder: a negotiation-based performance-driven router for fpgas, in: FPGA, 1995, pp. 111–117.
[20] M.D. Moffitt, Maizerouter: engineering an effective global router, in: Asia and South Pacific Design Automation Conference, 2008, pp. 226–231.
[21] M.D. Moffitt, J.A. Roy, I.L. Markov, The coming of age of (academic) global routing, in: ISPD, 2008, pp. 148–155.
[22] W.C. poore, The transient response of damped linear networks with particular regards to wide-band amplifiers, J. Appl. Phys. 19 (1948) 55–63.
[23] J.A. Roy, I.L. Markov, High-performance routing at the nanometer scale, in: ICCAD, 2007, pp. 496–502.
[24] J.A. Roy, I.L. Markov, High-performance routing at the nanometer scale, in: 2007 International Conference on Computer-Aided Design (ICCAD 07), November 5–8, 2007, San Jose, CA, USA, 2007, pp. 496–502.
[25] J. Rubinstein, P. Penfield, M.A. Horowitz, Signal delay in RC tree networks, IEEE Trans. CAD Integr. Syst. 2 (1983) 201–211.
[26] R. Samanta, A.I. Erzin, S. Raha, Y.V. Shamardin, I.I. Takhonov, V.V. Zalyubovskiy, A provably tight delay-driven concurrently congestion mitigating global routing algorithm, in: International conference on Numerical Computations: Theory and Algorithms (NUMTA), Falerna, Italy, 2013, p. 119.
[27] F. Shahrokhi, D.W. Matula, The maximum concurrent flow problem, J. ACM 37 (1990) 318–334.
[28] I.I. Takhonov. On properties of a Dijkstra-based algorithm for constructing delay driven Steiner trees. Unpublished Manuscript, Novosibirsk State University, Russia.
[29] J.T. Yan, Dynamic tree reconstruction with application to timing-constrained congestion-driven global routing, IEEE Proc. Comput. Digital Tech. 153 (2) (2006) 117–129.
[30] J.-T. Yan, S.-H. Lin, Timing-constrained congestion-driven global routing, in: Asia and South Pacific Design Automation Conference, 2004, pp. 683–686.
[31] H. Youssef, S.M. Sait, Timing-driven global routing for standard-cell VLSI design, Comput. Syst. Sci. Eng. 14 (1999) 175–185.