Tree Structures and Algorithms for Physical Design

Chung-Kuan Cheng[a], Ronald Graham[a], Ilgweon Kang[a],
Dongwon Park[b] and Xinyuan Wang[b]

Computer Science and Engineering[a], Electrical and Computer Engineering[b]
UC San Diego, La Jolla, California
{ckcheng,graham,igkang,dwp003,xiw193}@ucsd.edu

ABSTRACT

Tree structures and algorithms provide a fundamental and powerful data abstraction and methods for computer science and operations research. In particular, they enable significant advancement of IC physical design techniques and design optimization. For the last half century, Prof. T. C. Hu has made significant contributions to broad areas in computer science, including network flows, integer programming, shortest paths, binary trees, global routing, etc. In this article, we select and summarize three important and interesting tree-related topics (ancestor trees, column generation, and alphabetical trees) in the highlights of Prof. T. C. Hu's contributions to physical design.

ACM Reference Format:
Chung-Kuan Cheng, Ronald Graham, Ilgweon Kang, Dongwon Park and Xinyuan Wang. 2018. Tree Structures and Algorithms for Physical Design. In *Proceedings of 2018 International Symposium on Physical Design (ISPD'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/http://dx.doi.org/10.1145/3177540.3177564

1 INTRODUCTION

A *tree* is a fundamental data structure in computer science. From operations research to VLSI integrated circuit (IC) design and their optimization, and even for artificial intelligence (AI) technologies and warehouse computing systems, a tree structure provides a simple but powerful data abstraction in various applications. Starting with a *root* node on the top of all other nodes, a *tree* is composed of a set of linked nodes without any cyclic paths. In a tree structure, we define a *parent* as a node that directs to some other node (i.e., *child*), an *ancestor* as a node reachable by repeatedly going upward from child to parent, and a *leaf* as a node without any children.

The tree structure and its related algorithms are also widely used in IC physical layout design. In floorplanning, tree structures (e.g., twin binary trees [41, 42], O-trees [17], and B*-trees [5]) are popular methods to represent the given floorplan for the layout design. Ratio cut and replication cut provide breakthroughs for two-way partitioning [9, 32]. The column generation technique that reduces the linear program complexity can be applied to efficiently construct tree structures for global routing [4, 21]. Alphabetical trees can be used for synthesizing adders [33, 43] and interconnect design models [39]. Also clock tree synthesis (CTS) is a crucial procedure during physical design to build a tree having low skew, while delivering the same clock signal to every sequential gate [28].

For the past half a century, Prof. T. C. Hu has contributed significantly to the improvement of algorithms in operations research and computer science. His papers and journal articles have been incredibly important across broad topics, including network flows, integer programming, shortest paths, binary trees, etc [44]. Specifically, many of his research results on tree structures suggest better opportunities to observe and understand IC physical layout designs. One of his notable achievements is the paper with Gomory on multi-terminal flows [14]. The so-called Gomory-Hu tree of an undirected graph in combinatorial optimization is a weighted tree that represents the minimum source-sink cuts for all source-sink pairs in the graph [44]. They proved that we can use $n-1$ computations through the ancestor tree since the Gomory-Hu cut tree uses $n-1$ non-crossing minimum cuts to find all the ordinary maximum flow values between all nodes in a network [8, 9].

In this work, we present several tree structures and related algorithms in the highlights of Prof. Hu's contributions. The remaining sections are organized as follows. Section 2 introduces the Gomory-Hu cut tree and the ancestor tree. Section 3 describes the duality in linear programs and the column generation. Section 4 presents the alphabetical tree and Hu-Tucker algorithm. Section 5 concludes this article.

ISPD'18, March 25–28, 2018, Monterey, CA, USA

ACM ISBN ISBN 978-1-4503-5614-5/18/03…$15.00
https://doi.org/http://dx.doi.org/10.1145/3177540.3177564

2 ANCESTOR TREES

For ancestor trees, we try to represent the partitioning structure of a network. The cost of the cut can be arbitrary and it is known that the general problem of the finding the optimal cut is NP complete. The formulation is inspired by the Gomory-Hu cut tree which starts with a maximum flow minimum cut partitioning. The multi-terminal maximum flow problem is to find the maximum flow values for all pairs of nodes in a network. By the Max-Flow Min-Cut Theorem of Ford and Fulkerson [11], the maximum flow between a given pair of nodes is equal to the capacity of the minimum cut separating the two nodes.

We are given an undirected network with node set V, $|V| = n$, and edge set E. Each edge $e_{ij} \in E$ connects nodes i and j with a non-negative capacity b_{ij}. The maximum flow value between nodes i and j is denoted by f_{ij}. We utilize the description in Prof. Hu's book [22] to state the following theorems and lemmas.

THEOREM 1. *A necessary and sufficient condition for a set of non-negative numbers $f_{ij} = f_{ji}$ $(i, j = 1, \cdots, n)$ to be the maximum flow values of a network is*

$$f_{ik} \geq \min(f_{ij},\ f_{jk}) \quad \forall i,\ j,\ k. \tag{1}$$

Once inequality (1) is established, then by induction we have

$$f_{ip} \geq \min(f_{ij},\ f_{jk},\ f_{kl},\ \cdots,\ f_{op}), \tag{2}$$

where indices $i, j, \cdots, p$, represent an arbitrary sequence of nodes in the network. With inequality (1), the following lemma states that the maximum flow values of the network can be simplified.

LEMMA 1. *For any three nodes of the network, at least two of the maximum flow values between them must be equal.*

By induction following the inequality (2), we find that among the $n(n-1)/2$ maximum flow values $f_{ij} = f_{ji}$, there exist at most $n-1$ distinct values.

Note that we can generalize the above theorem and lemma by extending the cutting objective to arbitrary cost functions f as long as the cost is symmetric, i.e. $f_{ij} = f_{ji}$. We can prove the extended statement by enumeration, which is similar to the original proof.

2.1 The Gomory-Hu Cut Tree

Among all $\binom{n}{2}$ pairs of nodes, Gomory and Hu [14] showed that $n-1$ minimum cost cuts which do not cross each other are sufficient to separate all pairs of nodes. We call any two networks *flow-equivalent* if the maximum flow values between all pairs of nodes are the same.

THEOREM 2. *Any network is flow-equivalent to a cut tree.*

To construct the cut tree, we first shall describe a process called "*condensing nodes*". A set of nodes in original graph is replaced by a single node, and all the edges connecting any node not in the set are condensed to a single edge associated with the sum of capacities of original edges. Then we will have a simplified network which will reduce computations.

The *non-crossing minimum cuts* provide the condition as to whether any subset of nodes can be condensed while not affecting the maximum flow computation. In [22], Chapter 2.3.2 discusses related lemmas and their proofs in detail.

We adopt the algorithm of constructing the tree network from [14] and use a numerical example to demonstrate the technique. Consider the network with five nodes in Figure 1, the numbers associated with the edges are their corresponding capacities. We are going to find the maximum flow value between each pair of nodes in the network.

First let us arbitrarily choose a pair of nodes (2,4) and perform a maximum flow computation. A minimum cut $(1,2 \mid 3,4,5)$ separates nodes 2 and 4 with flow 8. The set of nodes on each side of the cut can be represented with a terminal node as shown in the first step in Figure 1, and the cut value is assigned to the link of the cut tree.

Next choose any two nodes from a terminal node of the current cut tree. We pick the terminal node $\{1,2\}$ and perform a maximum flow computation between nodes 1 and 2 on the condensed network (Figure 1.2)). The result gives a minimum cut $(1 \mid 2,3,4,5)$ with value 8. The terminal nodes $\{1,2\}$ are partitioned into two sets connected by a link with the cut value, which then replace the terminal node $\{1,2\}$. The computation is repeated until there is only one node of original network in each terminal node of the cut tree.

After $n-1$ iterations, a tree diagram is obtained, which is termed a *Gomory-Hu cut tree* of this 5-node graph. The $n-1$ non-crossing minimum cuts are shown in red dashed lines in the original network.

2.2 The Construction of Ancestor Tree

The construction of the Gomory-Hu cut tree relies on the fact that there always exists a set of $n-1$ non-crossing minimum cuts. With arbitrary cut functions, the cuts may cross each other. However, Cheng and Hu [8, 9] show that we still need only $n-1$ computations by constructing a binary tree called an *ancestor tree*. This tree has $n-1$ internal nodes and each represents a cut, as well as n leaves and each of these represents a node of the original network. For each pair of leaves, their lowest common ancestor is the minimum cut of the corresponding pair of nodes. We call the $n-1$ distinct cuts an *essential cut set*.

In the construction of the tree, we define some nodes as *seeds*. Each cut C_{ij} in the tree is minimum cost cut separating seeds i and j. The tree structure that maintains the following three properties throughout the iterations is *admissible* [9].

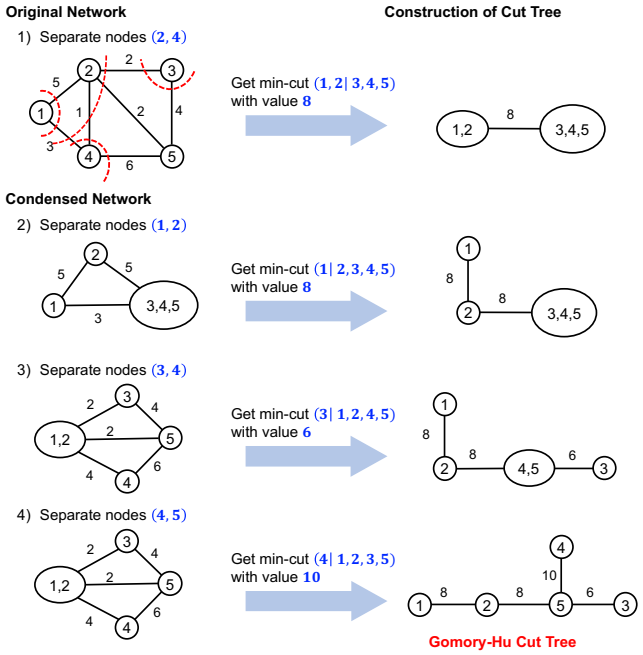

Figure 1: Construction of a Gomory-Hu cut tree.

- The cut value of an internal node is always no smaller than the value of its father.
- Each leaf L_i contains exactly one seed i.
- For any pair of leaves L_i and L_j with seeds i and j, let the internal node C_{pq} be their lowest ancestor.

Then C_{pq} with its two branches defines a minimum cut separating nodes i and j.

The algorithm to construct an ancestor tree gradually builds trees T_0, T_1, $\cdots$, T_{n-1} by adding one more cut node and leaf [9] at a time. A numerical example is shown in Figure 2. The cost function is defined as $F_{ij} = \min \frac{C(X, \bar{X})}{|X| \cdot |\bar{X}|}$ with node $i \in X$ and node $j \in \bar{X}$. Seeds are postfixed with '*'.

Figure 2 illustrates an example of the process. We choose a pair of nodes $(2,4)$ and separate them with minimum cost cut $\{1, 2^* | 3, 4^*, 5\}$. The cut cost is $F_{24} = 8/(2 \times 3) = 4/3$, then we have a cut node C_{24} with two branches and each is labeled with a partitioned subset of nodes. As shown in Figure 2:step 1, the tree has two leaves. One contains the subset $\{1, 2^*\}$ with seed 2^* and one contains $\{3, 4^*, 5\}$ with seed 4^*.

In the next iteration, we pick nodes $(1,2)$ from the left leaf and compute the cut. The minimum cut $\{1^*, 4 \mid 2^*, 3, 5\}$ has value no smaller than its father $F_{12} = 10/(2) = 5/3 > F_{24}$, so we append the cut node C_{12} to the position of the old leaf and append new leaves to C_{12}. The leaves contain nodes by intersecting the labels along the branches. For example, the left leaf of C_{12} contains nodes $\{1,2\} \cap \{1,4\} = \{1^*\}$.

There exists another case when the cost is smaller than its father. In Figure 2:step 4, the cut cost $F_{45} = 3/2 < F_{34}$.

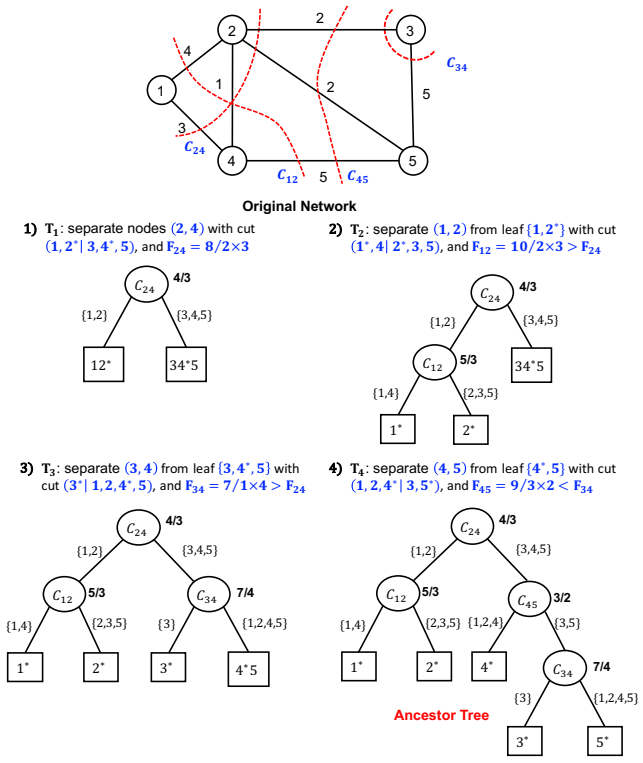

Figure 2: An example of constructing an ancestor tree.

We find the highest ancestor of C_{45} with a larger cut cost which is C_{34}, so we place C_{45} at this position. Node C_{34} with its subtree is appended to C_{45} and the subsets of nodes are updated. Repeat these steps until there exists only one seeded node in each leaf. The cuts are shown in red dashed lines in the original network among which C_{12} and C_{24} cross each other.

Given an admissible tree T_k, we can prove the tree T_{k+1} is still admissible after an iteration. The correctness of the algorithm is proved by Theorem 3 [9].

THEOREM 3. *The cut tree algorithm derives an essential cut set with $n-1$ minimum cut calls.*

2.3 Applications of the Ancestor Tree

The ancestor tree provides a natural hierarchical representation of a complicated network. The technique has been applied to solving complex multi-commodity network optimization problems [12] as well as network partitioning problems [7, 30]. This partitioning can be further applied to solve VLSI design problems for logic synthesis and physical layout [6]. An extension to directed graphs would be an interesting research problem.

3 COLUMN GENERATION

Column generation is a powerful technique utilizing the duality of the problem formulation. For example, in global

routing, the original problem has a huge number of possible variables where each variable represents a tree structure. The search for an optimal routing tree is by itself an NP-complete problem in general. The competition among the trees for a limited routing resources complicates the issue. Column generation utilizes the shadow price of the dual formulation to provide a systematic guide in searching for the tree [4, 19].

3.1 Duality in Linear Programming

Formulations (3) and (4) represent the *primal* and *dual* of a linear programming problem, where $\mathbf{c}$ and $\mathbf{y}$ are row vectors, $\mathbf{x}$ and $\mathbf{b}$ are column vectors, and $\mathbf{A}$ is an $m \times n$ matrix. The primal (resp. dual) has n (resp. m) variables and m (resp. n) constraints. The variables $\mathbf{y}$ in the dual program are often denoted by π, and are called the *shadow prices*[1].

$$\begin{aligned} &\text{Maximize} && z = \mathbf{cx} && (3) \\ &\text{subject to} && \mathbf{Ax} \le \mathbf{b} \\ & && \mathbf{x} \ge \mathbf{0} \end{aligned} \qquad \begin{aligned} &\text{Minimize} && w = \mathbf{yb} && (4) \\ &\text{subject to} && \mathbf{yA} \ge \mathbf{c} \\ & && \mathbf{y} \ge \mathbf{0} \end{aligned}$$

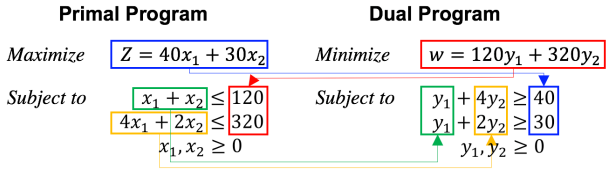

Figure 3: A primal program and its dual program [46].

Figure 3 illustrates the relation between the primal and dual programs. For the primal program, the maximum $z = 4000$ occurs when $x_1 = 40$ and $x_2 = 80$. For the dual, the minimum $w = 4000$ occurs when $y_1 = 20$ and $y_2 = 5$. For the objective function in the linear program, we define the *reduced costs* (i.e., opportunity costs) as $\mathbf{c} - \mathbf{yA}$, implying that the current basis can be improved when the reduced cost is negative. In other words, if all the reduced costs are non-negative, then the current solution is optimal. For the primal program in the figure, the reduced costs are $[0, 0]$ at the optimal solution as $[40, 30] - [20, 5]\begin{bmatrix} 1 & 1 \\ 4 & 2 \end{bmatrix} = [0, 0]$.

3.2 The Technique of Column Generation

For a routing problem, the primal formulation can have a huge number of variables which can far exceed the computational capability of a computer. The *column generation* technique offers an efficient technique to avoid this computational blowup. This technique allows us to reduce our attention to only a subset of the variables. Through primal-dual iterations, we swap improved choices through shadow prices.

[1]At the optimal solution of the linear program, the shadow price equals to the value of the *Lagrangian multiplier*.

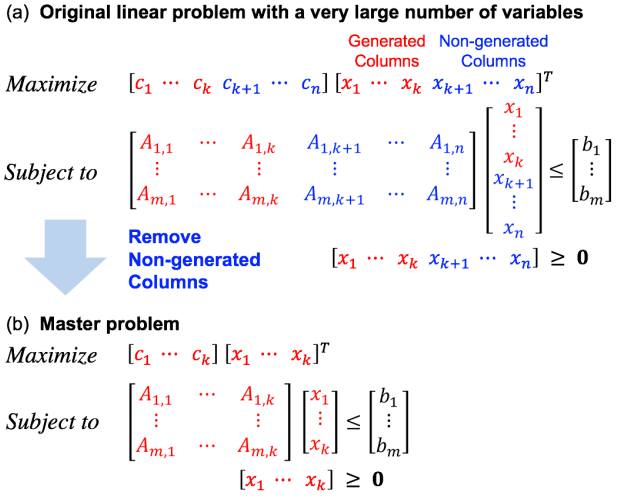

Figure 4: An example of producing the master problem for the column generation procedure. The original figure can be found in [45].

Figure 4 demonstrates the variable reduction process. We partition columns into two groups (Figure 4(a)). From an initial solution of the linear program (3), we have (i) generated columns and (ii) non-generated columns. We only deal with a small subset of variables (i.e., generated columns) to solve the given linear program. By removing non-generated columns, we obtain the *master problem* composed of generated columns so that we can reduce the problem size (shown in Figure 4(b)).

Figure 5 shows an overview of the column generation technique and the relation between the master problem which gives the dual variables (i.e., the shadow prices) and the *subproblem* which identifies a new column (if it exists). Whenever the subproblem proposes a new variable to add, then the master problem adds a column to the matrix $\mathbf{A}$. This is

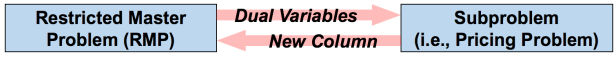

Figure 5: An overview of column generation.

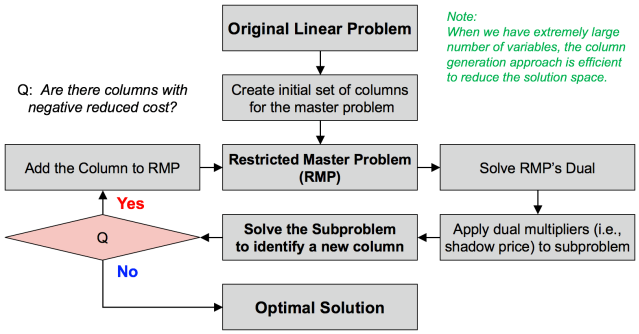

Figure 6: A flow chart for column generation.

why this process is called column generation [45]. By alternating the two problems, the column generation efficiently approaches the optimal solution.

As mentioned above, if the minimum of all reduced costs is non-negative, then all the reduced costs are non-negative and the current solution is optimal. If the least reduced cost is negative (meaning that the current solution can be further improved), then we add the new column into the current basis. The iteration procedure of the column generation terminates when the least reduced cost is non-negative.

Figure 6 summarizes the overall flow of the column generation technique. From the original linear program, we first find a feasible solution and produce the restricted master problem. We then solve the dual of the master problem to obtain the shadow prices π as the sensitivity analysis effectively identifies the most critical variable. Through the dual feasible solution of the linear program, we try to find the column $\mathbf{a_j}$ which leads to the maximum $\pi\mathbf{a_j}$. We can find the minimum reduced cost since the reduced cost is equal to $c_j - \pi a_j$. If there are no columns with the negative reduced cost, we end up with the optimal solution [20].

3.3 Applications of Column Generation

The column generation technique enables a large reduction of computation efforts when the number of variables is very large [15, 16]. For instance, we typically assign a variable for every possible route in vehicle routing problem. Thus, the number of variables is of the order of the factorial of the number of places to visit [45]. Some applications of the column generation technique are vehicle routing [10, 34], integrated circuit routing [4, 21, 26, 27], scheduling [25, 37], transportation [35], network design [2, 3], etc.

For physical design, our choice of the column generation approach for global routing [4, 18] is inspired by the discussion with Prof. Hu. The approach allows us to derive a theoretical bound from the optimal routing solution. Huang et al. [24] improved the circuit performance of the routing tree with timing cost in the formulation. Albrecht [1] incorporated the wirelength as part of the constraints in the primal problem. In [40], Wu et al. used the column generation to enumerate candidate routing trees and complete the tree selection using integer programming at local regions.

4 ALPHABETICAL TREES

Alphabetical trees fit physical design very well because the tree construction respects the sequence of the nodes. In the formulation, the order of the nodes is given. We construct a tree to connect these nodes so that a depth-first traversal maintains the same order. Figure 7 shows an example of an alphabetical tree. In this tree structure, we can obtain the letters in the alphabetic order by scanning the leaves from left to right. For a binary tree, where left edges are assigned

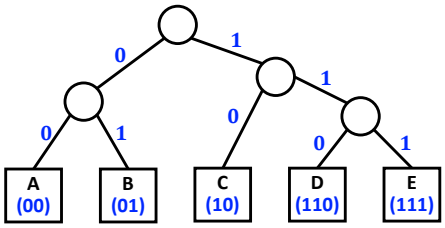

Figure 7: An alphabetical tree

with 0 and right edges are assigned with 1, each leaf L_i can be associated to a binary number $n(L_i)$ along the path from the root node. For an alphabetical tree, and the following condition is always satisfied:

$$n(L_i) < n(L_{i+1}),$$

where i is the index of a leaf specified from left to right. For physical routing, a mismatch in the order indicates a crossing of the wires and a waste of routing resources.

4.1 The Hu-Tucker Algorithm

Several algorithms have been proposed to find optimum alphabetical trees. Gilbert and Moore constructed optimum alphabetical trees using dynamic programming techniques which has $O(n^3)$ in runtime and $O(n^2)$ in memory space [13]. Knuth [29] proposed a method that reduces the computation time to $O(n^2)$ but still has $O(n^2)$ in memory space. Hu and Tucker [23] further reduces the complexity to $O(n \log n)$ in runtime and $O(n)$ in memory space.

The Hu-Tucker algorithm has three phases which are the *combination, level assignment* and *reconstruction* phases as shown in Figure 8. In the combination phase, nodes are merged to produce the smallest weight when combined nodes are adjacent in the sense that there is no leaf between them. Note that between two adjacent leaves, we could have internal nodes, e.g., internal node labeled 10 between nodes 7 and 5 in Figure 8(a,b). The corresponding tree of the combinational phase is denoted by T' (Figure 8(a)), which is an optimum binary tree but not an alphabetical tree. To translate the optimum binary tree T' into an optimum alphabetical tree T'_N, the level assignment phase is required to identify the level of each leaf using the topology of T'. The level number is the path-length from the root node as shown in Figure 8(b). According to the result of the level assignment phase, an optimum alphabetical tree T'_N is composed by a level-by-level construction method from the bottom to the top in the reconstruction phase (Figure 8(c)).

4.2 Applications of Alphabetical Trees

Alphabetical trees have been applied in several applications such as adder synthesis, interconnect design problem and layout-driven logic synthesis. In generic parallel adders based on prefix computation, the computation structure of adders

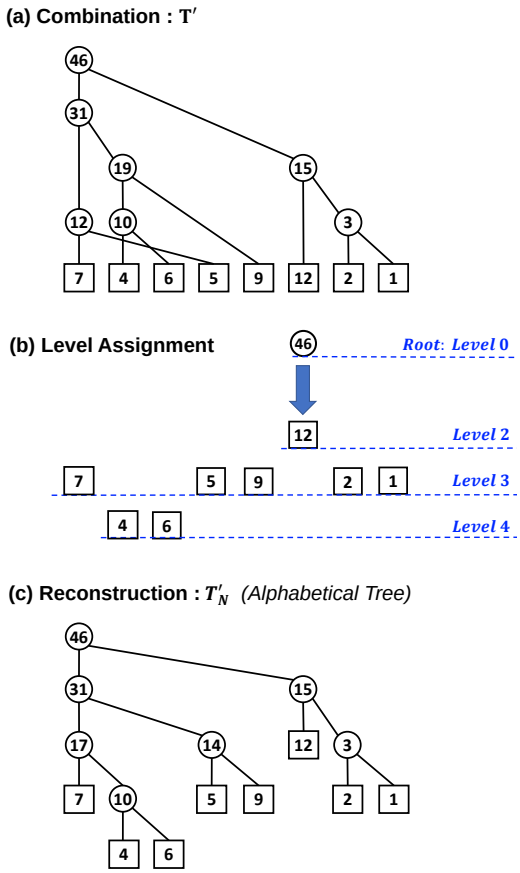

Figure 8: The Hu-Tucker algorithm [22].

can be represented and simplified by an alphabetical tree [33, 43]. In [39], Vittal and Marek-Sadowska used the alphabetical tree with an analytical model to minimize the interconnect delay. In [36, 38], Vaishnav and Pedram adopted the alphabetical tree to optimize the fan-out in the layout-driven logic synthesis and technology decomposition.

5 CONCLUSION

In this article, we selected three tree-related topics, i.e., ancestor trees, column generation, and alphabetical trees. We then briefly described them in each section highlighting Prof. Hu's contributions. These techniques will certainly contribute to future innovation in these fundamental areas.

6 ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation (Grant CCF-1564302).

REFERENCES

[1] C. Albrecht, "Provably Good Global Routing by A New Approximation Algorithm for Multicommodity Flow", *Proc. ISPD*, 2000, pp. 19-25.
[2] F. Alvelos and J. M. V. de Carvalho, "An Extended Model and a Column Generation Algorithm for the Planar Multicommodity Flow Problem", *Networks* 50(1) (2007), pp. 3-16.
[3] W. Ben-Ameur and J. Neto, "Acceleration of Cutting-Plane and Column Generation Algorithms: Applications to Network Design", *Networks* 49(1) (2007), pp. 3-17.
[4] R. C. Carden, J. Li and C. K. Cheng, "A Global Router with a Theoretical Bound on the Optimal Solution", *IEEE Trans. on CAD* 15(2) (1996), pp. 208-216.
[5] Y. C. Chang, Y.-W. Chang, G. M. Wu and S. W. Wu, "B*-Trees: A New Representation for Non-Slicing Floorplans", *Proc. DAC*, 2000, pp. 458-463.
[6] S.-J. Chen and C. K. Cheng, "Tutorial on VLSI Partitioning", *VLSI Design* 11(3) (2000), pp. 175-218.
[7] C. K. Cheng, "The Optimal Partitioning of Networks", *Networks* 22(3) (1992), pp. 297-315.
[8] C. K. Cheng and T. C. Hu, "Ancestor Tree for Arbitrary Multi-Terminal Cut Functions", *Annals of Operations Research* 33(3) (1991), pp. 199-213.
[9] C. K. Cheng and T. C. Hu, "Maximum Concurrent Flows and Minimum Ratio Cuts", *Algorithmica* 8(1) (1992), pp. 233-249.
[10] C. Contardo, J.-F. Cordeau and B. Gendron, "An Exact Algorithm Based on Cut-and-Column Generation for the Capacitated Location-Routing Problem", *INFORMS Journal on Computing* 26(1) (2014), pp. 88-102.
[11] L. R. Ford and D. R. Fulkerson, "Maximal Flow through a Network", *Canadian Journal of Mathematics* 8(3) (1956), pp. 399-404.
[12] V. Gabrel, A. Knippel and M. Minoux, "Exact Solution of Multicommodity Network Optimization Problems with General Step Cost Functions", *Operations Research Letters* 25(1) (1999), pp. 15-23.
[13] E. N. Gilbert and E. F. Moore, "Variable Length Binary Encodings", *Bell System Technical Journal* 38 (1959), pp. 933-968.
[14] R. E. Gomory and T. C. Hu, "Multi-Terminal Network Flows", *Journal of SIAM* 9(4) (1961), pp. 551-570.
[15] J. Gondzio, P. González-Brevis and P. Munari, "New Developments in the Primal-Dual Column Generation Technique", *European Journal of Operational Research* 224(1) (2013), pp. 41-51.
[16] J. Gondzio, P. González-Brevis and P. Munari, "Large-Scale Optimization with the Primal-Dual Column Generation Method", *Mathematical Programming Computation* 8(1) (2016), pp. 47-82.
[17] P.-N. Guo, T. Takahashi, C. K. Cheng and T. Yoshimura, "Floorplanning Using a Tree Representation", *IEEE Trans. on CAD* 20(2) (2001), pp. 281-289.
[18] J. Hu and S. S. Sapatnekar, "A Survey on Multi-Net Global Routing for Integrated Circuits", *Integration, the VLSI Journal* 31(1) (2001), pp. 1-49.
[19] T. C. Hu, "Multi-Commodity Network Flows", *Operations Research* 11(3) (1963), pp. 344-360.
[20] T. C. Hu and A. B. Kahng, *Linear and Integer Programming Made Easy*, Springer, 2016.
[21] T. C. Hu and M. T. Shing, "A Decomposition Algorithm for Circuit Routing", *Mathematical Programming Study* 24 (1985), pp. 87-103.
[22] T. C. Hu and M. T. Shing, *Combinatorial Algorithms*, 2nd Ed., Dover Publications, 2002.
[23] T. C. Hu and A. C. Tucker, "Optimal Computer Search Trees and Variable-Length Alphabetical Codes", *Journal of SIAM* 21(4) 1971, pp.514-532.
[24] J. Huang, X. L. Hong, C. K. Cheng and E. S. Kuh, "An Efficient Timing-Driven Global Routing Algorithm", *Proc. DAC*, 1993, pp. 596-600.
[25] J. Janacek, M. Kohani, M. Koniorczyk and P. Marton, "Optimization of Periodic Crew Schedules with Application of Column Generation Method", *Transportation Research Part C: Emerging Technologies* 83 (2017), pp. 165-178.
[26] G.-W. Jeong, K. Lee, S. Park and K. Park, "A Branch-and-Price Algorithm for the Steiner Tree Packing Problem", *Computers & Operations Research* 29(3) (2002), pp. 221-241.
[27] D. G. Jørgensen and M. Meyling, "A Branch-and-Price Algorithm for Switch-Box Routing", *Networks* 40(1) (2002), pp. 13-26.
[28] A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Springer, 2011.
[29] D. E. Knuth, "Optimum Binary Search Trees", *Acta Informatica 1* (1971), pp. 14-25.
[30] M. E. Kuo and C. K. Cheng, "A Network Flow Approach for Hierarchical Tree Partitioning", *Proc. DAC*, 1997, pp. 512-517.
[31] G. Laporte, "The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms", *European Journal of Operational Research* 59(3) (1992), pp. 345-358.
[32] L.-T. Liu, M.-T. Kuo, C. K. Cheng and T. C. Hu, "A Replication Cut for Two-Way Partitioning", *IEEE Trans. on CAD* 14(5) (1995), pp. 623-630.
[33] J. Liu, S. Zhou, H. Zhu and C. K. Cheng, "An Algorithmic Approach for Generic Parallel Adders", *Proc. ICCAD*, 2003, pp. 734-740.
[34] M. E. Lübbecke and J. Desrosiers, "Selected Topics in Column Generation", *Operations Research* 53(6) (2005), pp. 1007-1023.
[35] T. Nishi and T. Izuno, "Column Generation Heuristics for Ship Routing and Scheduling Problems in Crude Oil Transportation with Split Deliveries", *Computers & Chemical Engineering* 60 (2014), pp. 329-338.
[36] M. Pedram and H. Vaishnav, "Technology Decomposition Using Optimal Alphabetic Trees", *Proc. ECDA*, 1993, pp. 573-577.
[37] D. Potthoff, D. Huisman and G. Desaulniers, "Column Generation with Dynamic Duty Selection for Railway Crew Rescheduling", *Transportation Science* 44 (2010), pp. 493-505.
[38] H. Vaishnav and M. Pedram, "Alphabetic Trees - Theory and Applications in Layout-Driven Logic Synthesis", *IEEE Trans. on CAD* 42(2) (2002), pp. 219-223.
[39] A. Vittal and M. Marek-Sadowska, "Minimal Delay Interconnect Design Using Alphabetic Trees", *Proc. DAC*, 1994, pp. 392-396.
[40] T. H. Wu, A. Davoodi and J. T. Linderoth, "GRIP: Global Routing via Integer Programming", *IEEE Trans. on CAD* 30(1) (2011), pp. 72-84.
[41] B. Yao, H. Chen, C. K. Cheng and R. Graham, "Floorplan Representations: Complexity and Connections", *ACM Trans. on DAES* 8(1) (2003), pp. 55-80.
[42] E. F. Y. Young, C. C. N. Chu and Z. C. Shen, "Twin Binary Sequences: A Nonredundant Representation for General Nonslicing Floorplan", *IEEE Trans. on CAD* 22(4) (2003), pp. 457-469.
[43] Y. Zhu, J. Liu, H. Zhu and C. K. Cheng, "Timing-Power Optimization for Mixed-Radix Ling Adders by Integer Linear Programming", *Proc. ASP-DAC*, 2008, pp. 131-137.
[44] Brief Biography of Dr. Te Chiang Hu, *The Institute for Operations Research and the Management Sciences*, http://www.informs.org/Explore/History-of-O.R.-Excellence/Biographical-Profiles/Hu-Te-Chiang/.
[45] Column Generation and Dantzig-Wolfe Decomposition, *Group for Research in Decision Analysis (GERAD)*, http://www.science4all.org/article/column-generation/.
[46] Duality Theory in LP, http://www.slideshare.net/jyothimonc/duality-in-linear-programming/.