Accelerating FPGA Design Space Exploration Using Circuit Similarity-Based Placement

Xiaoyu Shi [#1], Dahua Zeng [#2], Yu Hu [*3], Guohui Lin [#4], Osmar R. Zaiane [#5]

[#] *Department of Computing Science, University of Alberta, Canada*
{[1] xshi, [2] dahua, [4] guohui, [5] zaiane}@cs.ualberta.ca

[*] *Department of Electrical and Computer Engineering, University of Alberta, Canada*
[3] bryanhu@ece.ualberta.ca

***Abstract*—This paper describes a novel and fast placement algorithm for FPGA design space (*e.g.*, area, power or reliability) exploration. The proposed algorithm generates the placement based on the topological similarity between two configurations (netlists) in the design space. Thus, it utilizes the sharing of reusable information during the design space exploration and avoids the time-consuming placement computation like VPR. Tested on logic-level and algorithm-level design space exploration cases, our similarity-based placement accurately depicts the "shape" of a design space and pinpoints the designs which are of most interest to IC designers. Moreover, a turbo version of circuit similarity-based placement performs an average of 30x (up to 100x) faster than VPR's while still achieving comparable placement results.**

I. Introduction

An FPGA design offers a variety of customizations by varying design parameters. Those parameters include decisions at the algorithm level (*e.g.*, SIMD or pipeline) or at the architecture level (*e.g.*, cache and bus structures); options at high-level synthesis (*e.g.*, scheduling and resource binding tradeoff); combinations of various logic synthesis and optimization (*e.g.*, ABC toolset [1]). Efficiency of a design space exploration tool is of paramount importance for designers to quickly identify a small set of favorable design parameter combinations (*i.e.*, configurations) for a multi-objective design. However, they still heavily rely on the general CAD tools (*e.g.*, Altera Quartus or Xilinx ISE) to generate every single configuration among thousands of them, and suffer from long runtime.

Previous work on accelerating design space exploration can be divided into two categories: methods that minimize the number of configurations to be evaluated [2], and methods that generate design evaluation by modeling [3] [4] [5]. The runtime for generating one configuration is crucial for the efficiency of both methodologies. In this paper, we accelerate placement, one of the most time-consuming phases in FPGA synthesis, to increase the efficiency of generating each configuration in the design space.

A unique property of design space exploration problem is the existence of *similarities* between netlists of different configurations. Such similarities include both *local similarity* and *global similarity*. The local similarity is due to the use of common primitives (*e.g.*, DSP modules or macros) in different configurations. The global similarity exists because the characteristics of the application are shared by all configurations that implement it. We illustrate this property using an algorithm-level example with two implementation algorithms (*i.e.*, RAG-n [6] and Hcub [7]) of a constant multiplier block. The algorithm-level schematics of these two implementations (generated by CMU SPIRAL [8]) are shown in Figure 1. Although there is a significant difference between the structures of these two configurations

This work was partially sponsored by the Natural Sciences and Engineering Research Council of Canada.

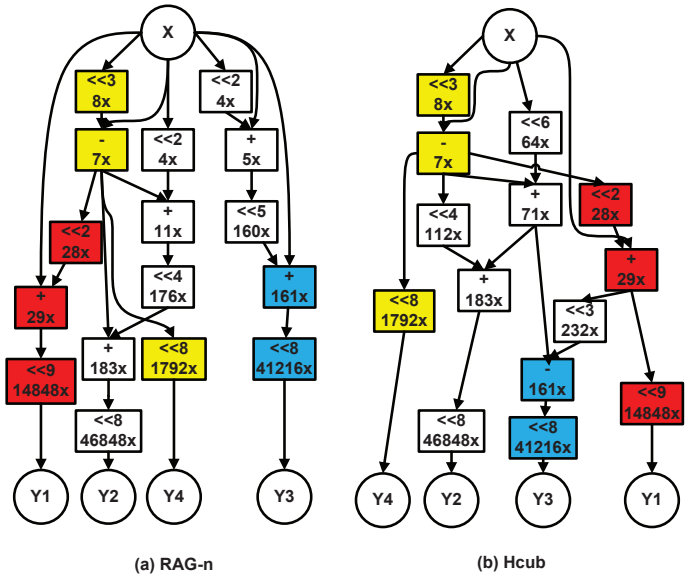


Fig. 1. Constant multiplier blocks generated by CMU SPIRAL (integer constants: 58, 183, 161, 7; bit width is 8)

at the first glance, they both use adders, subtractors and shifters as building blocks (primitives), which lead to a local similarity. When these algorithm level designs are mapped to FPGAs, such local similarity results in similarities of local clusters that contain look-up tables (LUTs) or DSPs. In addition, both configurations implement the same functionality, which results in global similarity. Specifically, the I/O (X and Ys) of both configurations are identical; there are identical internal structures (*e.g.*, subgraph $28x \rightarrow 29x \rightarrow 14848x$ is shared by both implementations) as highlighted in Figure 1 using different colors; both configurations are sparse directed acyclic graph (DAG) structures, and they are topologically similar.

Our approach takes advantage of this *similarity* property to accelerate the placements for all configurations in the design space. The kernel part is the *circuit similarity* algorithm, which quickly detects the similarity between configurations based on topological structures. Our proposed CAD flow, shown in Figure 2, starts with the computation of a reference placement for a particular configuration using an existing FPGA placer (*e.g.*, VPR [9]). After that, the placements for the rest of configurations can be generated very efficiently using the circuit similarity algorithm.

To verify the effectiveness of the circuit similarity algorithm, we have performed experiments on both the logic-level and algorithm-level design space cases. Experimental results show that our circuit

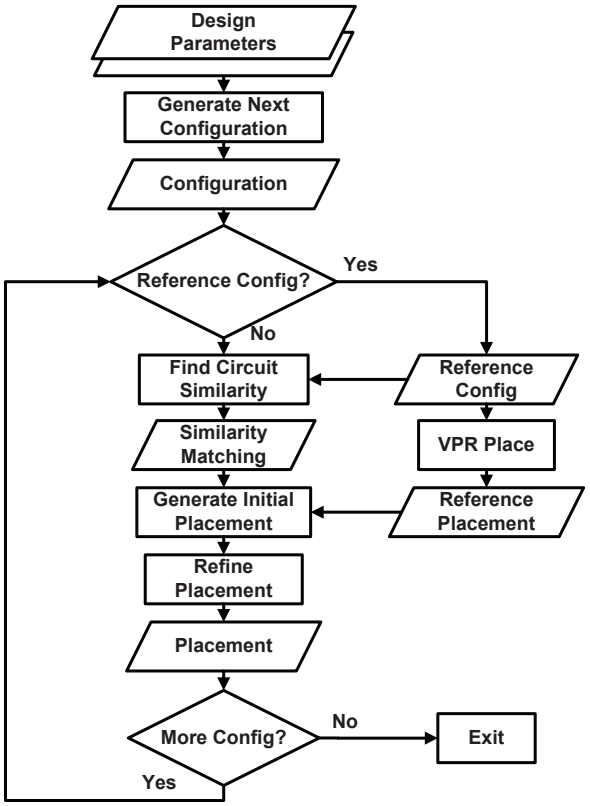


Fig. 2. CAD flow for the design space exploration using circuit similarity-based placement

similarity-based placement captures the characteristics of the design space with accurately estimated wire length and critical delay, and pinpoints the best designs. Moreover, our approach achieves averaged 30x speedup compared to VPR's placement.

The remainder of this paper is organized as follows: Section II describes the circuit similarity algorithm in detail. Section III experimentally demonstrates logic-level and algorithm-level design space exploration. The paper is concluded in Section IV.

II. Circuit similarity

A. Circuit Similarity Detection

Time complexity and global topological information are two major concerns in the existing graph similarity algorithms. Coupled node-edge algorithm computes graph similarity based on the iterative definition [10]. Our circuit similarity algorithm employs iterative method used in molecular graphs [11], which has less computation and contains global topological information.

The iterative similarity algorithm is summarized in Algorithm 1. In each iteration (t), the algorithm computes the *similarity score* $X_{i,j}^{(t)}$ ($0 \leq X_{i,j}^{(t)} \leq 1$) between each node pair (v_i, v_j') in two graphs, where $v_i \in G$ and $v_j' \in G'$. The higher the similarity score of a node pair is, the more likely these two nodes are matched together. This score is updated based on the values of their adjacent node pairs ($n(v)$) obtained in the previous iteration and the predefined inter-similarity between two nodes/edges (k_v/k_e). π is an injective map from $n(v_i)$ to $n(v_j')$, if $|n(v_i)| < |n(v_j')|$, and vice-versa. The predefined similarity is used to capture non-topological connections between two graphs. The algorithm terminates when the difference between the total similarity scores in two consecutive iterations is smaller than ϵ, or the number of iterations reaches an upper bound M.

Algorithm 1 Similarity of G and G'

Initialize $X_{i,j}^{(0)}$
while $|\sum X^{(t)} - \sum X^{(t-1)}| > \epsilon$ and $t < M$ **do**
  if $|n(v_i)| < |n(v_j')|$ **then**

$$X_{i,j}^{(t)} = (1-\alpha)k_v(v_i, v_j^{'}) + \alpha \max_{\pi} \frac{1}{|n(v_j')|} \sum_{v \in n(v_i)} X_{v,\pi(v)}^{(t-1)} k_e((v_i, v), (v_j^{'}, \pi(v)))$$

  else

$$X_{i,j}^{(t)} = (1-\alpha)k_v(v_i, v_j^{'}) + \alpha \max_{\pi} \frac{1}{|n(v_i)|} \sum_{v^{'} \in n(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)} k_e((v_i, \pi(v^{'})), (v_j^{'}, v^{'}))$$

Algorithm 1 is designed for undirected molecular graphs where the computational complexity is too expensive to handle real circuits. We first adapt Algorithm 1 to consider a directed circuit graph and then present two techniques to significantly improve both time and space efficiency.

One unique constraint for circuit similarity detection is that the matching of the corresponding PIs and POs of the two circuits must be guaranteed. Therefore, the similarity score of corresponding PI/PO nodes is set to be constant 1 and is not updated during the iteration. For those internal nodes without predefined similarity, we replace k_v with $X_{i,j}^{(t)}$, and k_e with 1. Moreover, we can perform the update for edges that leave the nodes and edges that enter the nodes, separately. More specifically, given the two graphs, we initialize the similarity scores of all pairs of nodes to be 1. In each iteration, for $|in(v_i)| < |in(v_j^{'})|$ and $|out(v_i)| < |out(v_j^{'})|$, the update of similarity score $X_{i,j}^{(t)}$ is modified as follows, where $in(v)$ is the set of all adjacent nodes entering v and $out(v)$ is the set of all adjacent nodes leaving v.

$$X_{i,j}^{(t)} = (1-\alpha)X_{i,j}^{(t-1)} + \alpha \frac{1}{|out(v_i)| + |in(v_i)|} [\max_{\pi}(\sum_{v^{'} \in out(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)}) + \max_{\pi}(\sum_{v^{'} \in in(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)})]$$

In our experiment, we find $\alpha = 0.75$ gives the best matching quality. After obtaining a similarity matrix that describes a complete bipartite graph, where the weight associated with each edge denotes the similarity score of two nodes, we can then compute a maximum matching to obtain a node matching between the two graphs using the min-cost network flow [12].

B. Performance Enhancement

In order to save runtime and storage, we present two pruning techniques in DACs to significantly reduce the number of updated node pairs.

Support Constraint. Two internal nodes are less likely to be matched if they share few predefined matchings in their supports. A *support* of a node is the set of nodes with predefined matchings in the transitive fanin or fanout cone of this node. Formally, for two nodes $v \in G$ and $v' \in G'$, the support constraint requires

$$\min(\frac{X_{SP(v),SP(v')}}{|SP(v)|}, \frac{X_{SP(v),SP(v')}}{|SP(v')|}) \geq \beta$$

where $X_{SP(v),SP(v')}$ denotes the *support similarity* of v and v', which is the sum of similarity scores of all $v \rightarrow v'$ node pairs in their supports, $SP(v)$ is the support node set of v and $\beta \in (0, 1]$ is a constant. If the support constraint of the two nodes is not satisfied, we do not update their similarity score in the iteration.

Level Constraint. Given a DAG, a topological sort and reverse topological sort can label each internal node v with two values, *i.e.*, $level(v)$ and $rlevel(v)$, where $level(v)$ ($rlevel(v)$) denotes the length of the longest path from PIs (node v) to node v (POs). Two nodes with significantly different ($level$, $rlevel$) values are less likely to be matched. Formally, for two nodes $v \in G$ and $v' \in G'$, the level constraint requires

$$|level(v) - level(v')| \leq B_l, |rlevel(v) - rlevel(v')| \leq B_r$$

where B_l and B_r are two nonnegative constant integers.

We have tested the above two pruning techniques with different settings. In summary, compared to the number of nodes without pruning, our pruning techniques reduce the number of nodes by 3 to 4 orders with comparable placement quality. Interested readers can refer to technical report [13] for more details.

C. Circuit Similarity-based Placement

As shown in Figure 2, circuit similarity is used to speed up placement which allows faster design space exploration. More specifically, given a network G where each node denotes a LUT and each edge denotes an interconnection between LUTs, the placement of network G can be obtained by performing a highly-optimized placement (*e.g.*, VPR). For another network, G', which is generated by other design parameters, its placement is generated by first computing the similarity between networks G and G', and finding the correspondence of nodes in these two networks. Based on such node correspondence, the initial placement of network G' can be determined using the placement of network G, *e.g.*, if node V' in network G' corresponds to node V in network G, V' is assigned the same coordinates as node V. Further refinement (*e.g.*, low-temperature simulated annealing) is applied to the initial placement of G' to gain better results.

III. Design Space Exploration

A. Logic-Level Design Space Exploration

Experimental CAD flow and settings. The objective of this design space exploration is to identify the influence of logic-level optimization to a post-layout design. Starting from 19 ABC logic synthesis scripts (abc.rc in ABC [1]), we have the resulting synthesized netlists stored in BLIF file format. Next, a technology mapping ("if -k 4") is performed on the netlists to map them into a 4-LUT-based network. Afterwards, the technology mapped netlists are packed into CLBs using T-VPack [9] with "no_cluster" option. After this point, we compare two CAD flows: (a) circuit similarity-based flow and (b) VPR *from-scratch* flow. Flow (a) first selects the largest configuration (*i.e.*, the one with largest number of CLBs) as the reference. Then the reference configuration is placed using VPR and produces a reference placement (".p" file). The reference configuration and its placement are then used to guide the initial placement of the new configuration by finding the similarity between the new configuration and the reference configuration. A low-temperature annealing process using VPR (initial temperature is set to 0.1) is performed to further refine the placement results. Flow (b) simply uses VPR to re-place every single configuration from scratch.

Based on different pruning settings, we develop two versions of circuit similarity. A high-quality version, CS, uses $\beta = 0.5, B_l = B_r = 1$ and $inner\_num = 1$ (Controls the number of moves at each temperature in VPR). A turbo version, CS-t, uses $\beta = 1, B_l = B_r = 0$ and $inner\_num = 0.1$. Both CS and CS-t are evaluated in our experiments.

Our proposed circuit similarity algorithm is implemented in C and evaluated on the 20 applications from the MCNC benchmark. We collect the results on a Linux server with an 8-core 2.66GHz CPU and 32GB memory averaged over five runs. The CS2 package [14] is used to solve the min-cost network flow for the maximum matching problem in the circuit similarity algorithm.

The number of CLBs (averaged from 1300 to 1506) and levels (averaged from 7 to 11) vary widely in different configurations in our design space case.

Quality of the placement. Due to limited space, we show one representative circuit, "dsip" as an example. The results for the other circuits are similar. Two essential measures in initial placement stage are compared, the bounding box cost and the delay cost. The initial placement results generated by CS and CS-t are significantly better than VPR's random initial placement results. CS reduces the bb cost and delay cost by 76% and 48% compared to VPR, respectively. We also compare the final placement results of circuit "dsip" for 19 designs using a low-temperature annealing after the initial placement. We evaluate the final wire length and the critical delay. For wire length, CS and CS-t produce the results close to VPR's final results, increasing with 32% and 53% overhead, respectively. For critical delay, CS and CS-t achieve better results than VPR, reducing it by 18% and 20%, respectively. More details are included in [13].

The initial and final placement results show the effectiveness of circuit similarity-based placement that generates an optimized initial placement which in turn leads to an optimized final placement. CS-t, geared with aggressive pruning and significantly lower annealing effort, still produces placement with comparable or even better quality than VPR.

Design space shape characterization. In order to characterize the design space, we compare the minimal, median and maximal wire length and critical delay produced by CS and CS-t to VPR. Figure 3 shows the minimal critical delay curves of all 19 designs for 20 circuits using CS, CS-t and VPR. The almost identical curves prove that both CS and CS-t can precisely pinpoint the minimal critical delay design. The curves of both CS and CS-t also follow close to VPR's w.r.t. median and maximum critical delay. The shape of most configurations w.r.t. wire length is accurately matched as well (More details in [13]). The results show that CS and CS-t closely capture the relative critical delay and wire length of each configuration which is essentially useful in the design space exploration.

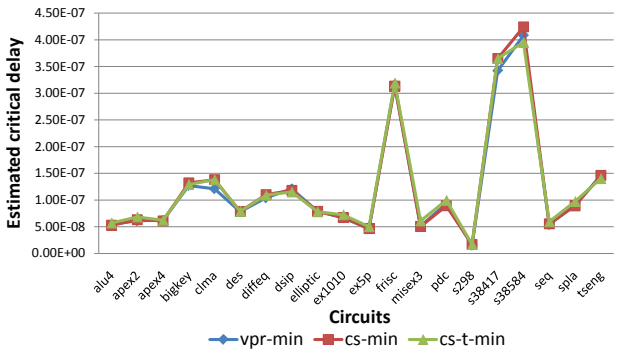


Fig. 3. Minimal estimated critical delay design space shape of 20 circuits on 19 designs

Runtime Comparison. We compare the total runtime of placing 19 designs of each MCNC application using CS, CS-t and VPR. The time of placing the reference configuration (the largest one) is measured separately from the rest of 18 configurations. Without considering the reference configuration placement time, CS achieves averaged 3x speedup while CS-t achieves averaged 30x speedup with up to 100x compared to from-scratch VPR placement. As the number of configurations increases, the reference placement time becomes negligible since it is a one-time cost.

We can take advantage of the significant speedup of CS-t and use it to perform quick design space exploration. For instance, the total time of exploring the whole design space of 20 MCNC applications with 19 designs is more than 8 hours using VPR (place_only). In contrast, it only takes 37 minutes (reference placement time included) using our CS-t. More significant speedup is expected when larger design space is explored.

B. Algorithm-Level Design Space Exploration

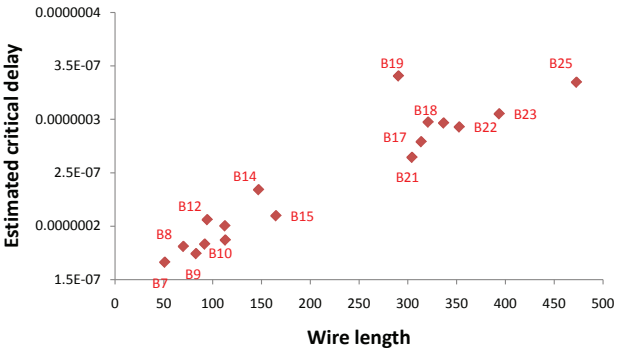
Experimental CAD flow and settings. We now demonstrate the effectiveness of the proposed method at the algorithm-level design space exploration. We use CMU SPIRAL multiplier block generator (with default constants) to generate the RTL design of each configuration based on the Hcub algorithm [7]. We choose the fractional bits as the design parameter and vary it from 7 to 25, resulting in a design space containing 18 configurations (Bits=16 is abandoned since ABC crashed when synthesized it). Once we obtain the RTL design, we use Altera Quartus to perform RTL elaboration and generate a BLIF file from a verilog (.v) file. Other experimental settings are the same as the logic level experiments. Since those configurations vary at algorithm level, the topological structure and circuit size differ considerably compared to logic-level variations. Specifically, the number of CLBs ranges from 501 to 3234 and the number of levels ranges from 35 to 62, respectively.

Experimental results. Figure 4(a) shows the wire length-critical path delay space produced by VPR. The label besides each point indicates the corresponding configuration (*e.g.* "B7" means configuration using Bits=7). Figure 4(b) shows the same design space using CS. From these two figures, we can clearly see that CS and VPR find the same pareto-points, *i.e.*, optimal configurations of this design space, such as $B7$, $B8$ and $B9$. In addition, the overall shapes of the two design spaces match well. This proves that our circuit similarity not only works well at low level logic synthesis, but also at high level algorithm level. Moreover, in terms of runtime, CS and CS-t achieve 7x and 30x speedup compared to VPR, respectively.

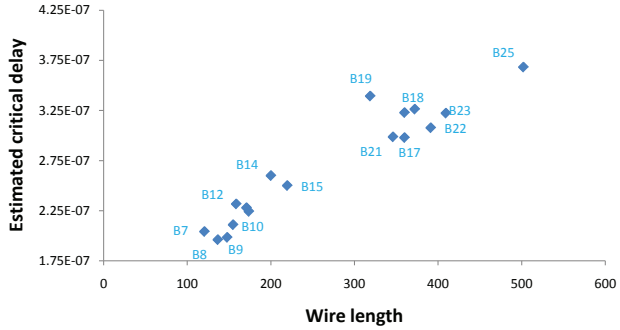
IV. Conclusions and Future Work

In this work, we have presented circuit similarity-based placement for accelerating FPGA design space exploration. The characteristics of each design can be captured by finding the similarity between each configuration and a reference configuration. The experimental results prove that our algorithm works well at both logic level and algorithm level. The shape of the design space can be precisely depicted and design curves can be well matched. Moreover, our CS-t achieves averaged 30x speedup compared to VPR placement. From the perspective of both design space estimation quality and runtime, our circuit similarity has been demonstrated to be a good tool for efficient FPGA design space exploration.

For future work, we will try to apply our circuit similarity to other applications, *e.g.*, FPGA incremental design, FPGA architecture design and FPGA verifications.



(a) Wire length-delay space of VPR for 18 configurations



(b) Wire length-delay space of CS for 18 configurations

Fig. 4. Comparison of wire length-delay space of VPR and CS

References

[1] "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/~alanmi/abc/.

[2] D. Sheldon and F. Vahid, "Making Good Points: Application-Specific Pareto-Point Generation for Design Space Exploration using Statistical Methods," *ACM International Symposium on FPGAs*, 2009.

[3] J. Das, S.J.E. Wilton, W. Luk, P.H.W. Leong, "Modeling Post-Techmapping and Post-Clustering FPGA Circuit Depth," *International Conference on Field-Programmable Logic*, 2009.

[4] A.M. Smith, J. Das, S.J.E. Wilton, "Wirelength Modeling for Homogeneous and Heterogeneous FPGA Architectural Development," *ACM International Symposium on FPGAs*, 2009.

[5] M. Xu and F. Kurdahi, "Area and Timing Estimation for Lookup Table Based FPGAs," *European Design and Test Conference*, 1996.

[6] A. G. Dempster and M. D. Macleod , "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Transactions in Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 42, pp. 569–577, 1995.

[7] Y. Voronenko and M. Pschel , "Multiplierless Multiple Constant Multiplication," *ACM Transactions on Algorithms*, 2007.

[8] "SPIRAL: Software/Hardware Generation for DSP Algorithms," http://spiral.ece.cmu.edu/mcm/gen.html.

[9] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *International Workshop on Field Programmable Logic and Applications*, 1997.

[10] L. Zager, "Graph Similarity and Matching," Ph.D. dissertation, MIT, 2005.

[11] M. Rupp, E. Proschak and G. Schneider, "Kernel Approach to Molecular Similarity Based on Iterative Graph Similarity," *Journal of Chemical Information and Modeling*, 2007.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2001.

[13] X. Shi, D. Zeng, Y. Hu, G. Lin and O. R. Zaiane, "Accelerating FPGA Design Space Exploration Using Circuit Similarity-Based Placement," http://www.cs.ualberta.ca/~xshi/TR10-04.pdf, University of Alberta, Tech. Rep. TR10-04, 2010.

[14] A. V. Goldberg, "An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm," *Journal of Algorithms*, vol. 22, pp. 1–29, 1997.