Timing-Driven Steiner Trees are (Practically) Free

Charles J. Alpert,[†] Andrew B. Kahng,[‡] C. N. Sze[†] and Qinke Wang[‡]
[†] IBM Austin Research Lab, Austin, TX 78758
[‡]CSE Dept., Univ. of California at San Diego, La Jolla, CA 92093
E-mail: {alpert,csze}@us.ibm.com, {abk,qiwang}@cs.ucsd.edu

ABSTRACT

Traditionally, rectilinear Steiner minimum trees (RSMT) are widely used for routing estimation in design optimizations like floorplanning and physical synthesis. Since it optimizes wirelength, an RSMT may take a "non-direct" route to a sink, which may give the designer an unnecessarily pessimistic view of the delay to the sink. Previous works have addressed this issue through performance-driven constructions, minimum Steiner arborescence, and critical sink based Steiner constructions. Physical synthesis and routing flows have been reticent to adapt universal timing-driven Steiner constructions out of fear that they are too expensive (in terms of routing resource and capacitance). This paper studies several different performance-driven Steiner tree constructions in order to show which ones have superior performance. A key result is that they add at most 2%-4% extra capacitance, and are thus a promising avenue for today's increasingly aggressive performance-driven P&R flows. We demonstrate using a production P&R flow that timing-driven Steiner topologies can be easily embedded into an incremental routing subflow to obtain significantly improved timing (3.6% and 5.1% improvements in cycle time for two industry testcases) at practically no cost of wirelength or routability.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS—*Design Aids*; J.6 [**Computer Applications**]: COMPUTER-AIDED ENGINEERING

General Terms

Algorithms, Design, Performance

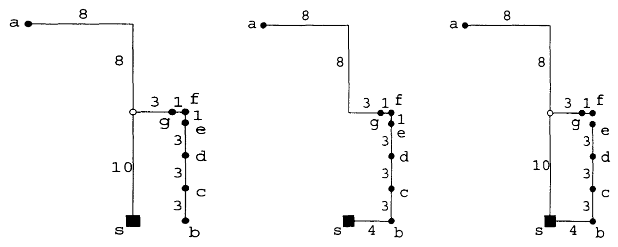
Keywords

Timing-Driven, Rectilinear Steiner Tree, Arborescence

1. INTRODUCTION

Recent technologies have made it necessary to quickly model the impact of wiring earlier in the design process. Such interconnect estimation is usually formulated as a rectilinear Steiner minimum tree (RSMT) problem. There are lots of heuristics which can calculate near-optimal RSMT quickly such as BI1S [10] and BOI [4]. GeoSteiner [21] is an optimal algorithm which has been empirically shown to solve the RSMT problem in reasonable time for relatively small trees. Recently, Chu and Wong [6] proposed a very efficient RSMT heuristic, namely FLUTE, based on the lookup table technique and that produces optimal solution if the number of sinks is less than 10.

There is a body of literature to construct timing-driven rectilinear Steiner trees (RSTs) that adapt to whichever sink is critical, e.g., [3] [11]. These works assume that timing constraints at the sinks are given and that the topology is adjusted accordingly. In the context of pre-routing design optimization, this approach will have three main problems. (1) Meaningful timing constraints may not be available early in the design process; any timing estimates will not necessarily be accurate. (2) Timing constraints may change during design at which point the tree will need to be recomputed. (3) Perhaps most importantly, this approach inherently causes *instability* in the design process. Whenever timing constraints change or blocks are moved slightly, this affects the topology of a critical-sink based RST construction. Having the topology change can affect the timing of other parts of the chip and make design convergence much more difficult.



(a) Tree with poor sink radius (b) RSMT (c) MRSA

Figure 1: Examples of various rectilinear Steiner trees

One might argue that a minimum rectilinear Steiner arborescence (MRSA) (which is an RST such that each source-to-sink path is a shortest path) [7] should always be used to estimate the wiring. However, the RSMT also serves to calculate the load seen by the driving pin. If a tree has significantly more wirelength than necessary, the load seen by the driver could end up being much higher than it would under a normal RSMT. That again might cause one to try to power up the driver or make other design modifications that may be deemed unnecessary.

This paper makes the following contributions: (1) We study the wirelength penalty induced by switching from RSMT to MRSA on a variety of designs and find that it is unexpectedly small, of the order of 2-4% on average. (2) We evaluate existing Steiner tree constructions to understand their abilities to trade-off between wirelength and radius, and propose new RST constructions that improve upon them. (3) We design an incremental detailed routing subflow to validate the benefit of timing-driven Steiner topologies, using commercial tools, libraries and design examples. Timing-critical nets are analyzed and timing-driven Steiner topologies are embedded to improve timing. We observe significant improvements in minimum cycle time (3.6% and 5.1% for two real designs) with negligible wirelength and routability impact.

2. PROBLEM STATEMENT

Let $V = \{v_0, v_1, ...v_n\}$ be a set of nodes where v_0 is the source node and the other n nodes are sinks. A Rectilinear Steiner Tree (RST) connects all $n+1$ nodes in V with either horizontal or vertical edges and some additional nodes. Let the path length from the source v_0 to a sink v_i on the edges in the Steiner tree be denoted by l_i. The *radius* of a routing tree T is defined as $R = \max_{i \leq i \leq n} l_i$. Some works [8] generate performance-driven trees by trading off radius with wirelength. The problem with this formulation is that it may still leave some sinks close to the source with much worse path length than necessary. E.g., the RST in Figure 1(a) minimizes the tree radius, however the path to sink b has long detour, which may produce a very unfavorable timing result if sink b is placed close to the source for some performance reasons.

To avoid this situation, we extend the concept of radius to each sink v_i as $R(i) = l_i/d_i$, where d_i is the Manhattan distance from v_0 to v_i, and we propose the **Bounded Radius-Ratio Steiner Minimum Tree (BRSMT)** problem: For a given parameter $\alpha \geq 1$, construct an RST such that $R(i) < \alpha, \forall\, v_i \in V$ with minimum total wirelength.

Several works [2, 8, 12, 13, 15, 16] have addressed similar forms of this problem statement. In [12], an algorithm is proposed to find a shallow-light $(1+\sqrt{2}\gamma, 1+\sqrt{2}/\gamma)$-tree, which is a spanning tree such that (1) the total wirelength is no more than $1+\sqrt{2}/\gamma$ times that of a minimum spanning tree; and (2) the path length from the source v_0 to a sink v_i is no more than $1+\sqrt{2}\gamma$ times the Manhattan distance between them, for all sinks. The Bound-Radius-Bounded-Cost (BRBC) algorithm [8] bounds the maximum source-to-sink path while minimizing total wirelength of the RST. Alpert, et al [2] combine the Prim's minimum spanning tree algorithm and Dijkstra's shortest path algorithm to trade off between the two. Kortsarz and Peleg [13] consider a different problem formulation such that the edge distance is always one and they try to construct a spanning tree with diameter ≤ 5. Most "shallow-light" works [2, 12, 16] are focusing on spanning trees. In [17], a new problem is defined as to find an RSMT such that the weighed sum of all source-to-sink path length is minimized.

3. COMPARISON OF RST ALGORITHMS

This section compares several well-known RST construction techniques in terms of the tree wirelength and radius-ratio for all the sinks. The algorithms are summarized in Table 1.

We extracted 4000 arbitrary nets from an industrial ASIC design to test the algorithms. The distribution of sink numbers and the average RSMT wirelength are shown in Table 2. BOI is used as the RSMT approximation, thus we can expect a little bit smaller wirelength for the optimal RSMT. All experiments are performed on a Sun Ultra Sparc 450 machine running in 400 MHz.

Results of various RSMT constructions are shown in Table 3. "max $w(T)$" and "avg $w(T)$" are the maximum and average wirelength increase in percentage, compared to the BOI wirelength. "max $R(v_i)$" and "avg $R(v_i)$" are the maximum and average radius-ratio among 4000 nets. The last column shows the total runtime, except for SERT since it is bundled with a routing package and its runtime could not be isolated.

Table 3 shows that PD1-1.0 and CL perform similarly. Due to the greedy nature of edge overlapping in PD algorithms, PD1-0.6 and PD1-0.8 are obviously outperformed by PD1-0.4 and PD1-1.0, and PD1-0.0 is not as good as BOI, either. However, the set of PD1 trees provide a better trade-off curve between wirelength and radius-ratio than other algorithms. Note that PD2 does not produce competitive results.

Figure 2 compares different RST implementations in terms of average wirelength and radius-ratio trade-off. We observe that PD1 outperforms other RST heuristics. Together with BOI, PD1-0.2, PD1-0.4 and PD1-1.0 forms a good trade-off between average radius-ratio and average wirelength.

Furthermore, we observe the following from Table 3:

- Some trees produced by RSMT heuristics have a radius as much as 6 times the optimal and they may exhibit poor performance.
- PD1-1.0, which is essentially a Steiner arborescence, suffers about a 4% wirelength penalty on average.
- PD1-1.0 produces just about the same quality of trees in terms of wirelength as CL but in slightly faster runtime.
- SERT produces terribly large wirelength even though it minimizes the Elmore delay for one parameter instance. However, the result is unfavorable when the timing or placement changes.
- PD1 provides a "smooth trade-off" except PD1-1.0 while for PD2, the trade-off does not correlate with the parameter p.
- k-SMT heuristics provide a good trade-off curve comparable to PD1. However, PD1 has a better control on the maximum wirelength and radius-ratio.
- The runtime of all implementations are similar.

Greedy RSMT [20]	A greedy RSMT algorithm that starts from the source node and greedily picks a node to expand the subtree.
BOI (RSMT) [4]	This is a simple RSMT approximation algorithm that starts with a minimum spanning tree and then iteratively adds a Steiner node by connecting a node to an edge and removes the longest edge on the cycle. BOI acts as a reference for wirelength comparison in the set.
PD [2]	A spanning tree algorithm combining Prim's and Dijkstra's algorithms to balance the wirelength and radius optimization. There are two versions described in [2], namely PD1 and PD2 in this paper. The algorithm framework is similar to the Prim's algorithm. However, in PD1, we consider both source-to-sink path length and edge length when choosing the next node to add into the tree with the balance controlled by a parameter c. In PD2, a node v_i is picked when $(l(i,j)^p + \sum_{(k,l)\in path} l(k,l))^{1/p}$ is minimized. The rest of the paper denotes the algorithms by "PD1-c" and "PD2-p" to indicate different parameter used. PD produces a better wirelength and radius trade-off than BRBC [8] and KRY [13].
CL (MRSA) [9]	An MRSA heuristic that generalizes the algorithm in [19], which simply assumes that all sinks are on the first quadrant. The MRSA heuristic guarantees that the radius-ratio at every sink is exactly one. We would like to see how much wirelength penalty an MRSA pays. There are other MRSA algorithms (e.g., A-tree) which produce better wirelength optimization than CL but the algorithm runs in $O(nlgn)$ with a small constant.
SERT [3]	The Steiner Elmore Routing Tree aims to minimize the maximum source-to-sink Elmore delay over all the sinks. Starting with only the source node, it greedily includes a node so that the Elmore delay at any sink is minimized. We will show that the timing-driven algorithm is not stable because the performance changes with the timing.
k-SMT [18]	The k-th moment Steiner minimum tree (k-SMT), defined in [18], is shown to be be a trade-off between SMT and MRSA because 0-SMT is SMT and ∞-SMT is MRSA. Therefore, we tend to speculate the k-SMT provides a good trade-off between wirelength and radius-ratio. We implement a heuristic according to [1, 18], which is based on the Dijkstra's shortest path algorithm.

Table 1: Overview of selected RST constructions

#sinks	#nets	avg $w(RSMT)$	#sinks	#nets	avg $w(RSMT)$
4	734	6372777	11	179	8256610
5	285	6304983	12	90	8137800
6	527	3060273	13	81	7994200
7	179	7391604	14	40	10913850
8	209	7128543	15	58	9629938
9	171	5510011	16	1380	1969117
10	67	9070630	All	4000	4755137

Table 2: Distribution of sink numbers and the corresponding average BOI wirelength

Algorithm	Wirelength Increase		Radius-Ratio		CPU
	max	avg	max	avg	(s)
Greedy	35.32%	0.89%	6.384	1.084	8.7
BOI	0%	0%	6.030	1.090	11.4
PD1-0.0	21.60%	0.50%	5.980	1.093	8.9
PD1-0.2	31.30%	0.99%	3.345	1.059	8.7
PD1-0.4	35.94%	2.47%	2.057	1.037	8.6
PD1-0.6	53.01%	4.67%	1.588	1.021	8.6
PD1-0.8	76.76%	7.78%	1.239	1.009	8.4
PD1-1.0	48.53%	4.16%	1.000	1.000	9.4
PD2-1.0	48.53%	4.16%	1.000	1.000	8.9
PD2-2.0	70.83%	3.34%	2.540	1.039	9.3
PD2-3.0	46.21%	1.46%	3.614	1.060	9.3
PD2-4.0	69.57%	1.06%	3.676	1.071	9.4
PD2-5.0	69.57%	0.84%	4.085	1.078	9.4
PD2-6.0	69.57%	0.70%	4.097	1.082	9.1
PD2-8.0	69.57%	0.69%	5.980	1.086	9.7
PD2-10	69.57%	0.71%	5.980	1.087	9.2
PD2-12	69.57%	0.74%	5.980	1.090	9.5
PD2-14	21.69%	0.90%	5.980	1.092	9.3
CL	48.53%	4.19%	1.000	1.000	11.8
SERT	1328%	276%	2.66	1.002	NA
0.5-SMT	35.94%	1.34%	3.24	1.054	8.5
1.0-SMT	46.25%	2.735%	2.480	1.034	8.8
5.0-SMT	48.81%	4.39%	1.103	1.001	9.9

Table 3: Maximum and average wirelength increases and radius-ratios of selected RSA implementations

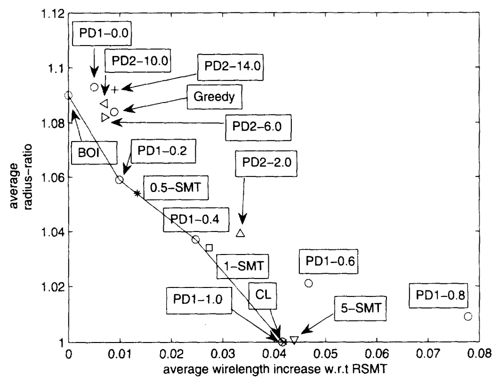


Figure 2: Comparison of different RST implementations in terms of average wirelength and radius-ratio trade-off

4. TWO NEW BRSMT HEURISTICS

From the previous section, two questions arise: (1) Although the RSMT heuristic, PD1, provides a better trade-off than other algorithms, can we do better? (2) Since PD1 appears to offer "stable, timing-driven topologies" with little wirelength overhead, will the construction be of benefit in a real timing-driven layout flow? We answer these questions in this and the following sections.

We propose two simple yet effective heuristics to deal with the BRSMT problem. Our H1 heuristic starts with an RSMT with minimum wirelength. Since it may have a terrible radius-ratio, we check each sink for the radius-ratio. If the ratio is larger than a given α, we disconnect the sink from the tree. Then, we try to reconnect all disconnected sinks with a direct path.

In Section 3, we find that PD1-1.0/MRSA produces only 4% more wirelength than RSMT heuristics while minimizing the maximum radius-ratio. Our second heuristic (H2) is then based on PD1-1.0/MRSA and tries to further improve the wirelength while relaxing the radius-ratio. We apply the BOI style edge addition and removal procedure while keeping track of the radius-ratio. An operation will be accepted only when the radius-ratio of the resultant tree is not larger than the parameter α.

α	Wirelength Increase		Radius-Ratio	
	max	avg	max	avg
1.0	96.27%	15.75%	1.000	1.000
1.2	76.38%	8.51%	1.200	1.018
1.4	60.65%	4.58%	1.400	1.037
1.6	55.85%	2.79%	1.600	1.052
1.8	43.43%	1.49%	1.800	1.062
2.0	41.14%	0.87%	2.000	1.071
2.2	40.61%	0.59%	2.200	1.076
2.4	34.74%	0.35%	2.397	1.080
2.6	34.24%	0.24%	2.594	1.082
2.8	25.23%	0.13%	2.792	1.084
3.0	22.67%	0.07%	3.000	1.086

Table 4: Results of H1 with various values of α

α	Wirelength Increase		Radius-Ratio	
	max	avg	max	avg
1.2	47.55%	3.88%	1.200	1.004
1.4	47.55%	3.60%	1.400	1.009
1.6	40.97%	3.41%	1.597	1.013
1.8	40.97%	3.28%	1.800	1.017
2.0	40.97%	3.18%	2.000	1.021
2.2	40.97%	3.12%	2.197	1.024
2.4	40.97%	3.07%	2.393	1.026
2.6	40.97%	3.04%	2.591	1.028
2.8	40.97%	3.03%	2.800	1.029
3.0	40.97%	3.00%	3.000	1.031

Table 5: Results of H2 with various values of α

Designs	design1	design2	design3
#Nets	146k	330k	223k
Total WL (RSMT)	124057096	610531613	88485247
Total WL (MRSA)	124453400	619629806	89660129
WL Increase (%)	0.32	1.49	1.33
Designs	design4	design5	design6
#Nets	168k	342k	342k
Total WL (RSMT)	116000653	154572475	57887506
Total WL (MRSA)	116631947	155663525	59050751
WL Increase (%)	0.54	0.71	2.01

Table 6: RSMT and MRSA wirelength for six designs

Experimental results of our new heuristics H1 and H2 are given in Table 4 and 5, respectively. According to the tables, H1 and H2 provide a smooth trade-off between wirelength and radius-ratio. Compared to H2, H1 generates trees with smaller wirelength when $\alpha > 1.6$ because the trees are constructed based on RSMT. H2 tends to be more stable in terms of wirelength, though it maintains a smaller average radius-ratio.

We are interested in understanding how much wirelength penalty is incurred when one replaces RSMT with MRSA for wiring estimation in real designs. We construct both MRSA and RSMT for 6 industrial designs and report the total wirelength. The results are shown in Table 6. For all the designs, the wirelength increase of MRSA over RSMT is less than 3%. This shows that the penalty of using a timing-driven Steiner construction is indeed small.

5. VALIDATION IN PRODUCTION P&R

We now demonstrate the benefit of timing-driven Steiner topologies by implementing an incremental detailed routing sub-flow using commercial tools, libraries and design examples. The goal of this experiment is to show that timing-driven Steiner trees can achieve better timing with a very small wirelength penalty. To achieve this, we extract timing-critical nets from routed designs and embed modified topologies to improve timing.

We experiment with two designs from *OpenCores*, which are synthesized using *Encounter RTL compiler v4.2* in UMC 90nm li-

Design	Utilization	MCT	WL	#Vias	#Cells	#Nets
AES	70%	2.490ns	8.784	186068	25187	25446
JPEG	70%	3.605ns	27.85	836543	135652	135672

Table 7: Design characteristics of two benchmarking circuits

brary with tight timing constraints, and then floorplanned, placed and routed with timing-driven mode in *Cadence First Encounter* (v04.10). The design characteristics are summarized in Table 7.

We perform static timing analysis (STA) using *Synopsys Prime-Time* (v.2004.12) after RC extraction to identify timing-critical paths in the design. We analyze all the nets along the top critical paths with a path delay larger than $(1-\gamma)$ of the minimum cycle time (MCT) from a timing report, and query the slack and source-to-sink delay for each sink of a net through sockets from a concurrently running PrimeTime. If the sink with the smallest slack (i.e., the sink with the highest timing priority) in a net does not match the sink with the smallest ratio of source-to-sink delay and distance (i.e., the sink actually routed with the highest timing-driven effort), we construct timing-driven Steiner tree for the net using the PD1 algorithm described in Section 3.3.

To enforce the new timing-driven Steiner topology, we remove routed wire segments of the net from the DEF file and substitute it with VPIN (virtual pin) and SUBNET definitions. A VPIN specifies the location of a Steiner point, which separates the net into subnets as specified by SUBNET statements. To ensure wires of different sub-nets are connected after ECO routing, we set the width and height of virtual pins to the minimal track width. We do not specify the layer of the virtual pin so that ECO routing has more flexibility. We then use *Cadence WarpRoute* (v2.4.44) to perform routing in incremental mode (for selected nets only). The flow is iterated until the relative timing improvement is less than $\gamma/10$.

The results are summarized in Table 8. For each testcase, we run the validation flow with PD1-0.2, PD1-0.4 or PD1-1.0 algorithm and a variety of values of $\gamma \in (0,\ 0.1)$. We show the total number of selected and rerouted nets in the fourth column. MCT after the flow is reported to measure performance improvement, together with routed wirelength, the number of vias and the number of violations of WarpRoute's ECO routing results.

The results show the incremental flow reduces MCT by 5.1% and 3.6%, respectively for AES and JPEG, with negligible wirelength increase, since the number of nets selected and rerouted are relatively small compared to the total number of nets of the design. However, routing of these timing critical nets greatly affect design performance and thus timing-driven routing efforts should be taken and carefully traded off with wirelength-driven efforts. Because of routing constraints enforced by VPINs as well as existing routing obstacles, ECO routing may end up with a few violations. This can be improved by cautiously selecting nets for ECO routing and defining VPINs according to existing routing congestion.

6. CONCLUSIONS

In this work, we have studied different RST constructions in terms of the wirelength and radius-ratio trade-off. Perhaps the most surprising result uncovered is that minimum Steiner arborescences actually are not as expensive as expected in wirelength for obtaining optimal path lengths for all sinks: the average degradation in wirelength is just around 4%. If one is not willing to even pay that penalty, other algorithms which obtain better radius than the RSMT but at a smaller wirelength cost are presented. We also demonstrate using a production P&R flow that the results of our study can be straightforwardly brought into practical application: we embed the PD1 timing-driven Steiner topology construction into an incremental routing subflow using commercial P&R tools to obtain significantly improved timing (between 3.6% and 5.1% reductions in cycle time) at practically no cost of wirelength or routability.

Design	PD1-c	γ	#Nets	MCT (ns)	MCT (%)	WL (e5um)	#Vias	#Vios
AES	0.2	0.01	14	2.375	4.86	8.784	186120	1
		0.05	17	2.375	4.83	8.784	186124	0
		0.09	20	2.375	4.86	8.784	186134	0
	0.4	0.01	14	2.399	3.78	8.784	186116	0
		0.05	17	2.399	3.79	8.784	186123	0
		0.09	20	2.396	3.94	8.784	186124	0
	1.0	0.01	14	2.379	4.68	8.785	186133	2
		0.05	18	2.374	4.87	8.785	186147	0
		0.09	22	2.370	5.05	8.785	186149	0
JPEG	0.2	0.01	130	3.595	0.28	27.86	836647	0
		0.03	917	3.520	2.40	27.58	828600	0
		0.05	1733	3.520	2.41	27.45	823738	0
		0.07	3134	3.483	3.49	27.18	814541	0
	0.4	0.01	142	3.590	0.42	27.85	836567	0
		0.03	917	3.520	2.40	27.58	828615	1
		0.05	1762	3.519	2.44	27.45	823801	0
		0.07	3134	3.483	3.50	27.18	814544	0
	1.0	0.01	131	3.591	0.38	27.86	836686	0
		0.03	915	3.517	2.50	27.58	828663	0
		0.05	1786	3.513	2.62	27.44	823711	0
		0.07	3151	3.480	3.59	27.18	814497	0

Table 8: Results of incremental detailed routing flow with PD1-0.2, PD1-0.4 or PD1-1.0 algorithm and a variety of values of γ's for circuits AES and JPEG

7. REFERENCES

[1] Personal communication with Dr. Weiping Shi.

[2] C. J. Alpert, T. C. Hu, J. H. Huang, A. B. Kahng, and D. Karger. Prim-dijkstra tradeoffs for improved performance-driven routing tree design. *IEEE TCAD*, 14(7):890–896, July 1995.

[3] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins. Near-optimal critical sink routing tree constructions. *IEEE TCAD*, 14(12):1417–36, Dec. 1995.

[4] M. Borah, R. M. Owens, and M. J. Irwin. An edge-based heuristic for Steiner routing. *IEEE TCAD*, 13(12):1563–1568, Dec. 1994.

[5] M. Borah, R. M. Owens, and M. J. Irwin. A fast and simple Steiner routing heuristic. *Discrete Applied Mathematics*, 90:51–67, 1999.

[6] C. Chu and Y.-C. Wong. Fast and accurate rectilinear steiner minimal tree algorithm for vlsi design. In *ISPD*, pp. 28–35, 2005.

[7] J. Cong, A. B. Kahng, and K.-S. Leung. Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to vlsi physical design. *IEEE TCAD*, 17(1):24–39, Jan. 1999.

[8] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Provably good performance-driven global routing. *IEEE TCAD*, 11(6):739–752, June 1992.

[9] J. Cordova and Y. H. Lee. A heuristic algorithm for the rectilinear Steiner arborescence problem. *Tech. Report, CIS Department, Univ. of Florida*, (TR-94-025), 1994.

[10] J. Griffith, G. Robins, J. Salowe, and T. Zhang. Closing the gap: Near-optimal Steiner trees in polynomial time. *IEEE TCAD*, 13:1351–1365.

[11] A. B. Kahng and G. Robins. *On optimal interconnections for VLSI.* Kluwer Academic Publishers, Boston, MA, 1995.

[12] S. Khuller, B. Raghavachari, and N. Young. Balancing minimum spanning trees and shortest-path trees. *ACM-SIAM SODA*, pp. 243–250, 1993.

[13] G. Kortsarz and D. Peleg. Approximating shallow-light trees. *ACM-SIAM SODA*, pp. 103–110, 1997.

[14] P. Madden. BOI source code. http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/ Madden_BOI.html.

[15] M. Marathe, R. Ravi, R. Sundaram, S. Ravi, and D. Rosenkrantz. Bicriteria network design problems. In *Intl. Colloquium Automata Languages and Programming (ICALP'95)*, pp. 487–498, 1995.

[16] J. Naor and B. Schieber. Improved approximations for shallow-light spanning trees. *FOCS*, page 536, 1997. IEEE Computer Society.

[17] S. Peyer, M. Zachariasen, and D. G. J&#248;rgensen. Delay-related secondary objectives for rectilinear Steiner minimum trees. *Discrete Appl. Math.*, 136(2-3):271–298, 2004.

[18] W. Qiu and W. Shi. Minimum moment Steiner trees. *SODA*, pp. 488–495, 2004.

[19] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor. The rectilinear Steiner arborescence problem. *Algorithmica*, 7:277–288, 1992.

[20] J. Soukup. Uniform steiner tree representation from the floor planning to the final routing. Unpublished manuscript.

[21] D. M. Warme, P. Winter, and M. Zachariesen. Exact algorithms for plane Steiner tree problems: A computational study. pp. 81–116, 2000.