

Reducing Directed Max Flow to Undirected Max Flow

Henry Lin

Division of Computer Science, U.C. Berkeley. Email: henrylin@eecs.berkeley.edu.

Abstract. In this paper, we show that the directed maximum flow problem can be reduced to the undirected maximum flow problem. Our result yields a new algorithm for finding maximum flows in directed graphs, by reducing any directed maximum flow instance into an undirected maximum flow instance and running the Karger-Levine algorithm to find a maximum flow in the resulting undirected graph. Given a directed graph $G = (V, E)$ with n nodes, m edges, maximum flow value v , and capacities c_e , our reduction yields a $\tilde{O}(m + n(v + i))$ algorithm for finding a maximum flow in G , where i is the imbalance of the graph and defined to be $i = \sum_{u \in V - \{s, t\}} \left| \sum_{(w, u) \in E} c_{(w, u)} - \sum_{(u, w) \in E} c_{(u, w)} \right|$. For directed graphs with small flow value and small imbalance, this yields a faster algorithm for computing maximum flows.

1 Introduction

The maximum flow problem is one of the oldest and most well-studied problems in computer science. In the maximum flow problem, we are given a graph $G = (V, E)$ with a source $s \in V$, a destination $t \in V$, and a capacity c_e along each edge $e \in E$, and we seek to find a maximum flow from s to t such that the flow along each edge e is at most c_e . This maximum flow problem was first defined over 50 years ago, and has a long history. Many algorithms have been derived for the problem and different running time bounds have been proved for the algorithms under various conditions.

The maximum flow problem has four main variants, depending on whether or not the graph G is directed or undirected, and depending on whether or not the capacities c_e are allowed to be arbitrary integers or are restricted to be unit capacity ($c_e = 1$ for all $e \in E$). For conciseness, in this paper we will refer to these four main variants of the max flow problem as the *general directed max flow problem*, the *general undirected max flow problem*, the *unit capacity directed max flow problem*, and the *unit capacity undirected max flow problem*. (There are also results known for the max flow problem when the c_e are only required to be real numbers, but we refer the reader to [5] for more details).

Although it would be difficult to mention all the known maximum flow results, we mention some highlights in the history of maximum flow algorithms, and refer the reader to [5] for a more complete history. In the following results on finding a maximum flow in a graph $G = (V, E)$, we let $n = |V|$ be the number of nodes, $m = |E|$ be the number of edges, and v the value of the maximum flow of G . One of the first algorithms for finding maximum flows in graphs was due to Ford and Fulkerson in 1956 [8]. Their algorithm solved the general directed max flow problem in $O(mv)$ time, and for sparse graphs with small flow values, the original Ford-Fulkerson algorithm is still the best known algorithm.

In the 1970's, Dinic [1] and Edmonds and Karp [2] independently derived the first strongly polynomial time algorithms for the general directed max flow problem. Edmonds and Karp proved an $O(nm^2)$ running time bound for the shortest augmenting path algorithm, while Dinic utilized the concept of blocking flows to obtain an $O(mn^2)$ time algorithm [1]. Later developments were able to improve the best running time for the general directed max flow problem to $\tilde{O}(mn)$ (see [5] exact details and a more complete history), and most recently in 1997, Goldberg and Rao were able to beat the $\tilde{O}(mn)$ barrier by developing a $O(\min(m^{3/2}, mn^{2/3}) \cdot \log(n^2/m) \cdot \log v)$ time algorithm for the general directed max flow problem [6].

In terms of the unit capacity directed max flow problem, Even and Tarjan [3] and Karzanov [12] independently proved that Dinic's algorithm runs in $O(\min(m^{3/2}, mn^{2/3}))$ time, and this is the best known bound for unit capacity directed graphs, besides the Ford-Fulkerson bound of $O(mv)$.

Of course, all of the above results for finding max flows in directed graphs also extend to finding max flows on undirected graphs, as there is a simple well-known reduction from the undirected max flow problem to the directed max flow problem: simply replace each undirected edge $\{u, v\}$ with two directed edges (u, v) and (v, u) and set them to have the same capacity as the original undirected edge $\{u, v\}$. Even though, it was well-known that the undirected max flow problem was easier than the directed max flow problem, Karger was the first to show a result that indicated the undirected max flow problem might be *strictly* easier than the directed max flow problem.

More specifically, in 1994, Karger showed that a maximum flow could be found in $\tilde{O}(mv/\sqrt{c})$ time for undirected graphs with connectivity c [9]. Karger's seminal work in 1994 eventually culminated in the Karger-Levine randomized algorithm for the general undirected max flow problem, which ran in $\tilde{O}(m + nv)$ expected time [10], and this is still the best known algorithm for finding

maximum flows in undirected graphs with small v values. (See [10] for the best deterministic algorithms). It has been a long standing open question whether or not the Karger-Levine algorithm [10] could also be extended to yield an $\tilde{O}(m + nv)$ algorithm for the directed max flow problem as well.

One approach to deriving an $\tilde{O}(m + nv)$ directed max flow algorithm would be to find an efficient reduction from the directed max flow problem to the undirected max flow problem and apply the Karger-Levine algorithm. Although this would be a very natural approach, to the best knowledge of the author, there have not been previous reductions from the directed max flow problem to the undirected max flow problem. There is a reduction from the directed min cut problem to the undirected min cut problem shown by Picard and Ratliff [16], but the reduction does not appear to immediately imply that the directed max flow problem can be reduced to the undirected max flow problem.

Although the problem of finding the *value* of a maximum flow in a graph is known to be computationally equivalent to finding the *value* of a minimum cut in the same graph, via the max-flow min-cut theorem, the problem of finding an actual maximum flow is not known to be computationally equivalent to finding a minimum cut. For example, given a minimum cut in a graph, it is not clear whether or not it can be used to find a maximum flow in the same graph efficiently, say in $O(m)$ time. As a result, the reduction from directed min cut to undirected min cut only implies immediately that the problem of computing the *value* of a maximum flow in a directed graph can be reduced to the undirected min cut or undirected max flow problem.

In this paper, we improve on this prior work and show that finding a maximum flow in a directed graph can be reduced to finding a maximum flow in an undirected graph. Coincidentally, a preliminary and simple version of our reduction essentially yields the same undirected graph as the one generated by the reduction in Picard and Ratliff's paper [16], however, the inspiration for our reduction comes from an entirely different perspective. Moreover, we show that a maximum flow in the resulting undirected graph can be used to reconstruct a maximum flow for the original directed graph, and additionally, we introduce some modifications, which improve the efficiency of the final reduction.

In particular, our final reduction from the general directed max flow problem to the general undirected max flow problem, combined with the Karger-Levine algorithm, yields an $\tilde{O}(m + n(v + i))$ algorithm for computing maximum flows in general directed graphs, where i is the imbalance of the graph and defined to be $i = \sum_{u \in V - \{s, t\}} |\sum_{(w, u) \in E} c(w, u) - \sum_{(u, w) \in E} c(u, w)|$. In other words, the imbalance of the graph is simply the sum over all nodes $u \neq s$ or t , the absolute value of the difference between the capacity of edges into u and the capacity of edges out of u . For the special case of unit capacity graphs, the imbalance of a graph is simply the sum over all nodes $u \neq s$ or t , $|\text{indegree}(u) - \text{outdegree}(u)|$. Our reduction is fairly simple, yet appears not to be known. For graphs with small flow value v and small imbalance i , our reduction yields a faster maximum flow algorithm than previously known algorithms. For example, if $v = O(n)$ and $i = O(n)$, then our algorithm runs in $\tilde{O}(n^2)$, which is better than previous results.

In the couple sections, we give some intuition for our reduction, and explain a preliminary version of our reduction. We then improve and extend our preliminary reduction until we have our final reduction, and conclude with some open problems and acknowledgements. In this paper, we will assume readers are familiar basic concepts related to finding maximum flows in graphs – for example, the concept of residual graphs and decomposing flows in linear time. For more background, there are many standard textbooks or references available, see [13] for example.

2 Reduction from Directed Max Flow to Undirected Max Flow

In this section, we describe how the general directed max flow problem can be reduced to the general undirected max flow problem. By applying the Karger-Levine max flow algorithm for undirected graphs [10], we can then achieve an $\tilde{O}(m + n(v + i))$ time algorithm for finding maximum flows in directed graphs, where i is the imbalance of the graph, defined as $i = \sum_{u \in V - \{s, t\}} |\sum_{(w, u) \in E} c(w, u) - \sum_{(u, w) \in E} c(u, w)|$. Before describing the reduction for graphs with general integer capacities, we start by showing how the unit capacity directed max flow problem can be reduced to the unit capacity undirected max flow problem. Given the reduction for unit capacity graphs, it will not be hard to see how the same idea also yields a reduction for graphs with general integer capacities.

We now describe a high level picture of the reduction from the unit capacity directed max flow problem to the unit capacity undirected max flow problem. Given a directed graph $G = (V, E)$ with n nodes, m edges, and max flow value v , we construct in $O(m)$ time an undirected graph $G' = (V, E')$ with n nodes, $O(m)$ edges, and max flow $(v + i)$, where i is the imbalance defined above. In the unit capacity setting, the imbalance is simply $i = \sum_{u \in V - \{s, t\}} |\text{indegree}(u) - \text{outdegree}(u)|$. We then run the Karger-Levine algorithm on the undirected graph G' to find a maximum flow f'_{kl} in G' in $\tilde{O}(m + n(v + i))$ time. Once we have found a maximum flow f'_{kl} for the graph G' , we can then modify f'_{kl} in $O(m)$ time to get a maximum flow for G . Thus, the final running time of our algorithm for the unit capacity directed max flow problem will be $\tilde{O}(m + n(v + i))$.

2.1 A Simple Lemma for Intuition

Before we describe how the graph G' is constructed and define the reduction exactly, let us first explain why one might expect that an algorithm for finding maximum flows in undirected graphs should also be able to find maximum flows in a directed graphs as well. Notice that if we follow any of the standard augmenting path algorithms for computing a maximum flow in an undirected graph, once we start sending flow in the undirected graph, we are left with a directed residual graph, and thus a directed max flow instance to solve. Thus, algorithms for finding a maximum flow in an undirected graph also seem to inadvertently require finding a maximum flow in a directed graph. Can we exploit this weak intuition in order to build a formal reduction? We formalize this intuition a bit further by proving the following simple lemma, which will be useful for our reduction:

Lemma 1. *For any directed graph $G = (V, E)$, there exists an undirected graph $G' = (V, E')$ and a flow f'_{init} , such that the residual graph of G' with respect to the flow f'_{init} is “essentially equivalent” to the directed graph G .*

Without being too formal, the two graphs will be essentially equivalent in the sense that the residual graph will be the same as the graph G , except it will have additional edges into s and out of t , and each edge will have capacity 2. We call these two graphs essentially equivalent, because finding a maximum flow in one graph is more or less equivalent to finding a maximum flow in the other.

Proof. To prove the lemma, we need to define an undirected graph G' and a flow f'_{init} for any given directed graph G , and show that the residual graph of G' w.r.t. f'_{init} is essentially equivalent to G . To define our undirected graph G' , we set $G' = (V, E')$ to have the same nodes as $G = (V, E)$, and for each edge $(u, v) \in E$, we add the following three edges to G' : (a) $\{s, v\}$, (b) $\{u, v\}$, and (c) $\{u, t\}$. For notational purposes, we call these three types of edges (a) *new source edges*, (b) *original*

edges, and (c) *new destination edges*, respectively. Note that G' could be a multigraph, as it may have multiple edges between different pairs of nodes. Now to define the flow f'_{init} , f'_{init} will be a flow that sends m units of flow from s to t as follows: for every edge (u, v) in G , we send one unit of flow along the path $s-v-u-t$ in G' .

Now that we have defined G' and f'_{init} , let us check that the residual graph of G' with respect to f'_{init} is essentially equivalent to the directed graph G . Note that for every edge (u, v) in G , there is an undirected edge $\{u, v\}$ in G' and one unit of flow traveling from v to u . As a result, in the residual graph of G' , this flow subtracts one unit of capacity along the edge (v, u) and adds one unit of capacity along the edge (u, v) . Thus the residual graph has no units of capacity along (v, u) and two units of capacity along (u, v) .

Furthermore, for any new source edge $\{s, v\}$ in G' , there is one unit of flow traveling along (s, v) , which leaves two units of capacity along the edge (v, s) and no units of capacity along (s, v) . Similarly, for the new destination edges $\{u, t\}$, the residual graph has two units of capacity along (t, u) and no units of capacity along (u, t) . Thus, the residual graph of G' with respect to flow f'_{init} is the same as G , except each edge $(u, v) \in E$ has capacity 2, and the residual graph has additional edges with capacity 2 into s and out of t . Even though these two graphs are different, they are nonetheless “essentially equivalent” as we defined above.

3 A Preliminary Reduction

For reasons that will become clear later, now let us assume that we are trying solve a maximum flow problem on a directed graph G , which has capacity 2 on each edge. Given the above lemma, we can now define a simple way to reduce the directed max flow problem on G to a unit capacity undirected max flow problem. Given a directed graph G with capacity 2 along each edge, we can solve the maximum flow problem on G , via the following reduction:

- Given the directed graph G , construct the undirected graph G' , as described in Lemma 1.
- Run Karger-Levine to find a maximum flow f'_{kl} in G'
- Construct f'_{init} as described in Lemma 1, and return $r(f'_{kl} - f'_{init})$ as a maximum flow for G

In the above reduction, the function r in the last step is a function that takes the flow $f'_{kl} - f'_{init}$, and removes any flow cycles, before returning it as a maximum flow for G . The function r can be implemented in $O(m)$ time by using a standard flow decomposition algorithm to decompose $f'_{kl} - f'_{init}$ into a set of simple s to t paths and flow cycles, and returning the set of simple s to t paths. Also note that the first step runs in $O(m)$ time, but for now let us not worry about that the value of the maximum flow of G' and running time of the Karger-Levine step. (The value of the maximum flow will be more than $v + i$ as we wanted, but we’ll fix that later).

Let us first prove that the above algorithm works. To prove that our algorithm works, we need to prove that $f = r(f'_{kl} - f'_{init})$ is a valid flow for G and that f a maximum flow for G . In particular, we need to prove the following three statements:

1. f sends v_G units of flow from s to t , where v_G is the value of the maximum flow of G
2. f sends at most two units along any edge $(u, v) \in E$, and no units of flow from v to u .
3. f does not send any flow on any new source edges, nor any new destination edges.

To prove statement 1, let us first compute $v_{G'}$, the value of the maximum flow of G' . Note that if we send the flow f'_{init} through G' , we have sent m units of flow through G' , and have a

residual graph which is essentially equivalent to the directed graph G . Thus, if v_G is the value of the maximum flow of the directed graph G with capacity 2 along each edge, then the value of the maximum flow of G' is $v_{G'} = v_G + m$. Therefore, if f'_{kl} is a maximum flow for G' , then it must send $v_{G'} = v_G + m$ units of flow from s to t . Now, if we subtract f'_{init} from f'_{kl} , we must have exactly v_G units of flow traveling from s to t . Moreover, removing flow cycles from $f'_{kl} - f'_{init}$ does not change the amount of flow sent from s to t , so we know f must send exactly v_G units of flow from s to t , and statement 1 holds.

To prove statement 2, note that for every edge $(u, v) \in E$, we created an undirected edge $\{u, v\}$ in G' . Thus, the flow f'_{kl} may send -1, 0, or +1 units of flow from u to v (-1 units of flow from u to v means one unit of flow travels from v to u). However, note that by definition $-f'_{init}$ sends +1 unit of flow from u to v so that if we add the two flows, we see that $f'_{kl} - f'_{init}$ sends either 0, +1, or +2 units of flow from u to v , and no units of flow in the opposite direction. Furthermore, this statement is true even after removing flow cycles from $f'_{kl} - f'_{init}$, and thus statement 2 holds as well.

To prove statement 3, we show that $r(f'_{kl} - f'_{init})$ does not have any flow on any new source edges. The argument for proving that the new destination edges do not have any flow is exactly the same. To prove that $r(f'_{kl} - f'_{init})$ does not have any flow on any new source edges, consider a new source edge $\{s, v\}$. We know that f'_{kl} may have sent -1, 0, or +1 units of flow from v to s , but $-f'_{init}$ sends +1 unit of flow from v to s . Thus, if we add the two flows $f'_{kl} - f'_{init}$ must send 0, +1, or +2 units of flow from v to s . This means that if $f'_{kl} - f'_{init}$ is sending flow along the edge $\{s, v\}$, it must be going into s . However, if the flow is going into the source, then that must mean that it is participating in a flow cycle. As a result, it is clear that after removing flow cycles, there must not be any flow remaining on the new $\{s, v\}$ edges. A similar argument can be made for the new destination edges, and thus, statement 3 holds as well.

4 Improving the Running Time and Extending the Reduction

We have proved our reduction works, but what about the running time? As we argued above, the reduction yields a graph G' with a maximum flow value of $v + m$, where v is the value of the maximum flow of the directed graph G and m is the number of edges of G . Thus our algorithm has a running time of $\tilde{O}(m + n(v + m))$, and not $\tilde{O}(m + n(v + i))$ as promised.

However, note that our reduction was somewhat inefficient as it often adds a pair of new source and new destination edges to a single node v , i.e. it adds the edges $\{s, v\}$ and $\{v, t\}$. It is not hard to show that if we remove these pairs of edges from G' and remove the flow that f'_{init} sends on these edges, we are left with a new graph G'' and new flow f''_{init} such that the residual graph of G'' with respect to flow f''_{init} is still essentially equivalent to G . Moreover, for edges leaving s or entering t in the original graph G , we can just add those edges as undirected edges to G'' , without adding any additional new source or new destination edges, or sending any f''_{init} flow them. Lastly, for edges entering s or leaving t , we can just ignore those edges without influencing the value of the maximum flow. With these improvements in mind, it is not hard to see we have found a undirected graph G'' and a flow f''_{init} , such that the residual graph of G'' w.r.t f''_{init} is essentially equivalent to G , and G'' has maximum flow $(v + i)$ as defined. Furthermore, applying the reduction defined in the previous section with the graph G'' instead of G' yields a correct reduction as well, and the running time of the new reduction is $\tilde{O}(m + n(v + i))$ as promised.

Also note that we solved the directed maximum flow problem on a graph G with capacity two along each edge. If we would like a solution to the maximum flow problem on a graph G with unit

capacity on each edge, we can simply apply the same reduction and divide the final flow by 2. This flow will be a maximum flow for the unit capacity version of the problem, although it may not be integral. If one would like an integral max flow, it turns out there is a simple procedure to convert a half integral flow to an integral flow in linear time, which we prove in the next subsection. We believe this procedure may be novel as well, and may be of independent interest.

Finally, in order to describe how one might reduce the general directed max flow problem to the general undirected max flow problem, note that a similar reduction can be derived when edges are allowed to have integer capacities. Simply consider each edge (u, v) with capacity $c_{(u,v)}$ as $c_{(u,v)}$ unit capacity edges between (u, v) , and apply the same reduction works as before. The reduction still holds and the Karger-Levine algorithm still works for undirected graphs with integer capacities. Thus, we have a valid reduction, and the reduction can be implemented efficiently, so that the total running time is $\tilde{O}(m + n(v + i))$ as stated.

4.1 Converting a Half-Integral Flow to an Integral Flow

To convert a half integral flow to an integral flow, let us consider the equivalent problem of converting a flow f which sends 0, 1, or 2 units of flow along each edge into a flow of equal or greater value which sends either 0 or 2 units of flow along each edge. In order to do so, let us consider the set of edges E_f , along which f sends one unit of flow. It is clear that for any node $v \neq s$ or t , the number of edges of E_f which are adjacent to v must be an even number due to flow conservation and the fact that all of v 's edges not in E_f have an even amount of flow. Thus, if we treat the edges E_f as undirected, it must be the case that we can be decompose E_f into a set of cycles and s to t paths.

Now consider a cycle consisting of edges in E_f , and consider traversing the cycle in an arbitrary direction. This traversal may traverse some edges in E_f in the correct direction, and some edges in E_f in the reverse direction. Note that if we add one unit flow along each edge in the cycle, in the direction we traversed the cycle, we still have a flow that satisfies flow conservation, and we have not changed the value of the flow (the total amount of flow leaving s or entering t). Moreover, this addition has the effect of adding one unit of flow to each edge we traversed in the correct direction, and subtracting one unit of flow to each edge we traversed in the reverse direction. Thus, this addition converts each edge in E_f into an edge with 0 or 2 units of flow depending on whether or not the edge was traversed in the correct or reverse direction. If we repeat this process for all edges in E_f occurring in cycles, we can then convert all these edges into to edges with 0 or 2 units of flow, without changing the amount of flow sent from s to t .

Now let us consider edges in E_f appearing in an s to t path, and consider traversing the path from s to t . If we add one unit of flow along each edge in the direction which we traversed it, we increase the amount of flow sent from s to t , and maintain flow conservation at each intermediate node. Furthermore, this process adds one unit of flow to each edge we traverse in the correct direction, and subtracts one unit of flow to each edge we traverse in the reverse direction. As before, this process converts the flow along each edge we traverse into having value 0 or 2. Thus, by repeating this process, we can convert our original $\{0, 1, 2\}$ flow into a $\{0, 2\}$ flow with equal or greater flow sent from s to t in $O(m)$ time, for a flow consisting of m edges.

5 Conclusion

To conclude, we finish with some open questions. In our reduction from the directed max flow problem to the undirected max flow problem, we saw that the only barrier towards getting an $\tilde{O}(m + nv)$ algorithm for finding maximum flows in directed graphs was the fact that our reduction created a graph G' with maximum flow value $v + i$. Is there a more clever reduction that generates an undirected graph G' with max flow value $O(v)$, which can be used to solve the max flow problem on the directed graph G ? A more efficient reduction could yield an $\tilde{O}(m + nv)$ algorithm for computing maximum flows in directed graphs.

Alternatively, it may also be interesting to explore the known reductions from the directed max flow problem to the bipartite matching problem. As the bipartite matching problem can be solved faster than the directed max flow problem, exploring this direction may also yield a faster directed maximum flow algorithm, since the bipartite matching problem can be solved in $O(m\sqrt{n})$ time [7], or $O(n^\omega)$ time [15], where $\omega \approx 2.376$ is the matrix multiply constant. With the application of Benes networks, the reductions shown in [4, 11, 14, 17] can be made more efficient in reducing the unit capacity directed maximum flow problem on a graph with n nodes and m edges, into a bipartite matching problem on a graph with $\tilde{O}(m)$ nodes and $\tilde{O}(m)$ edges. The reductions are not efficient enough to produce a faster directed maximum flow algorithm, but can the reductions be improved so that the resulting bipartite matching problem only has $\tilde{O}(n)$ nodes and $\tilde{O}(m)$ edges?

6 Acknowledgements

The author would like to thank Satish Rao for providing useful suggestions and informative discussions. The author would also like to thank previous anonymous reviewers for pointing out references to useful related work.

References

1. E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Dokl.* 11, pages 1277–1280, 1970.
2. Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. In *Journal of the ACM* 19 (2), page 248264.
3. S. Even and R. E. Tarjan. Network flow and testing graph connectivity. In *SIAM Journal on Computing*, volume 4, pages 507–518, 1975.
4. Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 448–456, New York, NY, USA, 1983. ACM.
5. A. Goldberg. Recent developments in maximum flow algorithms. Technical report, 1998.
6. A. Goldberg and S. Rao. Beyond the flow decomposition barrier. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 2–11, 1997.
7. John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *SIAM Journal on Computing* 2(4), pages 225–231, 1973.
8. L. R. Ford Jr. and D. R. Fulkerson. Maximal flow through a network. In *Canadian Journal of Mathematics*, volume 8, pages 399–404, 1956.
9. David R. Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, 1994.
10. David R. Karger and Matthew S. Levine. Random sampling from residual graphs. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 63–66, May 2002.

11. R M Karp, E Upfal, and A Wigderson. Constructing a perfect matching is in random nc. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 22–32, New York, NY, USA, 1985. ACM.
12. A. V. Karzanov. In russian; title translation: On finding maximum flows in networks with special structure and some applications. In *Moscow State University Press*, volume 5, 1973.
13. J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, 2005.
14. Eugene L. Lawler. *Combinatorial Optimization: Networks and Matriods*. Holt, Rinehart and Winston, 1976.
15. Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004.
16. J. C. Picard and H. D. Ratliff. Minimum cuts and related problems. In *Networks*, volume 5, pages 357–370, 1975.
17. Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.