Fast Path-Based Timing Analysis for CPPR

Tsung-Wei Huang*, Pei-Ci Wu†, and Martin D. F. Wong‡
*twh760812@gmail.com, †peiciwu@gmail.com, ‡mdfwong@illinois.edu
Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, IL, USA

***Abstract*—Common-path-pessimism removal (CPPR) is a pivotal step to achieve accurate timing signoff. Unnecessary pessimism might arise quality-of-result (QoR) concerns such as reporting worse violations than the true timing properties owned by the physical circuit. In other words, signoff timing report will conclude a lower clock frequency at which circuits can operate than actual silicon implementations. Therefore, we introduce in this paper a fast path-based timing analysis for CPPR. Unlike existing approaches which are dominated by explicit path search, we perform implicit path representation which yields significantly smaller search space and faster runtime. Specifically, our algorithm is superior in both space and time saving, from which the memory storage and important timing quantities are available in constant space and constant time per path during the search. Experimental results on industrial benchmarks released from TAU 2014 timing analysis contest have shown that our algorithm won the first place and achieved the best result in terms of accuracy and runtime over all participating teams.**

I. Introduction

Static-timing analysis (STA) is a pivotal step of the integrated-circuit (IC) design flow in order to verify the timing behaviors. Conventional STA tools rely on conservative dual-mode operations to estimate early-late and late-early path slacks [7]. This mechanism, however, imposes unnecessary pessimism due to the consideration of delay variation along common segments of clock paths, as illustrated in Figure 1. Unnecessary pessimism may lead to tests being marked as failing whereas in actuality they should be passing. Thus designers and optimization tools might be misled into an over-pessimistic timing report [17]. Therefore, the recent 2014 TAU computer-aided design (CAD) contest has aimed to seek novel ideas for accurate and fast common-path-pessimism removal (CPPR), so as to prevent the true timing properties of circuits from being skewed [12].

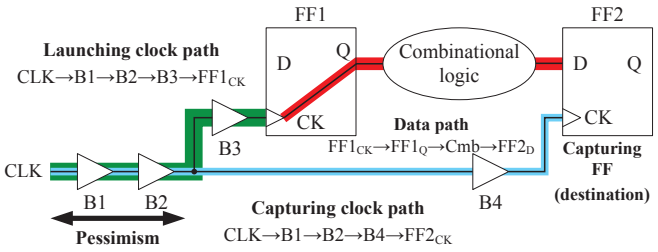

Figure 1. Clock network pessimism incurs in the common path between the launching clock path and the capturing clock path.

Unfortunately, it has been reported that CPPR is a tough task in current STA tools [3]. The real challenge is the amount of pessimism that needs to be removed is path-specific. The most critical path prior to pessimism removal is not necessarily reflective of the true counterpart [19]. Traditional graph-based timing analysis (GBA) is no longer suitable for CPPR because it deals only with the worst timing quantities on each endpoint other than path-by-path timing update [8], [9]. Therefore, path-based timing analysis (PBA) has been alternatively employed during STA in order to configure path-specific or less-pessimistic features such as CPPR and advanced-on-chip-variation (AOCV) derating into the timer [14]. It is obvious an optimal solution to CPPR may require analysis of all paths in the design or explore an exponential number of paths which could be computationally impractical. An intelligent algorithm that can quickly peel out a set of true critical paths is relatively desirable.

Our contributions are summarized as follows. 1) We have developed a new path ranking algorithm that can exactly and quickly peel out a set of true critical paths with CPPR. 2) The proposed algorithm is extremely fast. It performs implicit path representation along with two efficient data structures, namely suffix tree and prefix tree, and yields significant savings in both memory usage and cpu runtime. 3) Our algorithm has been verified and received the first place in 2014 TAU CAD contest. The final contest results have demonstrated the superior performance of our algorithm in terms of accuracy and runtime. Last but not least, we believe our algorithm can be beneficial to accelerate the signoff timing closure, on which up to 40% of the design flow are typically spent [16].

II. 2014 TAU CAD Contest Problem

The flow of 2014 TAU CAD contest on CPPR is shown in Figure 2. Given a set of input files, participants are requested to develop their own timers that report post-CPPR (i.e., slacks after CPPR) critical tests and paths. The number of tests and paths to be reported is controlled via a set of pre-defined inputs to the tool.

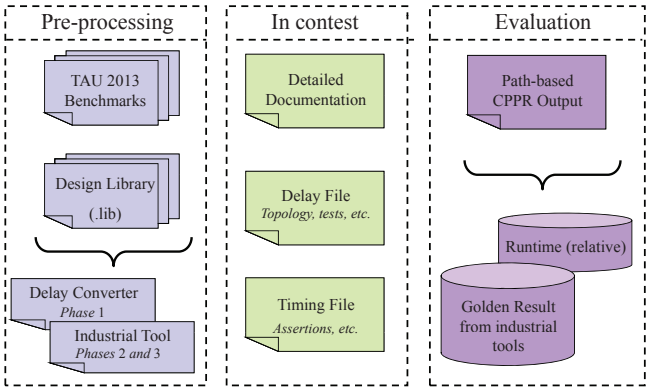

Figure 2. Guidelines of 2014 TAU CAD contest on CPPR.

Each design consists of a delay input file and a timing input file. The former describes the timing behavior and topology of the circuit while the later defines the initial operating condition. The timer is fed by a command line that allows the following user-defined parameters: 1) the type of test (-hold or -setup), being either hold or setup; 2) the number of test (-numTest) to be output; 3) the number of paths (-numPath) to be printed per output test. The output file contains the a set of paths for each test type specified in accordance with the input commands. The timer needs to identify the top "-numTest" critical test according to the respective worst post-CPPR slack and then print the top "-numPath" critical paths per identified test. The more negative the post-CPPR of a path is, the higher criticality the path has. For more details, we refer readers to the contest education file, which can be reached on the contest website [3].

978-1-4799-6278-5/14/$31.00 ©2014 IEEE

III. Algorithm

The overall of our algorithm is presented in Algorithm 1. It consists of of two stages: *lookup table preprocessing* and *pessimism-free path search*. The goal of the first stage is to tabulate the common path information for quick lookup of common-path pessimism. The goal in the second stage is to identify the top-k (i.e., -numPath) critical paths in a pessimism-free graph derived from each test.

Algorithm 1: CPPR(t, k)

Input: test t, path count k
Output: solution set Ψ of the top-k critical paths

```
1 BuildCreditLookupTable();
2 G_p ← pessimism-free graph for the test t;
3 Ψ ← GetCriticalPath(G_p.source, G_p.destination, k);
4 return Ψ;
```

A. Lookup Table Preprocessing

The pessimism between a source-target pair of flip-flops (FFs) incurs in the common path from the clock tree root to the clock tree node to which they reconverge along the upstream traversal. In graph theory, the clock reconverging node of a node pair in the clock tree is equivalent to the lowest common ancestor (LCA) of the node pair. The arrival time information of each node in the clock tree can be precomputed and therefore the pessimism of two nodes can be obtained immediately once their LCA is known. Many state-of-the-art LCA algorithms have been invented over the last decades. The table-lookup algorithm by [6] is employed as our LCA engine due to its simplicity and efficiency.

B. Formulation of Pessimism-Free Graph

When performing the hold or setup test, the required arrival time of the testing FF and the amount of pessimism between each source FF and the testing FF remain fixed regardless of which data path is being considered. Precisely speaking, the way a data path passing through plays the most vital role in determining the final slack values. In order to facilitate the path search without interleaving between slack computation and pessimism retrieval, we construct a pessimism-free graph for a given test. An example pessimism-free graph derived from a test is shown in Figure 3.

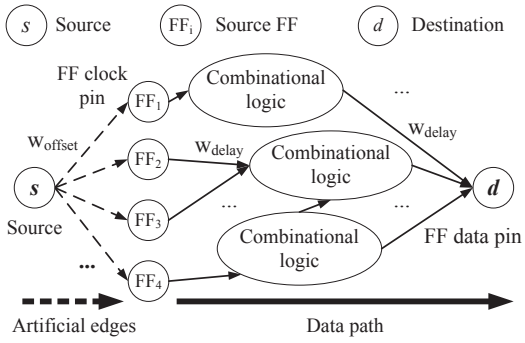

Figure 3. Derivation of the pessimism-free graph from a given test.

The intuition is to separate out the constant portion of the post-CPPR slack by an artificial edge such that the search procedure can focus on the rest portion which totally depends on the way data paths passing through. Each artificial edge comes with an offset weight which consists of the amount of pessimism between each lunching FF and the capturing FF as well as the value of the required arrival time at the capturing FF [13]. It is clear that the cost of any source-destination path (i.e., sum of all edge weights) in the pessimism-free graph is equivalent to the post-CPPR slack of the corresponding data path which is obtained by removing the artificial edge.

C. Generation of Top-k Critical Paths

The problem of identifying the top-k critical paths for a given test is equivalent to the path ranking problem applied to the pessimism-free graph. A number of state-of-the-art algorithms for path ranking have been proposed over the past years [5], [10], [15], [18]. The best time complexity acquired to date is $O(m + nlogn + k)$ from the well-know Eppstein's algorithm [10]. However, it relies on sophisticated implementations of heap trees which results in little practical interests. Moreover, most existing approaches are developed for general graphs and lack a compact and efficient specialization to certain graphs such as the directed-acyclic circuit network. The key contribution of this work is the new path ranking algorithm we have developed for this contest. In a high-level sketch, we propose two complementary data structures, namely suffix tree and prefix tree, to represent the search space of the path ranking. Figure 4 shows the concept of our implicit path representation. The suffix tree represents the shortest path tree rooted at the destination node of the pessimism-free graph. The prefix tree is a tree order of non-suffix-tree edges such that each tree node represents the path being deviated on the corresponding edge from its ordinary trace of the shortest route. As a result, each data path can be implicitly stored by the two data structures, such that the memory usage and the search time can be significantly reduced to constant time per path during the search. More algorithmic details can be referred to [13].

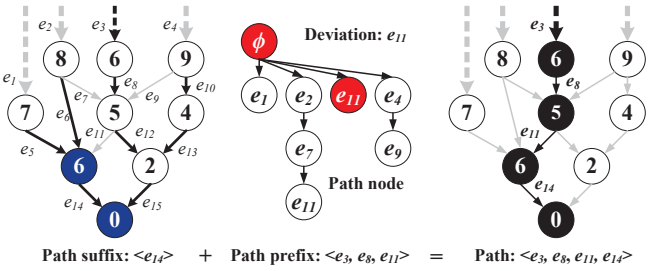

Figure 4. Implicit path representation using suffix tree and prefix tree.

IV. Contest Evaluation

The evaluation of 2014 TAU CAD contest is based on 1) *relative accuracy* to a "golden" reference from an industrial timer and 2) *relative runtime* of each participating timer. During the contest, timers are encouraged to employ multi-threaded programming interface [1]. Up to eight concurrent threads are supported by the contest machine which is configured with 8X Intel(R) Xeon CPU E7-8837 @2.67GHz. The benchmarks are well-known industrial circuits (e.g., s27, s510, systemcdes, wb_dma, pci_bridge32, vga_lcd, etc.) that have been released to the public domain for research purpose. Statistics of these benchmarks are listed in Table I. Each output file is uniquely identified by the benchmark. The overall score of a timer is the average across each output's weighted accuracy. For each benchmark, multiple output files that emphasize both the importance of finding the most critical tests and the set of paths that cause timing violations of each test are generated [3].

TABLE I
CONTEST BENCHMARKS AND STATISTICS [3].

Circuit	$\|V\|$	$\|E\|$	$\|I\|$	$\|O\|$	# Tests	# Paths
s27	109	112	6	1	6	9
s344	574	658	11	11	30	71
s349	598	682	11	11	30	71
s386	570	701	9	7	12	27
s400	708	813	5	6	42	77
s510	891	1091	21	7	12	99
s526	933	1097	5	6	42	44
s1196	1928	2400	16	14	36	478
s1494	2334	2961	10	19	12	105
systemcdes	10826	13327	132	65	380	41436
wb_dma	14647	17428	217	215	1374	158
tv80	18080	23710	14	32	838	19227963
systemcaes	23909	29673	260	129	2500	13069928
mem_ctrl	36493	45090	115	152	3754	62938
ac97_ctrl	49276	55712	84	48	9370	148
usb_funct	53745	66183	128	121	4392	129854
pci_bridge32	70051	78282	162	207	16450	17296
aes_core	68327	86758	260	129	2528	21064
des_perf	330538	404257	235	64	19764	1682
vga_lcd	449651	525615	89	109	50182	5281
Combo2	260636	284091	170	218	29574	62938
Combo3	181831	284091	353	215	8294	129854
Combo4	778638	866099	260	169	53520	19227963
Combo5	2051804	2228611	432	164	79050	19227963
Combo6	3577926	3843033	486	174	128266	19227963
Combo7	2817561	3011233	459	148	109568	19227963

$|V|$: # of nodes. $|E|$: # of edges. $|I|/|O|$: # of primary inputs/outputs.
Tests: # of setup tests and hold tests. # Paths: max # of data paths per test.

A. Contest Result

Our program is named as "UI-Timer" and it is the first place winner of 2014 TAU CAD contest. "Lightspeed" and "iTimerC" are the team names of the timers which received the second place and the third place, respectively. Due to the similar score with the third place timer, a special award of honorable mention is given to the four place timer "TimeKeepers". The overall comparison results in terms of raw accuracy, runtime value, and final average score are listed in Figure 5, Figure 6, and Figure 7. The raw accuracy is interpreted by a numerical value ranging from 0.0 to 1.0 such that the value 1.0 denotes that a timer obtains the full accuracy while the value 0.0 shows that the result of a timer is far from correctness. The runtime is measured by the minutes and total hours that are spent by a timer on finishing the input benchmarks. The final score is a weighted score combining raw accuracy and runtime into a numerical value. In general, the larger the value of the final score a timer has, the better the result it obtained [3].

It can be observed that our timer, UI-Timer, outperformed all participating timers in terms of accuracy and runtime. Our program is very reliable in accomplishing all benchmarks while the timer "LightSpeed" crashed in Combo6 and Combo7 and failed to generate any interpretable data. Although the timers "iTimerC" and "TimeKeeprs" successfully finish all benchmarks, they turn out to be less promising in either accuracy or runtime. We can see that from Figure 5 "iTimerC" loses the accuracy in Combo6 and Combo7 which might be a result of program errors or algorithmic shortcomings. On the other hand, "TimeKeeprs" demands an extremely high runtime complexity in order to achieve reliable and accurate results. The largest difference can be discovered in Combo4v2.setup.25000.1 where our timer is $\times$5.7 faster than "TimeKeeprs".

Benchmark.testType.numTests.numPaths	iTimerC	LightSpeed	TimeKeepers	UI-Timer
Combo2v2.hold.10000.15	1.00	0.85	1.00	1.00
Combo2v2.setup.10000.15	1.00	0.85	1.00	1.00
Combo2v2.setup.20000.1	1.00	0.85	1.00	1.00
Combo3v2.setup.6000.20	1.00	1.00	1.00	1.00
Combo4v2.hold.15000.15	1.00	1.00	1.00	1.00
Combo4v2.hold.25000.1	1.00	1.00	1.00	1.00
Combo4v2.setup.15000.15	1.00	1.00	1.00	1.00
Combo4v2.setup.25000.1	1.00	1.00	1.00	1.00
Combo5v2.hold.20000.15	1.00	1.00	1.00	1.00
Combo5v2.hold.35000.1	1.00	1.00	1.00	1.00
Combo5v2.setup.20000.15	1.00	1.00	1.00	1.00
Combo5v2.setup.35000.1	1.00	1.00	1.00	1.00
Combo6v2.hold.35000.15	1.00	0.00	1.00	1.00
Combo6v2.hold.50000.1	1.00	0.00	1.00	1.00
Combo6v2.setup.35000.15	0.80	0.00	1.00	1.00
Combo6v2.setup.50000.1	0.80	0.00	1.00	1.00
Combo7v2.hold.35000.20	1.00	1.00	1.00	1.00
Combo7v2.hold.50000.1	0.79	0.00	1.00	1.00
Combo7v2.setup.35000.20	0.80	1.00	1.00	1.00
Combo7v2.setup.50000.1	0.80	1.00	1.00	1.00

Figure 5. Raw accuracy of the top four timers in 2014 TAU CAD contest [3].

Benchmark.testType.numTests.numPaths	iTimerC	LightSpeed	TimeKeepers	UI-Timer
Combo2v2.hold.10000.15	1.01	0.18	0.23	0.45
Combo2v2.setup.10000.15	1.34	0.22	0.29	0.56
Combo2v2.setup.20000.1	0.38	0.08	0.15	0.37
Combo3v2.setup.6000.20	0.76	0.12	0.16	0.17
Combo4v2.hold.15000.15	3.56	6.82	7.73	1.59
Combo4v2.hold.25000.1	1.61	3.17	7.47	2.04
Combo4v2.setup.15000.15	13.49	2.70	10.49	1.85
Combo4v2.setup.25000.1	5.27	5.39	9.73	1.70
Combo5v2.hold.20000.15	8.58	2.62	22.47	5.28
Combo5v2.hold.35000.1	3.98	6.53	21.87	6.93
Combo5v2.setup.20000.15	26.43	4.08	24.92	5.86
Combo5v2.setup.35000.1	13.15	4.39	23.33	6.50
Combo6v2.hold.35000.15	13.31	n/a	24.44	12.09
Combo6v2.hold.50000.1	5.15	n/a	26.69	14.00
Combo6v2.setup.35000.15	34.05	n/a	27.66	15.62
Combo6v2.setup.50000.1	9.15	n/a	24.60	13.88
Combo7v2.hold.35000.20	16.87	6.24	58.69	13.25
Combo7v2.hold.50000.1	4.78	n/a	54.93	15.37
Combo7v2.setup.35000.20	61.65	8.80	62.72	13.09
Combo7v2.setup.50000.1	30.64	5.30	59.18	13.72
Total (hours)	4.25	0.94	7.80	2.41

Figure 6. Runtime of the top four timers in 2014 TAU CAD contest [3].

B. Beyond the Contest

In order to demonstrate the scalability of our program, an extra evaluation on the three largest cases, Combo5, Combo6, and Combo7 was made on a large distributed system. We evenly partitioned the test sets into groups with respect to the number of parallel cores being invoked. The application programming interface (API) provided by OpenMPI 1.6.5 is used as our message passing interface for

Benchmark.testType.numTests.numPaths	iTimerC	LightSpeed	TimeKeepers	UI-Timer
Combo2v2.hold.10000.15	0.81	1.90	1.87	1.20
Combo2v2.setup.10000.15	0.80	1.95	1.89	1.22
Combo2v2.setup.20000.1	0.98	2.31	1.73	0.99
Combo3v2.setup.6000.20	0.75	2.02	1.71	1.63
Combo4v2.hold.15000.15	1.23	0.88	0.83	2.13
Combo4v2.hold.25000.1	1.77	1.14	0.77	1.50
Combo4v2.setup.15000.15	0.75	1.77	0.83	2.36
Combo4v2.setup.25000.1	1.03	1.02	0.79	2.15
Combo5v2.hold.20000.15	1.01	2.17	0.69	1.33
Combo5v2.hold.35000.1	1.63	1.19	0.71	1.15
Combo5v2.setup.20000.15	0.77	2.22	0.78	1.70
Combo5v2.setup.35000.1	0.90	1.70	0.73	1.31
Combo6v2.hold.35000.15	1.17	0.00	0.87	1.24
Combo6v2.hold.50000.1	1.77	0.00	0.74	0.97
Combo6v2.setup.35000.15	0.76	0.00	1.06	1.49
Combo6v2.setup.50000.1	1.06	0.00	0.81	1.04
Combo7v2.hold.35000.20	1.12	2.17	0.68	1.29
Combo7v2.hold.50000.1	1.76	0.00	0.65	1.04
Combo7v2.setup.35000.20	0.62	2.45	0.77	1.81
Combo7v2.setup.50000.1	0.65	2.32	0.66	1.20
Sum	21.34	27.20	19.57	28.74

Figure 7. Final scores of the top four timers in 2014 TAU CAD contest [3].

distributed computing [2]. The evaluation is taken on a computer cluster having over 500 compute nodes with each configured with 16 Intel E5-2670 2.60GHz cores and 128GB RAM. The network infrastructure is 384-port Mellanox MSX6518-NR FDR InfiniBand for high speed cluster interconnect [4]. Access to compute nodes for running a program is achieved via a script submission specifying the number of process cores to be used.

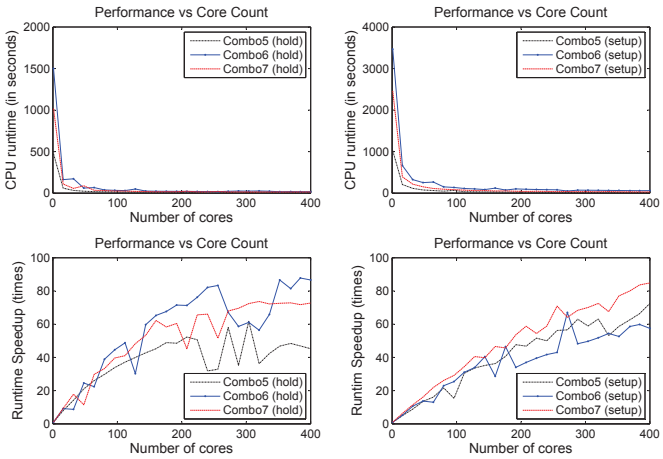

Figure 8. Runtime and speedup curves of hold tests and setup tests from benchmarks Combo5, Combo6, and Combo7.

We begin by demonstrating the runtime performance versus the the number of cores that is invoked for running our program. The quantity is varied from 1 core to 400 cores and the runtime is measured by a synchronized moment at which all process cores complete their jobs (i.e., reading the file, passing message, and handling all algorithmic procedures). The performance is interpreted in terms of the runtime and its relative speedup to a baseline which was run in single-core execution. Figure 8 shows the performance plot of this evaluation. It can be clearly seen that the runtime is reduced drastically as the number of cores increases. The largest difference is observed in hold tests of Combo6, where all timing tests are accomplished by 17.09 seconds using 384 cores. The speedup compared with single-core run reaches up to $\times 88$ (1500.55 seconds over 17.09 seconds). Similar trends can also be found in the other two testcases, where the largest speedup is $\times 72$ for setup tests of Combo5 using 400 cores and $\times 85$ for setup tests of Combo7 using 400 cores. In a single minute, hold tests and setup tests of all testcases are solvable using 80 cores and 272 cores.

V. Conclusion

In this paper we have presented an exact and extremely fast algorithm for handling the CPPR problem during static timing analysis. Unlike existing approaches which are predominated by exhaustive path search along with case-by-case speedup heuristics, our timer maps the CPPR problem to a graph-theoretic formulation and applies an efficient search routine using a highly compact and efficient data structure to obtain an exact solution. Our timer has several merits such as simplicity, coding ease, and most importantly the theoretically-proven completeness and optimality [13]. These advantages confer our timer a high degree of differential over existing methods. Comparatively, experimental results have demonstrated the superior performance of our timer in terms of accuracy and runtime over top-ranked timers from TAU 2014 CAD contest.

Future works shall focus on the development of even more efficient algorithm for path-based CPPR. Studies in fast CPPR algorithms are still eagerly in demand especially when we move to multi-core or many-core era [16]. As signoff timing still takes a significant portion of the entire design cycle, any developments that contribute to a substantial speedup will be beneficial to shorten the timing closure. Algorithms that are featured by massively-parallel accelerations in a large distributed system are in particular of our interests.

Acknowledgement

This work was partially supported by the National Science Foundation under Grant CCF-1320585. The authors acknowledge Y.-M. Yang, Y.-W. Chang, and I. H.-R. Jiang from team "iTimerC" and M. S. S. Kumar and N. Sireesh from team "LightSpeed" for sharing their binaries in TAU 2014 CAD contest. A special thank is given to Jin Hu and Debjit Sinha from IBM Corp., and Igor Keller from Cadence in organizing the 2014 TAU CAD contest.

References

[1] OpenMP: Parallel Programming API, http://openmp.org/wp/

[2] OpenMPI: Open Source High Performance Computing, http://www.open-mpi.org/

[3] TAU 2014 Contest: Pessimism Removal of Timing Analysis, http://sites.google.com/site/taucontest2014

[4] Illinois Campus Cluster, https://campuscluster.illinois.edu/

[5] H. Aljazzar and S. Leue, "K*: A Heuristic Search Algorithm for Finding the K Shortest Paths," *Artificial Intelligence*, vol. 175, no. 18, pp. 2129–2154, 2011.

[6] M. A. Bender and M. F. Colton, "The LCA problem revisited," *Proc. 4th Latin American Symposium on Theoretical Informatics*, vol. 1776, pp. 88–94. Springer, 2000.

[7] J. Bhasker and R. Chadha, "Static Timing Analysis for Nanometer Designs: A Practical Approach," *Springer*, 2009.

[8] Sarvesh Bhardwaj, Khalid Rahmat, and Kayhan Kucukcakar, "Clock-Reconvergence Pessimism Removal in Hierarchical Static Timing Analysis," US patent 8434040, 2013.

[9] S. Cristian, N. H. Rachid, and R. Khalid, "Efficient exhaustive path-based static timing analysis using a fast estimation technique," US patent 8079004, 2009

[10] D. Eppstein, "Finding the k shortest paths," *Proc. IEEE FOCS*, pp. 154–165, 1994.

[11] D. Hathaway, J. P. Alvarez, and K. P. Belkbale, "Network Timing Analysis Method which Eliminates Timing Variations between Signals Traversing a Common Circuit Path," US patent 5636372, 1997.

[12] J. Hu, D. Sinha, and I. Keller, "TAU 2014 Contest on Removing Common Path Pessimism during Timing Analysis," *Proc. ACM ISPD*, pp. 153–160, 2014.

[13] T.-W. Huang, P.-C. Wu, and Martin D. F. Wong, "UI-Timer: An Ultra-Fast Clock Network Pessimism Removal Algorithm," *Proc. IEEE/ACM ICCAD*, 2014.

[14] O. Levitsky, "Sign Off Quality Hierarchical Timing Constraints: Wishful Thinking or Reality?" *TAU workshop*, 2014.

[15] E. Q. V. Martins and M. M. B. Pascoal, "A New Implementation of Yen's Ranking Loopless Paths Algorithm," A quarterly Journal of Operation Research, vol. 1, no. 2, 2003.

[16] R. Molina, "EDA Vendors should Improve the Runtime Performance of Path-Based Timing Analysis," *Electronic Design*, 2013

[17] A. K. Ravi, "Common Clock Path Pessimism Analysis for Circuit Designs using Clock Tree Networks," US patent 7926019, 2011.

[18] J. Y. Yen, "Finding the k Shortest Loopless Paths in a Network," *Manage. Sci*, vol. 17 no. 11, pp. 712–716, 1971.

[19] J. Zejda and P. Frain, "General Framework for Removal of Clock Network Pessimism," *Proc. IEEE/ACM ICCAD*, pp. 632–639, 2002.