Max Flow on Pedestrian Transit Networks

Sasha Dolynuk , Alfonso Meraz, Mark Nudelman, Justin Sadler, and Nathaniel Voss

May 5, 2023

Table of Contents

(i) Team Information

Sasha Dolynuk
Alfonso Meraz
Mark Nudelman, mark, U49459775
Justin Sadler
Nathaniel Voss, vossn, U66104742

(ii) Abstract

Abstract

Flow network graphs model the transport of items from a source to a sink using a network of routes where each edge is directed and has a capacity. Maximum flow is an optimization problem that searches for the maximum possible flow rate through a network using multiple paths from source to sink. This problem has many applications including transport systems, the flow of information through computer networks, and image segmentation. The max-flow min-cut theorem states that the maximum possible flow for a network is equal to the minimum capacity over all cuts of the network. By examining the minimum cut of a network, one can analyze how the maximum flow is limited.

This project applies the Edmunds-Karp implementation of the Ford-Fulkerson algorithm to multiple pedestrian transit flow network problems based on the layout of the city of Boston. By developing multiple interpretations of how pedestrians flow through geographical areas, we were able to design multiple graphs based on the same map to analyze limiting factors of max flow. In particular, this project compares undirected vs directed edges, unit capacity vs capacity based on realistic measures, and expanded vs condensed/bundled nodes and examines how all of these factors impact the min cut and max flow of the network.

Additionally, to analyze the asymptotic runtime complexity, multiple flow network graphs of varying node and edge amounts were randomly generated. The runtime for each graph was calculated and plotted to analyze the relationship between nodes, edges, and runtime with the expectation that a time complexity of O(V*E^2) would be revealed.

(iii) Instructions for Running the Code

Instructions

In your terminal, clone the repository, go into the directory and run make.

```
git clone git@github.com:alfonsomeraz/MaxFlowMinCut.git
cd MaxFlowMinCut
make maxflow
./maxflow
```

(iv) Sample Results with Discussion

Sample Results

Max-Flow/Min-Cut on Boston Pedestrian Transit Networks

1. Boston University Graph

Scenario: what is the maximum number of students that can walk from Raising Cane's (on the west end of campus) to Kenmore Square (on the east end of campus) for dessert?

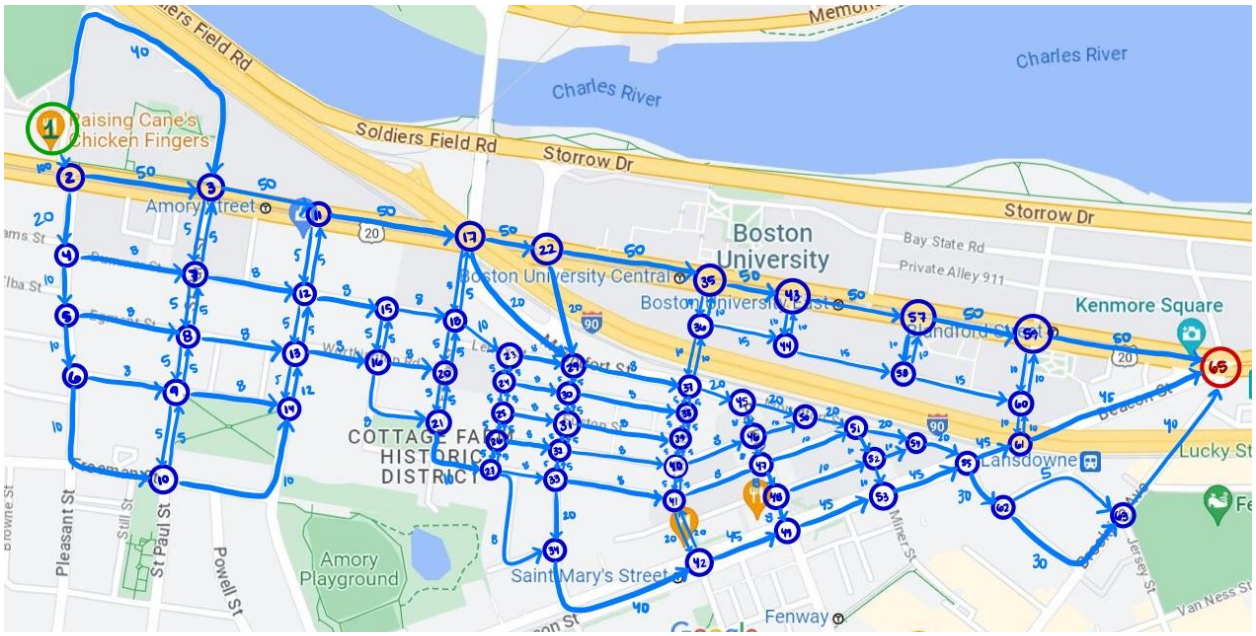

Figure 1. Flow network graph design based on BU campus with capacities of edges assigned by relatively realistic values of street capacities. Source = Raising Cane's, sink = Kenmore Square. Each node is an intersection and each edge is a street.

From this graph structure, we created 2 different graphs, one with unit capacity and one with capacities assigned based on how many people could use the street. For the graph unit capacity, the max flow was only 2, while the alternative capacity graph had a max flow of 66. This is because while the graph structure is the same, the source only has 2 edges radiating out from it. Thus when we use unit capacity, we limit the max flow to however many edges radiate from the source node, which in this case was 2.

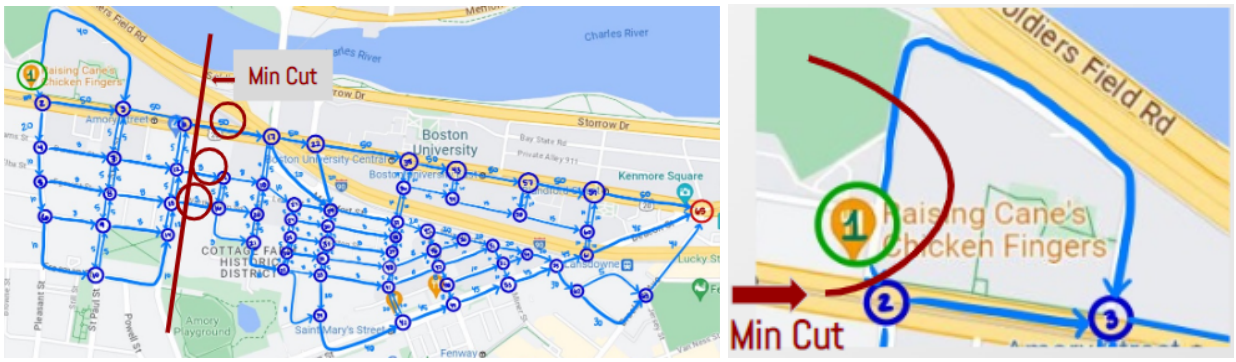

Figure 2. Min cut corresponding to max flow of BU network. Left: realistic relative capacities. Right: unit capacity.

2. Boston Common graph

Scenario: maximum number of pedestrians that can walk from the broken down T at Arlington St. to Park St. to transfer to the red line?

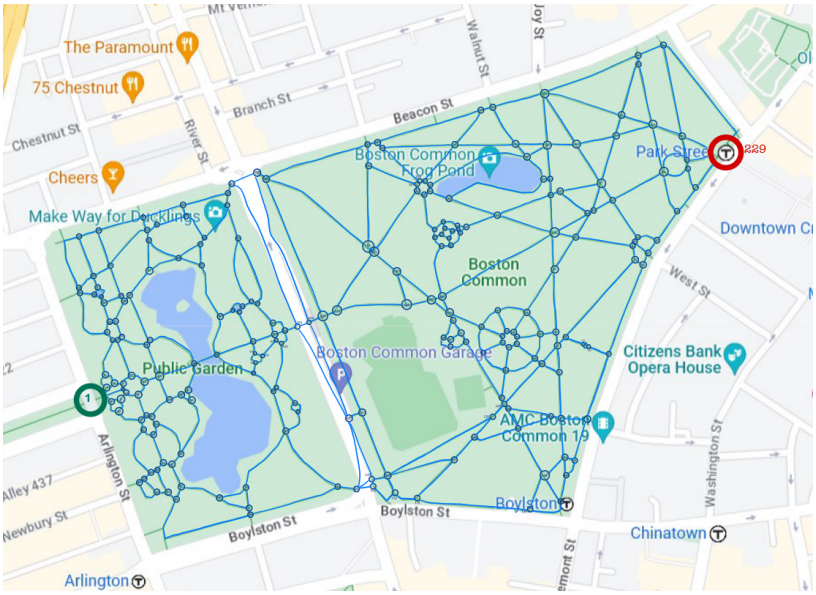

Figure 3. Nodes and edges for Boston Common graph. Source = Arlington St, sink = Park St T stop. Each node is an intersection and each edge is a walkway.

From the graph structure shown in Figure 3, we created 2 different versions: a directed and undirected graph. Our results show that the directed graph has a max flow of only 6 people, while the undirected graph has a max flow of 22 people. We inferred that this stark contrast between how many people can get through the network is because the undirected graph allows for more options for which people can walk in order to maximize flow. The directed graph makes assumptions on the direction that people should walk which may not be correct and may limit flow by preventing flow in a productive direction. Additionally, in real life, most streets and walkways are 2-way (undirected).

Lastly, we designed a consolidated version of the Boston Common network by bundling nodes together. This is shown below in Figure 4. After running our algorithm on this graph and comparing it to the version with expanded nodes, we found that the max flow for this bundled node version was twice as much as the alternative, at 44 pedestrians as opposed to only 22. This increase in flow can be attributed to the reduction in congestion and bottlenecks from the rotary structures by bundling those nodes together.

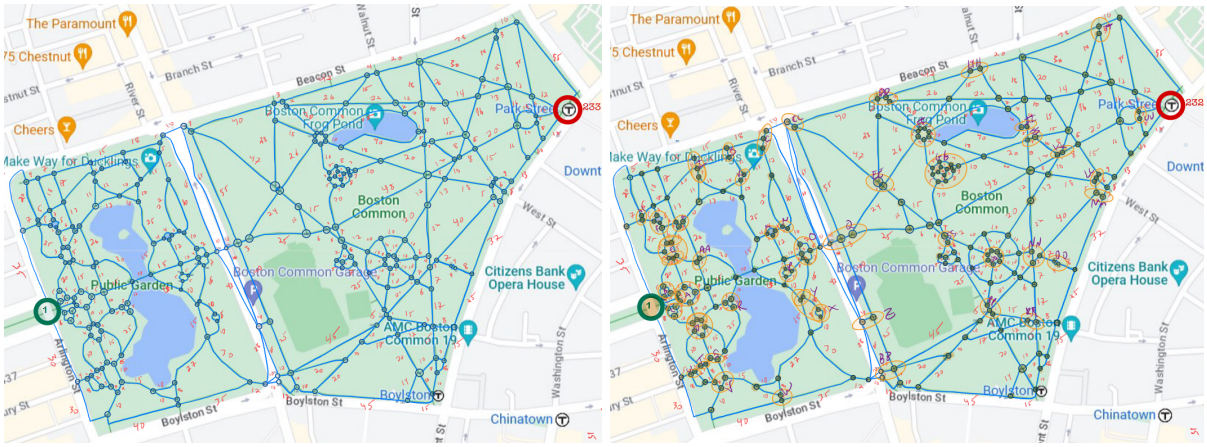

Figure 4. Expanded (left) and consolidated (right) version of the Boston Common flow network.

Our code produces an output file containing the number of nods, number of edges, max flow, and runtime for each csv graph file analyzed. This is pictured below in Figure 5.

```
Ford-Fulkerson Algorithm
------------------------
    Nodes   |    Edges    |    Flow    |     Time     |     File
    0232            380          006       5.606e-03 ms    BostonCommonDirected.csv
    0232            751          022       1.298e-02 ms    BostonCommonUndirected.csv
    0232            751          003       3.728e-03 ms    BostonCommonUndirectedUnitCap.csv
    0232            380          002       3.241e-03 ms    BostonCommonDirectedUnitCap.csv
    0135            525          044       1.223e-02 ms    BostonCommonConsolidatedUndirected.csv
    0135            267          020       4.044e-03 ms    BostonCommonConsolidatedDirected.csv
    0062            131          066       7.323e-04 ms    flowBU.csv
    0065            131          002       4.537e-04 ms    flowBU_UnitCap.csv
```

Figure 5. Output file with flow and runtime for all scenarios described above.

We can note that, as expected, graphs with less edges and nodes have a shorter runtime. However, we also see that for graphs with the same number of nodes and edges, there can still be significantly different runtimes. This is best exemplified by a comparison between the regular and unit capacity graphs: each unit capacity graph has a runtime <⅔ of its corresponding varying capacity graph. We can attribute this to the reduction in complexity of calculating the residual graph in the Ford-Fulkerson algorithm.

Discussion

Max-Flow/Min-Cut on Boston Pedestrian Transit Networks

We analyzed different factors that impact the max flow of a flow network. We determined that using unit capacity vs realistically relative capacity values limits the max flow to the amount of edges radiating from the source node if that is the min cut of the graph. Additionally, unit capacity graphs have a faster runtime than their counterparts. We also found that directed graphs limit flow by blocking paths in certain directions, thus to ensure maximum flow, an undirected graph is best. Finally, bundling nodes that are spatially close together reduces congestion and bottlenecks, improving the maximum flow.

(v) Context & References

Context

We initially set out to implement the recent (Chen et al) [8] “near-linear time” algorithm for the max flow/min-cost-flow problem. However, the near-linear time solution turned out to be overly complex to implement as it uses some dynamic “tricks” beyond those found in purely combinatorial algorithms, and thus was beyond the scope of this class.

We therefore shifted our objective to test an implementation of the Edmonds-Karp (BFS) version of the Ford-Fulkerson Max Flow Algorithm, experimenting on its performance on different graph structures. We specifically examined its performance with different graph density and node layer levels of pedestrian crowd transit problem graphs of (a) the max flow of students who can traverse the length of Mass. Ave (Agganis to Fenway) and (b) the max flow of pedestrians who can traverse from the Arlington St. station area to the Park Street stop should there be an MBTA breakdown between stops necessitating traversing via foot.

Although we were not able to implement the near-linear time version of the max flow algorithm, to further explore some of the context behind the max flow problem we are including a brief history of the Max flow algorithm below. Within this selected history we will predominantly focus on some of the earlier advances amongst combinatorial approaches and the different “methods” advancements that influenced the more recent near-linear time implementations for specific max flow problem types. We thenl wrap up with a brief overview of the near linear time approach.

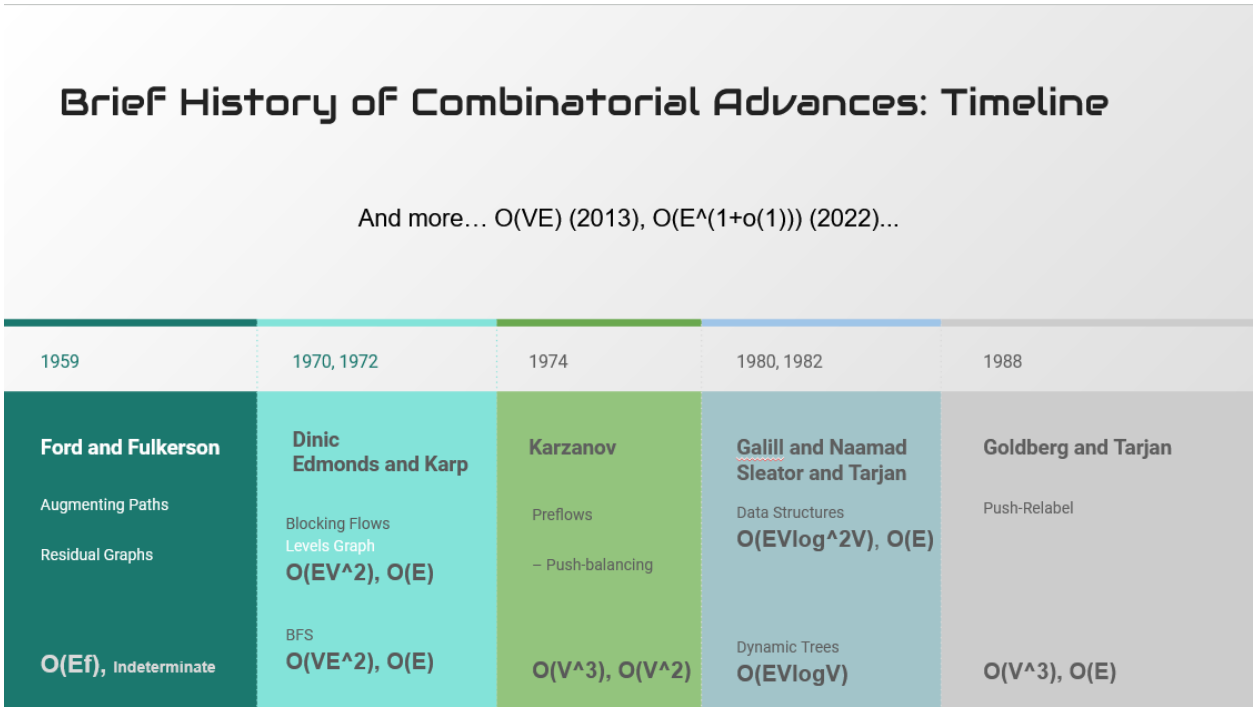

Figure 6. Selected Timeline of Early Max Flow Methods Advances.

SELECTED HISTORY OF EARLY COMBINATORIAL MAX FLOW METHODS

The Max Flow Algorithm, and its special case, the Min-Cost Flow Algorithm - in which costs are assigned for transiting different arcs - are widely used algorithms with a broad range of real world applications, such as in transportation networks and resource allocation.

An algorithm solving the Max Flow problem was first developed by Ford and Fulkerson [1] in 1959, though with an indeterminate run time for non-integer capacities (both rational and irrational) due to unbounded run time, with the chance of the algorithm never terminating if the capacities were irrational. If capacities were integers, run time was O(Vf) where V denotes number of vertices in the graph, and f denotes the max flow value. Space utilization of this initial algorithmic solution was O(E), where E is the edge count. To solve the Max Flow problem, Ford and Fulkerson developed an algorithm that visited each path in which flow could be augmented, and updated a secondary 'residual graph' that tracked the updated flows. Fully saturated edges could experience "backflow" if another edge pushed flow into the same head node that was also linked to the tail of a full capacity arc. Due to this there were an unbounded number of path-arc-flow combinations depending on the arc capacity type, causing the indeterminate time issue.

In 1972 Edmonds-Karp [3] added structure to the augmented path search order making the Ford Fulkerson algorithm capable of running in strongly polynomial time O(E^2V) by using a breadth first search order (assigning distance weights of unit (1) to each arc) to examine the paths without loops. Due to this, the algorithm progresses by examining paths with arc counts in a monotonically non-decreasing manner. Once an arc is fully saturated in one of these iterations it is removed from future considerations for longer path lengths, bounding the problem for all capacity types (rational, irrational, etc).

In 1970 Dinic [2] introduced a method with runtime O(V^2E), navigating a "levels graph" that iteratively updates by relabeling nodes based on distance from the sink. This solves similarly to the Edmonds-Karp method, with the Dinic method focused on identifying the blocking flows at every level corresponding to the monotonically non-decreasing levels graph iterations.

In 1974 Karzonov [4] published a paper in which he considered preflows in the max flow problem on layered networks (akin to levels graphs within Dinic's algorithm - where all paths on in the graph are the same length). Preflows are functions on the arcs, rather than on the augmented paths, that allow for exceeding the flow conservation principle on transit nodes (i.e. non-terminal nodes). When examining preflows, blocking flow is identified by any directed path's fully saturated arcs. This is done via alternating pushing and balancing iterations as part of the prefill algorithm (solving the auxiliary (inner) problem that finds the blocking flows). The prefill algorithm for the auxiliary problem runs in O(V^2) over V stages (corresponding to the max different potential layers) resulting in a total runtime of this max flow approach of O(V^3).

Karzanov's preflow concept led to further advances in the max flow algorithm, amongst them the Goldberg and Tarjan[7] (1986 conference/1988 paper) push-relabel method. Via this method, preflow is maintained, while the algorithm updates via push operations. Labeling occurs at the local level (specific only to a single node/vertex and arc/edge) versus the global path layers labeling approach of prior algorithms. This allows for progressively pushing forward each arc to capacity, updating the height of a specific node within a dynamic tree structure, then sending excess flow (akin to being stored in a reservoir at each node) back to the source, and incrementing the node height to account for each forward and backward push. Via this method it is possible to visit each node in the graph and find the optimal flow, terminating when all nodes have been visited using a queue/FIFO ordering, rather than considering every path combination. The Push-Relabel method is the fastest general purpose max flow algorithm, parts of which are the foundation upon which the faster "near-linear time" max flow algorithms are based.

LP interior point (*outer problem*) **+** Combinatorial (*inner problem*) **+** dynamic amortized time (*inner problem*)

Figure 7. Combination of Methods for Near-Linear Time Solution that result in E^(1+o(1)) time complexity.

BRIEF OVERVIEW OF NEAR-LINEAR TIME MAX FLOW/MIN COST APPROACH

Chen et. al's (2022) [8] "Near-linear time" approach was informed by the observation that Max Flow algorithms all tend to solve an "outer problem" via an outer algorithm (examining the graph in stages) and an "inner problem" via an inner algorithm (iteratively examining the graph components) , with most recent algorithmic improvements applying LP approaches to the inner algorithm [9].

In their paper, Chen et. al. focused on improving the data structure for the **outer** algorithm. They did this by creating a special minimum-ratio cycle data structure. The min-ratio cycle data structure uses an interior point LP method to solve the outer problem. It does this by reducing the min-cost flow to a sequence of minimum-ratio cycle computations on a slowly changing graph with the objective to approximately maintain the min-ratio cycles while recursively creating a hierarchy of graphs with fewer and fewer nodes and edges.

Chen et. al's approach includes other refinement "tricks" to further dynamically reduce the graph size, including an algorithm for manipulating the data so it is always only examining cycles that are relevant to further optimization (omitting non-salient cycles). SImilarly, they include use of a dynamic spanner data structure to reduce the number of edges at each level of the recursion hierarchy. The min-ratio cycle data structure solves the min-cost ratio outer problem in E^o(1) time, which when combined with their inner algorithm solution - merging a combinatorial approach and an amoritized time dynamic approach - results in total runtime of O(E^(1+o(1)).

Although extremely fast max flow algorithms are not necessary for solving currently known engineering problems - such as our pedestrian max flow problem, which was sufficiently solved on a standard personal computer using the Edmonds-Karp approach - from a competitive

mathematics standpoint the Chen et. al. paper successfully demonstrated methods that may be fruitful for future research and applications beyond the standard max flow problem.

References

[1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009). "Section 26.2: The Ford–Fulkerson method". Introduction to Algorithms (Third ed.). MIT Press and McGraw–Hill. pp. 651–664. ISBN 0-262-03293-7.

[2] Dinic, E.A. (1970) Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. Soviet Math Doklady, 11, 1277-1280.

[3] Edmonds, Jack; Karp, Richard M. (1972). "Theoretical improvements in algorithmic efficiency for network flow problems" (PDF). Journal of the ACM. 19 (2): 248–264. doi:10.1145/321694.321699. S2CID 6375478.

[4] Karzanov, Alexander. (1974). Determining the maximal flow in a network by the method of preflows. Doklady Mathematics. 15. 434–437. https://dl.acm.org/doi/10.1145/48014.61051

[5] Zvi Galil, Amnon Naamad,An O(EVlog2V) algorithm for the maximal flow problem,Journal of Computer and System Sciences,Volume 21, Issue 2,1980,Pages 203-217,ISSN 0022-0000,https://doi.org/10.1016/0022-0000(80)90035-5. (https://www.sciencedirect.com/science/article/pii/0022000080900355)

[6] Daniel D. Sleator, Robert Endre Tarjan, A data structure for dynamic trees, Journal of Computer and System Sciences,Volume 26, Issue 3,1983,Pages 362-391,ISSN 0022-0000, https://www.sciencedirect.com/science/article/pii/0022000083900065

[7] Andrew V. Goldberg and Robert E. Tarjan. 1988. A new approach to the maximum-flow problem. J. ACM 35, 4 (Oct. 1988), 921–940. https://doi.org/10.1145/48014.61051

[8] Chen, L., Kyng, R., Liu, Y. P., Peng, R., Gutenberg, M. P., & Sachdeva, S. (2022, October). Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)* (pp. 612-623). IEEE.

[9] Sachdeva, S. (2022, October). Almost Linear Time Algorithms for Max-flow and More [video]. Available: https://www.ias.edu/video/almost-linear-time-algorithms-max-flow-and-more