Faster approximation of maximum flow in graphs using electrical flows and laplacian systems.

Krupa Tharahunise Krishnappa
Paderborn University
krupa08@campus.uni-paderborn.de

1st March 2022

Abstract

Computing maximum flow in graphs has contributed immeasurably to technology with an extensive study since early 1930s. Hence research in computing maximum flow in graphs has been an interest of research for long, where computing electrical flows has been immensely beneficial. These electrical flows can be plausibly computed by a system of linear equations (Consider Ax = b). Solving such equations is possible in two ways. First, Gaussian elimination methods and second, the iterative methods. Out of which the iterative method will extensively be used in this report with laplacian matrix. Using laplacian matrices helps in framing novel fast algorithms for graph problems. So, the present report aspires to present a representative fast known algorithm to compute approximately maximum s-t flows. For a graph with n vertices and m edges, the algorithm computes a $(1-\epsilon)$ approximately maximum s-t flow in time $\widetilde{O}\left(mn^{1/3}\epsilon^{-11/3}\right)$. A dual version of the algorithm computes a $(1+\epsilon)$ approximately minimum s-t cut in time $\widetilde{O}\left(m+n^{4/3}\epsilon^{-8/3}\right)$, which was the fastest known algorithm introduced in 2010. Aiming to improve the algorithm in the future, along with which significance of both electrical flows and laplacian systems is analysed. Besides a broad study, associated problems are also deduced, which will aid the horizon of future research inclinations. Further the report will eventually provide possible solutions with a coherence.

***Keywords*— Electrical flows(EF), Laplacian systems(LS), Multiplicative-weights update method, maximum flow problem, minimum cut problem**

1 Introduction

Maximum s-t flow problem is an elementary problem in the optimization theory and a broadly studied problem in the field of operations research [1]. It's dual problem .i.e. the minimum s-t cut problem, also plays a similar role [1]. An extensive research has been done on these two problems since early 1930s, which has been broadly applied in practice. But since operations research is a field where an innumerable methods are studied for accomplishing a desired objective and developing best strategies in a structured well defined mathematical style, hence focusing on the areas of operations research where maximum flow problems have contributed becomes really important [2]. There are a plenty of optimization models in operations research where these problems have contributed like computing methodologies, theory of computation and mathematics of computing [2]. In all these models, the decision maker needs to have a proper measure of goodness to evaluate the best strategy [2]. This measure should be expressed in a certain way, where the problem is to maximize a measure of some aspect of the strategy involved. This maximization should be processed with some satisifcing constraints. In order to maximize in such a scenario, maximum s-t flow problem plays an important role. A brief hierarchy of these models is shown in figure 1. Hence focusing on maximum flow problems in graphs becomes very important.

Along with the maximum flow problems, computing electrical flows has been immensely beneficial. Consider a graph G, for every edge e in G, there is a resistance r_e with a source node s and a sink node t. It satisfies two conditions : Firstly, each edge in the graph is treated as a resistor. Secondly, we imagine the graph as an electrical circuit and connect an imaginary battery to the source node s and the sink node t (figure 2). The induced current

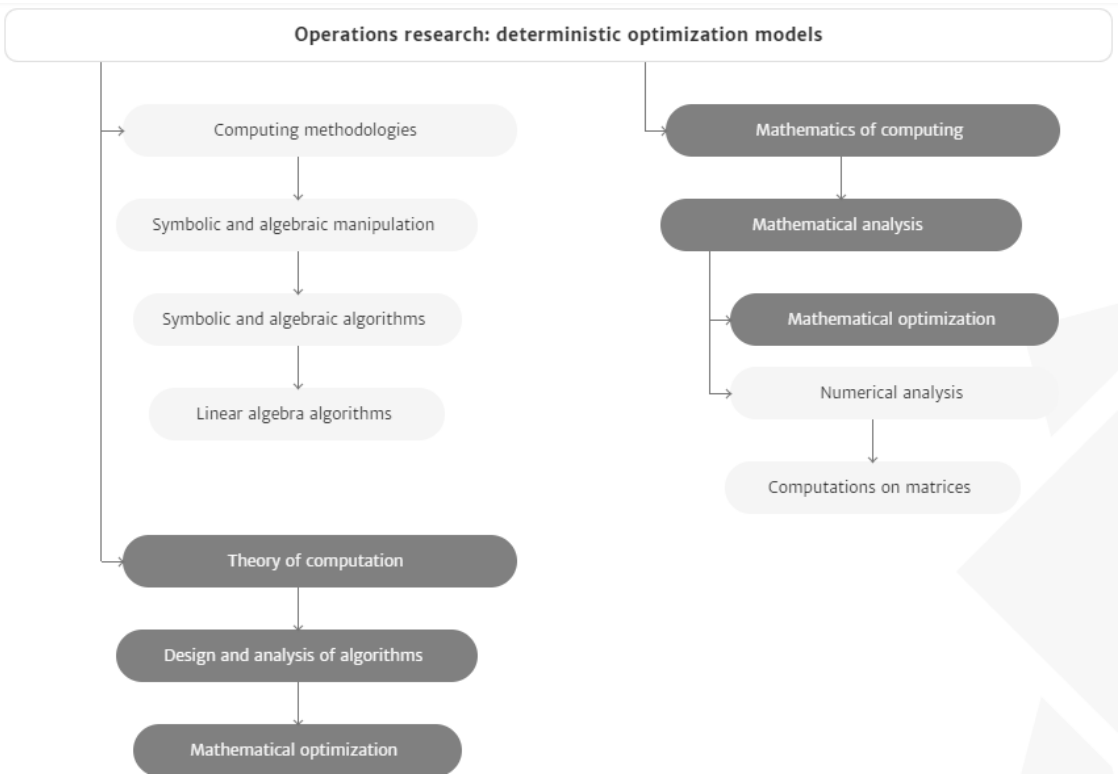


Figure 1: Optimization models in operations research [2]

in the imaginary circuit with flow conservation constraints is called an electrical flow. In other words, electrical flow is the current that settles in the network. Another definition of electrical flow that becomes significant to us in this context is : Electrical flow f_e of maximum flow value f is the unique s-t flow of value f that minimizes the energy $E_r(f) = \Sigma r_e f^2(e)$, which follows according to the "Principle of least energy" .i.e. the second law of thermodynamics in physics.The previous two definitions are equivalent, as it follows from Ohm's law and Kirchoff's law. When computing electrical flows, they can be plausibly computed by a system of linear equations, consider $Ax = b$. A pictorial representation of an implication of a system of linear equations from Ohms law and Kirchoff's law is shown in figure 3.

To compute the electrical flow, Kirchhoff's law and Ohm's law are used, which imply that: Firstly, the sum of the passed currents at every junction in an electrical circuit equals zero, which means that the flow in a networking problem which enters a junction of a network, also exits the network. Secondly the amount of current passed is directly proportional to the difference of voltage between two points. The resulting equations look like a linear system of equations which consists of n number of equations and n number of variables. This can be used in computing flows in graphs problems or to compute alternative paths in a networking problem.

Solving such linear equations is possible in two ways [4]. The first one being, the Gaussian elimination based methods which can be made to run in time proportional to $O(n^{2.3})$ [4]. The second one being, the iterative methods like conjugate gradient methods [4]. The iterative methods can be made to run in time proportional to $O(mn)$ [4]. Where m is the sparsity of matrix A and n is the number of iterations (is very tight in its worst case) [4]. Such worst cases can be improved if A has a unique structure, which could make iterative methods to be used popularly in practice [4]. Here, comes an important class of graphical representation, .i.e. laplacian matrix L, which can be replaced in place of matrix A. A laplacian matrix is used to enumerate the number of spanning trees of a graph G. Since laplacian matrix gives an exact measure of number of spanning trees of a graph G, we can use laplacian matrices in case of iterative methods to solve a system of linear equations .i.e. $Lx = b$, Which enables an accurate and a faster computation of maximum s-t flow and a minimum s-t cut [1]. Hence using laplacian matrices enables researchers to develop novel fast algorithms for graph problems.

Even though there has been a large research going on in case of maximum s-t flow problems, electrical flows and laplacian systems, there is always a scope of improvement. A study [5] which was published in 2004, gave a nearly linear time algorithm for solving a system of linear equations for graph partitioning and graph sparsification [5]. Since then there was a major turn in the research inclinations and numerous fast algorithms were

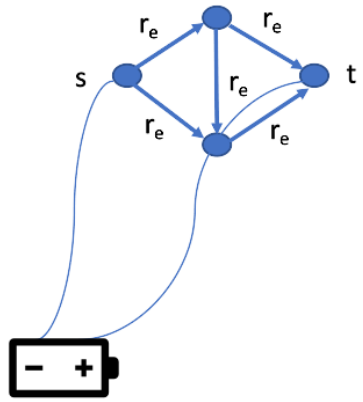


Figure 2: Graph as an electrical circuit with a connected imaginary battery to the source node s and the sink node t [1]

designed for solving a bundle of problems [Like : For example [1], [6], [7], [8], [9], [10]]. In the context of making the laplacian systems comparatively simpler, faster and more optimizable, there has been a large progress in algorithmic enhancements [11]. These enhancements have been adopted in a wide range of applications across multiple domains [11]. Nevertheless, laplacian systems have been extraordinarily successful in case of undirected graph optimization problems and symmetric linear systems, but comparably elusive for directed graph optimization problems and asymmetric linear systems [11]. Specifically, strategies to solve laplacian systems are inherently based on a several properties of undirected graphs. Hence, the present report aspires to review the algorithms which were unveiled in [1]. Where novel fastest algorithms to compute maximum s-t flow and minimum s-t cuts in undirected graphs were developed. A simplified version of the approximate maximum-flow algorithm that has running time $\widetilde{O}\left(m^{3/2}\epsilon^{-5/2}\right)$ was developed. Then, an improvement to the running time of the algorithm to $\widetilde{O}\left(m^{4/3}\epsilon^{-3}\right)$ was made; then this was combined with graph smoothing and techniques of sparsification to compute approximately maximum s - t flows in time $\widetilde{O}\left(\mathrm{mn}^{1/3}\epsilon^{-11/3}\right)$ and to approximate the value of such flows in time $\widetilde{O}\left(m+n^{4/3}\epsilon^{-8/3}\right)$. They present a variant of their algorithm that computes approximately minimum $s-t$ cuts in time $\widetilde{O}\left(m+n^{4/3}\epsilon^{-8/3}\right)$.

Since the present report focuses on a broad study, a focus on directed case becomes essentially important. Until 2016, laplacian systems were considered as unstructured matrices in the directed case. But in two of the important and recent studies [12], [13], the gap in the research direction was closed. The running time was reduced by a factor of polylogarithmic factor in the study of [12]. The reduction was then combined with an algorithm to solve eulerian laplacian systems in time proportional to $\widetilde{O}\left(m^{3/4}n+mn^{2/3}\right)$ [12]. After which, the time complexity was improved to run in almost linear time $\widetilde{O}\left(m+n2^{O(\sqrt{\log n \log\log n})}\right)$ by the study in [13]. However, the running time was again reduced to approximately $\tilde{O}(m)$ time in the study of [11]. Here they show that the eulerian laplacian systems consists of sparse-approximate LU factorizations. Considering such aspects in the directed case, the last section of the paper named as "Outlook", will channelize on what can be done in the future.

Since maximum s-t flow problems are a major area of concern, reason being that they are fundamentally important in operations research, further research is required. As a consequence, the present report gives a quick overview on Electrical flows and laplacian systems. Keeping Electrical flows and laplacian systems as a starting point, the present report discusses how things can be made dynamic for the future research using foundations made previously. Forthcoming challenges can be resolved by integrating maximum s-t flow, minimum s-t cut problems with high level architectures of mathematics and operations research. The present report seeks the heed of the individuals working and studying in the field of graph theory, data-structures/algorithms and in optimizations theory. The main motivation behind this report is to investigate the maximum flow problems broadly. The aspiration is to provide useful context and implications on how things could be improved. Specifically in terms of the graphs and operations research, considering all kinds of interdisciplinary aspects. Additionally, the report provides advantages, disadvantages, advancements and the questions left unanswered in some studies that could be carried out in the near future to address complex forthcoming problems.

Structure of the report: Section 2 is mainly focused on the prerequisites required to understand the report. It initially explains important concepts. Then about laplacian systems and electrical flows. Section 3 explains

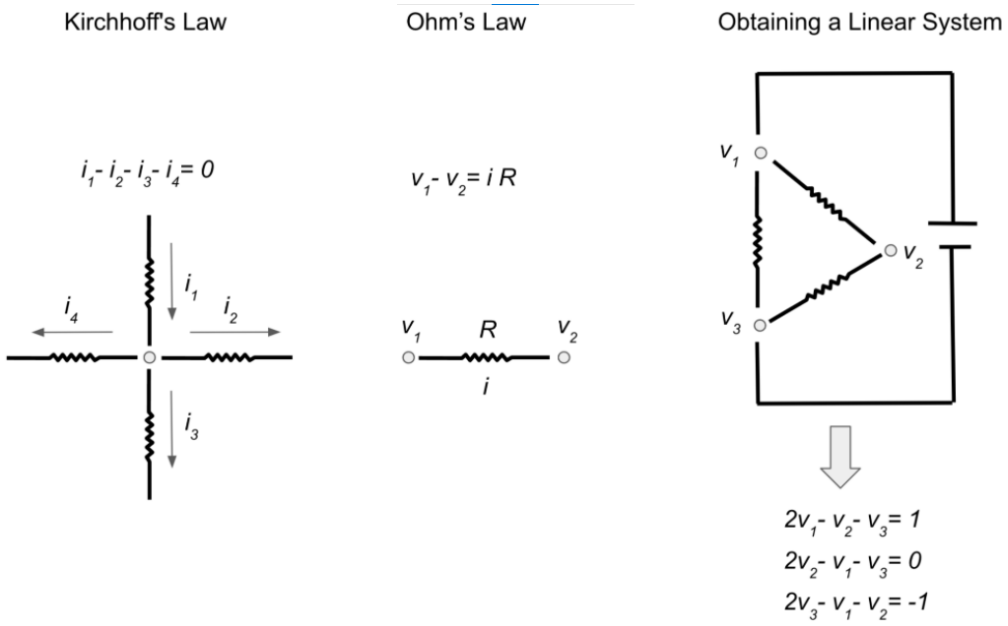


Figure 3: Implication of a system of linear equations from Ohms law and Kirchoff's law [3]

one of the significant, yet simple $(1-\epsilon)$ maximum s-t flow approximate algorithm with its improved version, which was unveiled in [1] to get deeper insights. Then a dual algorithm for minimum s-t cut [1]. Section 4 provides a brief summary. Finally concluding the report in section 5. Section 6 explains about findings from all the provided resources and is hence named as: Outlook. Where the present report provides its best efforts to study a possible number of algorithms that use laplacian systems, electrical flows and other algorithmic techniques. Then, the present report introduces an analysis and an outlook based on the key advantages provided by the laplacian systems, electrical flows and provides leads to integrate other algorithmic techniques in the near future to solve the forth coming challenges.

2 Pre-requisites

This section gives a detailed overview of the concepts required to understand the report. Initially an overview of all the concepts is explained. Further, about the electrical flows and laplacian systems with a taxonomic classification based on their advantages in various algorithms.

2.1 Concepts

For the algorithms explained in section 3, Assume $G = (V, E)$ to be an undirected graph which has n vertices and m edges. The graph consists of a source node s and a sink node t. Each edge e has a capacity $u_e \in Z^+$ .i.e. a non zero integer. Assume $U := \max_e u_e / \min_e u_e$ to be a ratio of largest to the smallest capacities. Also, let $E^+(v)$ indicate the flow of edges towards the vertex v and $E^-(v)$ indicate the flow of edges way from the vertex v. Here we are dealing with undirected graphs and the flow of edges can go in any direction. Cuts and flows in a graph are defined in the following ways : ([1])

A **cut** in a graph G is a partition of the vertices into two disjoint sets such that the source vertex s belongs to one set and the sink vertex t belongs to another. A graphical representation of a cut is depicted in figure 4.([1])

A **flow** in a graph G is a flow from the source vertex s to the sink vertex t and it is a function $f : E \to R$ that follows flow conservation constraints. Flow conservation constraints are as illustrated below : ([1])

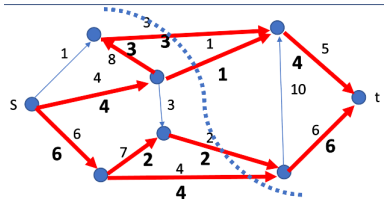


Figure 4: Minimum s-t cut [1]

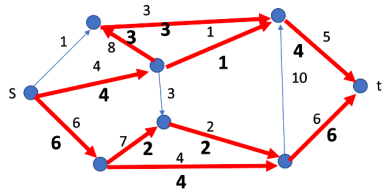


Figure 5: Maximum s-t flow [1]

$$\sum_{e \in E^-(v)} f(e) - \sum_{e \in E^+(v)} f(e) = 0 \quad \text{for all } v \in V \backslash \{s, t\}$$

Where $|f|$ stands the net flow out of the source vertex, $|f| := \sum_{e \in E^-(s)} f(e) - \sum_{e \in E^+(s)} f(e)$. Flow conservation constraints implies that the net flow coming out of s is equal to the net flow going into t, hence $|f|$ can be elucidated as the amount of flow sent from s to t. For instance in figure 5, the value of $|f|$ is 10. ([1])

Maximum flows and minimum cuts : A maximum flow problem is defined as a problem to find a feasible s - t flow of maximum value. Where a feasible s-t flow is an assignment of flows to edges in a graph such that it satisfies two constraints , namely : Capacity constraints and flow conservation constraints. First, the capacity constraint is when the flow on every edge does not exceed the capacity of the edge. ($|f(e)| \leq u_e$ for each edge e). Second, the flow conservation constraints follows from the equation depicted above .i.e. flow into a vertex v in a graph G is equal to the flow out of the vertex v, for all vertices except the source vertex s and the sink vertex t. ([1])
$f(e) = f(e'), (\forall v \; ! = s, t)$

The objective of the maximum flow problem is to maximize the feasible flow that is the net flow out of source vertex s in equal to the net flow into the sink vertex t. In section 3, the maximum flow in G (with the respective capacities) is represented by f^* and its value is denoted by $F^* := |f^*|$. f is known as $(1 - \epsilon)$ approximate maximum $s - t$ flow if the value of the feasible $s - t$ flow is not less than $(1 - \epsilon)F^*$. An example in this case can be considered from figure 5. ([1]). A minimum $s - t$ cut problem is defined as a problem to find an $s - t$ cut of minimum capacity, for example consider figure 4. The definition of an $s-t$ cut follows from the definition depicted above, and a capacity of a cut is the sum of the capacities of the edges leaving cut C. For example: Consider figure 4, where the flow leaving the cut (leaving the green circle) is 10. There is a connection between maximum s-t flow and the minimum s-t cut and the fact is depicted by a max-flow-min-cut theorem([14], [15]). The theorem states that the value of the minimum s-t cut is F^*, that is same as the value of the maximum s-t flow. A minimum s-t cut in a graph can be computed, if maximum flow is given but vice versa is not possible. In other words, there is no particular known strategy to obtain maximum s-t flow from minimum s-t cut (figure 6 recovers minimum cut from the maximum s-t flow in figure 5). Definitions of laplacian systems and electrical flows follows from the introduction section. In order to compute electrical flows with a system of linear equations, an incidence matrix B is required. An incidence matrix is a $n * m$ matrix in which rows are indexed by vertices and columns are indexed by edges. The incoming edge is represented by $e \in E^-(v)$ and the outgoing edge is represented by $e \in E^+(v)$. The matrix B gets a value of 1 in case of an incoming edge $(E^-(v))$,-1 in case of an outgoing edge $(E^+(v))$ and 0 otherwise. Figure 7 gives a very good example of an incidence matrix. Following is a mathematical representation of an incidence matrix B.

$$\boldsymbol{B}_{v,e} = \begin{cases} 1 & \text{if } e \in E^-(v) \\ -1 & \text{if } e \in E^+(v) \\ 0 & \text{otherwise} \end{cases}$$

An example of an incidence matrix is presented in figure 7. Where for the edge 1, vertex a gets a value of $+1$

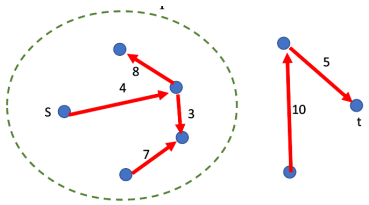
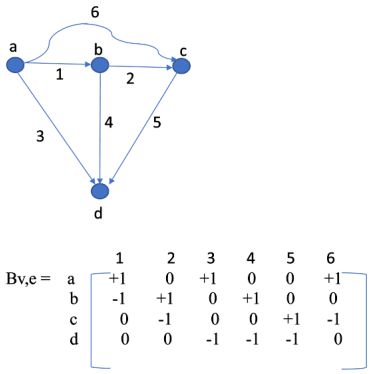


Figure 6: Recovering minimum cut from maximum flow. [1]



Bv,e =

	1	2	3	4	5	6
a	+1	0	+1	0	0	+1
b	-1	+1	0	+1	0	0
c	0	-1	0	0	+1	-1
d	0	0	-1	-1	-1	0

Figure 7: Incidence matrix. [1]

since the edge is outgoing from the vertex a, vertex b gets a value of -1 since the edge is incoming into the vertex b. An incidence matrix is defined because a laplacian matrix of a graph is defined in terms of an incidence matrix. A weighted laplacian matrix L, with respect to the resistances r to be the $n * n$ matrix and is represented by :

$$\boldsymbol{L} := \boldsymbol{BCB}^T$$

Where C is a diagonal matrix of the order $m * m$, with its elements represented by :

$$\boldsymbol{C}_{e,e} = c_e = 1/r_e$$

.

Formally from [1], the entries of a laplacian matrix is as follows :

$$\boldsymbol{L}_{u,v} = \begin{cases} \sum_{e \in E^+(u) \cup E^-(u)} c_e & \text{if } u = v \\ -c_e & \text{if } e = (u,v) \text{ is an edge of } G, \text{ and } ([1]) \\ 0 & \text{otherwise.} \end{cases}$$

Assume $\boldsymbol{R} = \boldsymbol{C}^{-1}$ to be the diagonal matrix with $\boldsymbol{R}_{e,e} = r_e$. The energy of a maximum flow $\boldsymbol{f}$ in a given graph is given by ([1])

$$\mathcal{E}_r(f) := \sum_e r_e \boldsymbol{f}(e)^2 = \boldsymbol{f}^T \boldsymbol{R} \boldsymbol{f} = \left\| \boldsymbol{R}^{1/2} \boldsymbol{f} \right\|^2$$

([1])

3 Novel fast algorithms for maximum $s-t$ flow and minimum $s-t$ cut.

This section explains the algorithms from the study of [1] for maximum flow and minimum cut problems. These problems are extremely important because they are the most basic optimization problems used in most of the algorithms and domains, they have been efficiently used in graph theory since ages and lately used in the field of automation (neural networking algorithms). Since these problems are very important lets dive deeper into the algorithms. In these problems, the basic problem is defined as a (1-ϵ) approximate maximum flow algorithm in un-directed graphs. For this basic problem, the previously known asymptotically fastest algorithm was in 1975 by Even and Tarjan [16] in time $O\left(n^{3/2}\right)$ [1]. Also there has been previous researches on this problem like for example in 1998, Goldberg rao uses augmenting paths framework [17]. This framework involves iterative finding of $s-t$ paths in a residual graph, that is extensively combinatorial and the flow is built path by path [17]. In this

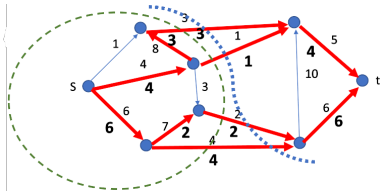


Figure 8: Capacity of a minimum cut. [1]

iterative framework, edges of all paths were flipped until there was no possibility to find an $s-t$ path [17]. Then the maximum flow solution for the corresponding graph is recovered in the final stage [17]. After this attempt it became unclear how to improve on the strategies. Hence we will go through the fastest known algorithm introduced in the study of [1].

Initially a simple (1-ϵ) approximate algorithm that runs in time $\widetilde{O}\left(m^{3/2}\epsilon^{-5/2}\right)$. Then an improved algorithm that runs in time $\widetilde{O}\left(mn^{1/3}\epsilon^{-11/3}\right)$, followed by a dual algorithm for minimum $s-t$ cuts that runs in time $\widetilde{O}\left(m+n^{4/3}\epsilon^{-8/3}\right)$.

3.1 A Simple $\widetilde{O}\left(m^{3/2}\epsilon^{-5/2}\right)$-Time Flow Algorithm

The main key of this algorithm is examining the graph flow problem by solving a system of linear equations. A strategy called multiplicative weights method is also employed in this algorithm. The multiplicative weights method solves a problem in a crude way, by calling it iteratively and converting it into an algorithm which results in a very good approximation of the maximum flow in graph G. In order to define the feasibility of an $s-t$ flow f, there is a need to define a ratio between the flow and the capacity of an edge ,which is defined as congestion and expressed as follows : ([1]) $\operatorname{cong}_f(e) := \frac{|f(e)|}{u_e}$.

Algorithm 1: Multiplicative-weights-update routine [1]

Input: A graph $G=(V,E)$ with capacities $\{u_e\}_e$, a target flow value F, and an (ϵ,ρ)-oracle O
Output: : Either a flow $\bar{f}$, or "fail" indicating that $F > F^*$;
Initialize $w_e^0 \leftarrow 1$ for all edges e, and $N \leftarrow \frac{2\rho \ln m}{\epsilon^2}$
for $i := 1,\ldots,N$ do
  Query O with edge weights given by w^{i-1} and target flow value F
  if O returns "fail" then return "fail"
  else
    Let f^i be the returned flow
    $w_e^i \leftarrow w_e^{i-1}\left(1+\frac{\epsilon}{\rho}\operatorname{cong}_{f^i}(e)\right)$ for each $e \in E$
  end
end
return $\bar{f} \leftarrow \frac{(1-\epsilon)^2}{(1+\epsilon)N}\left(\sum_i f^i\right)$

Basically in multiplicative weights update method, the weights are updated iteratively in a multiplicative manner. The multiplicative method is proportional to the factor ϵ, which is an accuracy parameter. This method takes in the algorithm as an oracle, assume A as an oracle, .i.e. an algorithm. The multiplicative weights updating strategy maintains all the weights w_e^i as 1 in its initial iteration. Then in every corresponding iteration, weights of each edge are updated based on the flow f^i. At the end, the strategy returns average of all the flows that were computed along the way, then F^* gets the average value of all the f^i's. After completing $\rho\varepsilon^{-2}$ iterations, an (1-ϵ) approximation of maximum-flow problems is obtained. Where ρ (is the width of the oracle) .i.e.the largest overflow encountered .In every iteration the weight of every edge e in every iteration i, is updated by the equation $w_e^i \leftarrow w_e^{i-1}\left(1+\frac{\epsilon}{\rho}\operatorname{cong}_{f^i}(e)\right)$ for each $e \in E$. Where ρ is the largest edge overflow that the edge incurs in flow f^i.In the equation used to update the weights, the numerator is being normalised by ρ .i.e. the largest edge overflow that the edge incurs. Normalizing here is important because there should be a sense of assurance that the

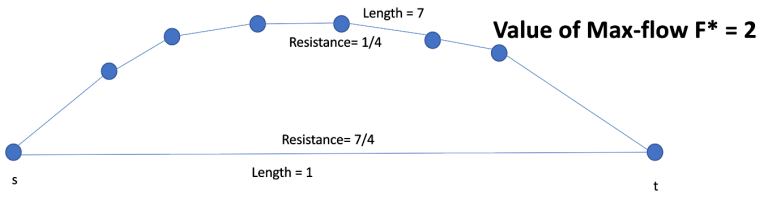


Figure 9: Electrical flows to approximate maximum flow. [1]

multiplicative weights update is somewhere between 1 and $1+\epsilon$. In this way, an assurance can easily be obtained about the evolution of the weights moving up, easily. Hence proving the algorithm appropriately. Now lets dive into the dynamics of the multiplicative weights updating strategy. Whenever an edge suffers a large overflow, its weight grows rapidly at an exponential rate. Which basically means that: if the average overflow is small, w_e^i grows slowly. Then we have to note an important point that no edge normally suffers large overflow too often. The algorithm of the multiplicative weights update routine is shown step by step with a pseudo code in Algorithm 1. According to [1], following definition and its conditions are used and followed in the algorithms explained in this report from [1].

Definition : $((\epsilon, \rho)$ oracle). For $\epsilon > 0$ and $\rho > 0$, an (ϵ, ρ) oracle is an algorithm that, given a real number $F > 0$ and a vector w of edge weights with $w_e \geq 1$ for all e, returns an s - t flow f such that:([1]) 1. If $F \leq F^*$, then it outputs an s-t flow f satisfying:([1])
(i) $|f| = F$; ([1])
(ii) $\sum_e w_e$ cong $_f(e)$.([1]) $\leq (1+\epsilon)|\boldsymbol{w}|_1$, where $|\boldsymbol{w}|_1 := \sum_e w_e$; ([1])
(iii) $\max_e$ cong $_f(e) \leq \rho$. ([1])
2. If $F > F^*$, then it either outputs a flow f satisfying conditions (i), (ii), (iii) or outputs "fail".([1])

Now that we have seen multiplicative weights update method in brief lets see how are electrical flows used in this context. As we have seen in the introduction that electrical flows can be computed by using a system of linear equations with an efficient way of using laplacian matrix as a constrained matrix and can be solved in linear time. Here we will see how are electrical flows used to approximate maximum flow. Here, capacity is initialized to 1 : $\mathrm{u(e)} = 1$, for every edge e. Maximum flow $\mathrm{F}*$ is known, which means that it can be predetermined by other algorithms like binary search. After knowing the capacity and maximum flow, a procedure can be applied, which has three steps. In the first step, $\forall$Edgee, all the resistances $\mathrm{r(e)}$ are set to 1. Then in the second step, the corresponding electrical $s-t$ flow $\mathrm{f_e}$ of maximum value $\mathrm{F}*$ is computed. Then in the third step, the computed flow is considered as a solution. This computation is not feasible enough because electrical flow favours a path with lower resistance. For example: Consider the graph example presented in figure 9. Here, the electrical flow favours lower path since the resistance is lesser. Such a scenario does not become feasible with maximum flow and we start getting deviated from maximum flow problems. Since it is not feasible, wa modification is required to the three step procedure described above. There is a simple modification, where in the third step, there is a repeat of steps: in increasing resistances on overflowing edges and repeat computing corresponding electrical flows until convergence. Finally, obtain the solution by taking average of all the electrical flows to get the final flow. This approach becomes feasible. Note that we are dealing with un-directed graphs, with capacities assigned to 1, the graph is sparse .i.e. number of edges m is O(n) and the maximum flow $\mathrm{F}*$ is known. ([1])

Consider every edge e in the graph in figure 5, with the resistances r(e), consider their corresponding electrical $s-t$ flow f_e of maximum value F^*. f_e is not expected to obey all capacity constraints .i.e. $|f(e)| >> 1$ for some edge e. Also note that the capacity constraints are obeyed on the weighted average, hence weights are directly proportional to the resistances .i.e. :([1])

Weights $w_e's \propto {r_e}'s$.

At this point in the algorithm, the resistances r_e are replaced with weights w_e and the multiplicative weights strategy gets utilized here. By this replacement the run time of the algorithm decreases, In other words the algorithm runs faster. Now lets recall the algorithm described above where the resistances $r(e)$ were assigned to 1 in the first step. Then in the second step where the corresponding electrical flow for every edge was computed and in the third step the iterations were repeated until convergence. The same algorithm gets modified when the multiplicative weights update method gets integrated. Even here a modification to the third step is required, where the number of iterations $N = O(\left(\rho\varepsilon^{-2}\right))$ are executed and in these iterations : resistances are modified in terms of the weights w_e using the equation : ([1]) $w_e^i \leftarrow w_e^{i-1}\left(1+\frac{\epsilon}{\rho}\operatorname{con} g_{f^i}(e)\right)$

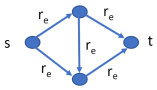


Figure 10: Simple graph [1]

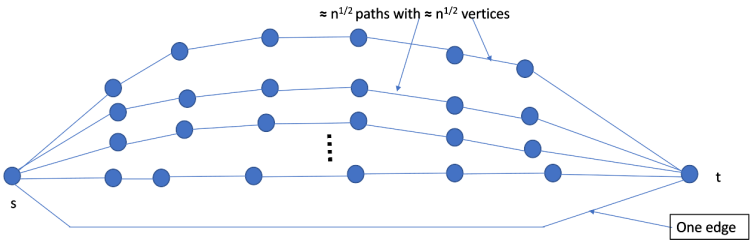


Figure 11: Simple graph [1]

([1]) Then compute the corresponding electrical flow of the respective edge e. After completing $O(\left(\rho\varepsilon^{-2}\right))$ iterations, the final solution is obtained by taking the average of all the electrical flows to get the final flow. Here the resistances $r(e)$ evolve as w_e and the convergence condition is changed to $N = O(\left(\rho\varepsilon^{-2}\right))$ number of iterations. This algorithm gives $(1-\epsilon)$ approximate maximum flow in a bound of $\mathrm{O}\left(\rho\varepsilon^{-2}\right) \cdot \mathrm{O(n)} = \mathrm{O}\left(\mathrm{n}\rho\varepsilon^{-2}\right)$. In order to determine the run time of this algorithm we also need to look into the worst case bound on the overflow ρ and for this we just need to see that what is the largest overflow ρ, an electrical flow can suffer ? Here we can make a simple claim that : If all $\mathbf{r}_\mathrm{e} = \mathbf{1}$, then in the electrical flow $\mathrm{f_e}$, no edge flows more than $\mathrm{m}^{1/2} = \mathrm{O}\left(\mathrm{n}^{1/2}\right)$. Which means that the uniformity is maintained in the resistances. But there maybe situations when the algorithm maybe used and there is no uniformity and we fix this by restricting the non-uniformity of resistances. ρ then gets bounded by $\mathrm{O}\left(\mathrm{n}^{1/2}\boldsymbol{\varepsilon}^{-1}\right)$. This gives us an $(1-\varepsilon)$ approx max-flow algorithm that runs in $\mathrm{O}\left(\mathrm{m}^{3/2}\varepsilon^{-5/2}\right)$ time. ([1])

3.2 An improved algorithm for approximate maximum Flow in time $\widetilde{O}\left(mn^{1/3}\epsilon^{-11/3}\right)$

Algorithm 2: An improved $(1-O(\epsilon))$-approximation algorithm for the maximum flow problem [1]

Input: A graph G = (V, E) with capacities $\{u_e\}_e$, and a target flow value F
Output: Either a flow $\bar{f}$, or "fail" indicating that $F > F^*$;
Initialize $w_e^0 \leftarrow 1$ for all edges e, $H \leftarrow \emptyset$, $\rho \leftarrow \frac{8m^{1/3}\ln^{1/3} m}{\epsilon}$, and $N \leftarrow \frac{2\rho \ln m}{\epsilon^2}$
for $i := 1, \ldots, N$ do
  Query O' with edge weights given by $\boldsymbol{w}^{i-1}$, target flow value F, and forbidden edge set H
  if O returns "fail" then return "fail"
  else
    Let f^i be the returned answer
    Replace H with the returned (augmented) set of forbidden edges
    $w_e^i \leftarrow w_e^{i-1}\left(1 + \frac{\epsilon}{\rho}\,\mathrm{cong}_{f^i}(e)\right)$ for each $e \in E$
  end
end
return $\bar{f} \leftarrow \frac{(1-\epsilon)^2}{(1+\epsilon)N}\left(\sum_i f^i\right)$

The full algorithm is defined in the previous section, but there is still a possibility of improvement. According to the authors in [1], the running time of the algorithm was improved upto $\mathrm{O}\left(\mathrm{n}^{4/3}\varepsilon^{-3}\right)$. In the current algorithm the running time is dominated by $\rho = \mathrm{O}\left(\mathrm{n}^{1/2}\right)$ electrical flow computations. Here, the bound on the worst case overflow ρ that an electrical flow can suffer, can still be improved. Consider the graph in figure 11, in order to compute the value of maximum flow F^*, one unit of flow is sent on each path. It becomes clear that the lower edge gets half of the flow and hence this scenario is not optimal enough. So, the graph needs some modification.

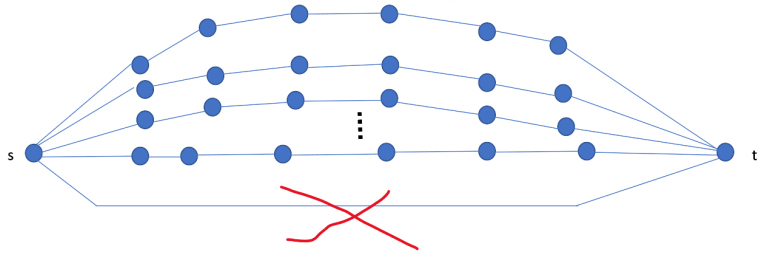


Figure 12: A graph on which the electrical flow sends approximately $\sqrt{n}$ units of flow across an edge when sending the maximum flow F^* from s to t. [1]

Hence we remove the bottom edge and by removing this bottom edge we get few observations : if we cut this really bad edge (figure 12), the maximum flow does not change too much and the electrical flow is much better halved. This observation can be changed into an algorithmic idea and for this we make our algorithm self enforce smaller overflow $\rho' \ll \rho$. The algorithm is then modified, the modification follows this way : An edge from the respective graph is removed if it overflows more than ρ' and then recompute the flow afterwards. As a result there is a need to note that if the algorithm always successfully terminates, its worst case overflow is at-most ρ'. But a simple question arises that how should one choose ρ' ? Here there should be an assurance that ρ' should be as small as possible. But if it becomes too small the edge removal might become too aggressive. In other words too many edges are cut. An interesting value for ρ' is $n^{1/3}$, and the key reason for this value is removal of an edge that flows a lot, increases the energy significantly, but decreases the maximum flow only slightly. This choice of ρ' gives a $O\left(n^{4/3}\varepsilon^{-3}\right)$ time algorithm. The theorem 4.5 in [1], can be applied to the $\widetilde{O}\left(\mathrm{m}^{4/3}\epsilon^{-3}\right)$ algorithm, to obtain the required running-time $\widetilde{O}\left(mn^{1/3}\epsilon^{-11/3}\right)$. A brief improved algorithm with the pseudo code from [1] is presented in Algorithm 2. ([1])

3.3 A dual Algorithm to find approximately minimum $s-t$ cut in time $\widetilde{O}\left(m+n^{4/3}\epsilon^{-8/3}\right)$

This section explains the dual algorithm to find approximately minimum $s-t$ cut in brief from [1]. The dual algorithm involves the computation of maximum flow first. For the purpose of approximation, minimum cut algorithm always assumes that the number of edges are m = O(n) (sparsity of the graph). From the maximum flow - minimum cut theorem one can compute minimum $s-t$ cut from the given maximum optimal flow. But electrical flows have to be used in this context in order to run the algorithm efficiently. For this, the cuts have to be made small enough. To make the cuts small, a two step procedure has to be followed. First, Increase the resistances of edges (based on how much they flow). Second, recompute the electrical flow. In this way one can show appropriate update rule for resistances. $(1+\varepsilon)$ approximate, minimum s-t cut of a given graph is finally exposed after $\mathrm{O}\left(\mathrm{m}^{1/3}\varepsilon^{-8/3}\right)$ iterations. Core of the analysis is essentially the same as in the maximum flow algorithm. A pictorial representation of applying electrical flows to minimum $s-t$ cuts with an imaginary battery is represented in figure 13. The dual algorithm for computing a minimum $s-t$ cut is represented in the algorithm 3 from the study of [1].

3.4 Analysis and further improvements on the above algorithms

The algorithms provided in this report use electrical flows, laplacian systems and multiplicative weights update method to run faster. These techniques have served a great advantage in computing maximum flows and minimum cuts efficiently. But the procedures in these algorithms could be simplified, since they follow such lengthy procedures, for example : since weights are replaced with resistances, computation of resistances in between the algorithm could be reduced to decrease the complexity. ([1])

When studying these algorithms, a natural question arises : Can this approach lead to a nearly linear time max-flow algorithm ?. Well, there are some possible ways, one idea is to make one of the variant of the algorithm run in linear time. Other approaches like modifying update rules, using differential calculus, using better heuristics and better thresholding of overflows can be very useful in improving these algorithms. Where modifying the update rules may lead to a better run time complexity, using differential calculus can make the algorithms more efficient, using better heuristics will help in improving most of the performance metrics and better thresholding of the overflows will also contribute to better running time of the algorithms that will lead us to apply the algorithms more practically in the real time scenarios (Which indicates better robustness). Another research direction for the

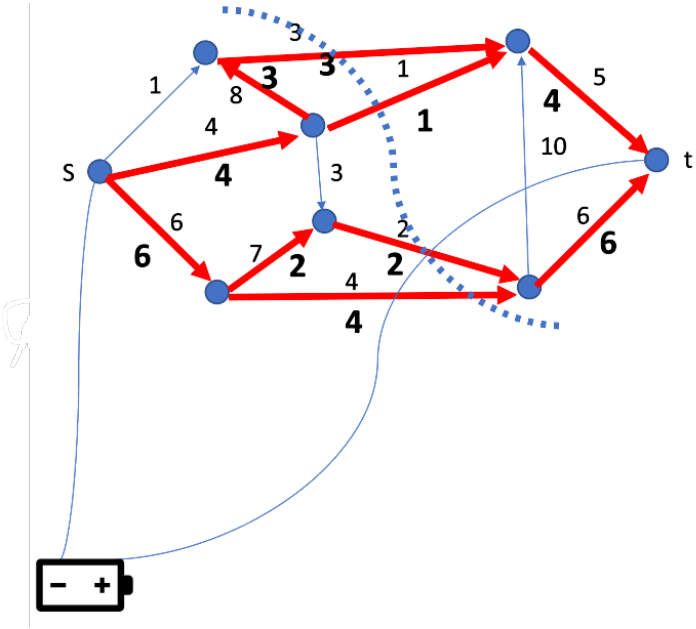


Figure 13: Applying electrical flows to minimum s-t cuts with an imaginary battery. [1]

Algorithm 3: A dual algorithm for finding an s-t cut [1]

Input: A graph G = (V, E) with capacities $\{u_e\}_e$, and a target flow value F
Output: A cut $(S, V \backslash S)$
Initialize $w_e^0 \leftarrow 1$ for all edges e, $\rho \leftarrow 3m^{1/3}\epsilon^{-2/3}$, $N \leftarrow 5\epsilon^{-8/3}m^{1/3}\ln m$, and $\delta \leftarrow \epsilon^2$
for $i := 1, \ldots, N$ do
  Find an approximate electrical flow $\widetilde{f}^{i-1}$ and potentials $\vec{\phi}$ using Theorem 2.3 on G with resistances $r_e^{i-1} = \frac{w_e^{i-1}}{u^2}$, target flow value F, and parameter δ.
  $\mu^{i-1} \leftarrow \sum_e w_e^{i-1}$
  $w_e^i \leftarrow w_e^{i-1} + \frac{\epsilon}{\rho}\,\mathrm{cong}_{\tilde{f}^{i-1}}(e) w_e^{i-1} + \frac{\epsilon^2}{m\rho}\mu^{i-1}$ for each $e \in E$
  Scale and translate $\tilde{\phi}$ so that $\widetilde{\phi}_s = 1$ and $\widetilde{\phi}_t = 0$
  Let $S_x = \{v \in V \mid \phi_v > x\}$
  Set S to be the set S_x that minimizes $(S_x, V \backslash S_x)$
  If the capacity of $(S_x, V \backslash S_x)$ is less than $F/(1 - 7\epsilon)$, return $(S_x, V \backslash S_x)$
end
return **"fail"** "

study of [1] would be to take the same study forward and perform robustness testing and some validation testing against some optimal strategies. Validation testing always helps us to come to a better conclusion. Yet another direction could be to apply trigonometric and sigmoid functions in these algorithms to make them perform better. ([1])

4 Summary

The whole report gives a brief overview of the algorithms used to compute the optimal maximum flows and minimum cuts in un-directed graphs using electrical flows and laplacian systems. The deepest connection between maximum flows and minimum cuts using the maximum-flow-minimum-cut theorem plays a very significant role in designing algorithms for the computation. Where the simple algorithm runs in time approximate to $\widetilde{O}\left(m^{3/2}\epsilon^{-5/2}\right)$, improved algorithm runs in time approximate to $\widetilde{O}\left(mn^{1/3}\epsilon^{-11/3}\right)$ and the dual algorithm runs in time approximate to $\widetilde{O}\left(m + n^{4/3}\epsilon^{-8/3}\right)$. In all these algorithms, the multiplicative weights update method have been very useful. Even though the algorithms run in approximately linear time, further improvements are possible.

So, in order to contribute to the future research inclinations, the present report gives a brief outlook at the end of the report.

5 Conclusion

The present report explains the optimal algorithms for computing maximum flows and minimum cuts in un-directed graphs. Even though these algorithms run in approximately linear time, further improvements are always possible. Since the algorithms presented in this report from the previous studies bridges a connection between linear algebra and combinatorial algorithms, we can think of using this connection in other areas of graph theory/operations research like to advance search strategies, path computations, graph homomorphism, graph partitioning etc. Another direction for future can be to use the main concepts of this report like : electrical flows, laplacian systems and multiplicative weights update methods in other areas of reasearch. Well, a detailed outlook is presented at the end of this report which will aid the directions of future research in networking, graph theory and flow problems. Finally concluding, as said by Leonardo da Vinci "Learn how to see. Everything connects to everything else", hence all the studies were analysed, linked and a coherence was provided. It is expected that the depicted analysis will aid the future research in solving forthcoming challenges.

6 Outlook

This section provides a brief summary of the other relevant literature reviewed apart from the study [1] that are used for discussions in detail for the previous sections. Two important yet dominant factors which affects the modern approaches in network flow problems are discussed. First being: the factors that affect the modelling of algorithms and second being the performance metrics with some of their validation techniques. Then, As discussed in previous sections, electrical flows and laplacian systems might have overcome the challenges in improving the algorithms, but they have their own challenges. Hence few potentially identified challenges with their corresponding future directions are explained in this section. Further some other areas where the bridging a connection between linear algebra and combinatorial algorithms can become significant in near future. ([1])

First lets the two factors : Modelling of algorithms and the aspects of performance metrics. Consider the modelling of algorithms in network flow problems. Initially, let's consider the modelling of algorithms. The directions or ideas discussed in section 3.4 are reasonable enough, but can we make these ideas to work for directed graphs ? Here we need to note that: to get an exact algorithm for directed case , we just need a $(1 - \epsilon)$ - approximation for undirected graphs with logarithmic dependence of $1/\epsilon$. Other aspects of modelling the algorithms are : Optimizing the available algorithms for efficient memory consumption, better optimization of the data structures, better optimization of the arrays used in the algorithms etc. Yet, after having all these techniques, the fields of linear algebra, discrete mathematics, mathematics and operations research are very vast enough to extract the required methods to improve our algorithms. For example : The logical part in discrete mathematics has a reasonable number of inference rules, logical implications etc, that can be utilized into the existing algorithms. For example : the improved algorithm explained in this report from [1] can be integrated with one of the inference rules to improve the performance. In this way : there are many more advancements possible. ([1])

Now that we have seen about the factors that affect the modelling of algorithms, let's discuss about improving the performance metrics. There are a huge variety of performance metrics that algorithm can be bench marked upon. For example the networking problems always have a wide variety of performance parameters like: bandwidth, throughput, latency, packet loss, Jitter etc. (in case of real time problems). But when it comes to designing algorithms, the performance metrics differ a lot like : running time, compile time, space complexity, accuracy etc. Overall in order to come out with a better performance, a good testing and a better benchmarking is required. For example : When the study of [1] is benchmarked against the most recent algorithms after the publication of the research work, any researcher can ensure that there can be great improvements possible like: benchmarking the study [1] against [18] because James B Orlin came up with different aspects to improve the performance of computation of maximum flow problems. Hence benchmarking an algorithm against other algorithms becomes very essential to bridge the gap in the network flow problems. ([1])

Second lets discuss the challenges and some possible solutions for electrical flows and laplacian systems in networking problems. One of the most important challenges being that: according to electrical/electronics domain, resistance hinders the electric current settled in a circuit [19]. But in the algorithms represented from the study of [1], the resistances are replaced by weights (weights proportional to resistances as discussed above), in order to make the algorithm run faster. Hence, no matter which algorithms are modelled in the future, this fact has to

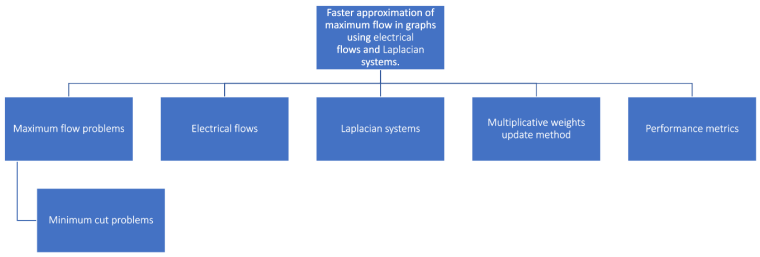


Figure 14: A thematic analysis of the research topic [1]

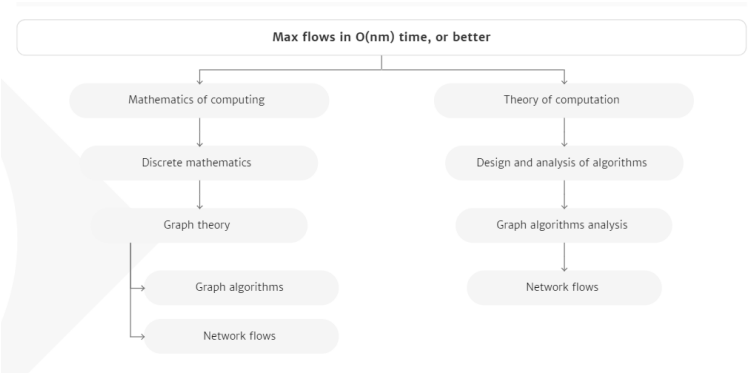


Figure 15: Max flows in O(nm) time, or better [18]

be noted before the beginning of the research. Computation of electrical flows in the electrical/electronics domain domain has advanced very much, but all the advancements are not yet adapted in the network flow problems. This fact helps a lot in improving our network flow problems. ([1])

Third lets discuss about how and where can the bridging of linear algebra and combinatorial algorithms be useful. This bridging is helpful in classical methods like : The eigen value connections, spectral graph partitioning, understanding of random walks, improving search algorithms in networking and in improving other areas of network flow problems which are left untraversed. Further, this bridging can also be used in finding alternative paths in a networking problem. In this way every smallest aspect addressed in the research topic of this report can be analysed. Hence to aid in this perspective, the present report provides a thematic analysis that is provided in figure 14 . Yet another analysis of the research topics from the study of [18] is provided in figure 15, which clearly indicates the areas of the topic where further research is required. ([1])

References

[1] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs, 2010.

[2] Katta G Murty. *Operations research: deterministic optimization models*. Prentice-Hall, Inc., 1994.

[3] Google ai blog: Robust routing using electrical flows. https://ai.googleblog.com/2022/02/robust-routing-using-electrical-flows.html. (Accessed on 02/28/2022).

[4] Nisheeth K Vishnoi. Laplacian solvers and their algorithmic applications. *Theoretical Computer Science*, 8(1-2):1–141, 2012.

[5] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 81–90, New York, NY, USA, 2004. Association for Computing Machinery.

[6] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

[7] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016.

[8] Aleksander Madry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 2019–2036. SIAM, 2014.

[9] Michael B Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in õ (m 10/7 log w) time*. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 752–771. SIAM, 2017.

[10] Jonathan A Kelner, Gary L Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1–18, 2012.

[11] Michael B. Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B. Rao, and Aaron Sidford. Solving directed laplacian systems in nearly-linear time through sparse lu factorizations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 898–909, 2018.

[12] Michael B Cohen, Jonathan Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 583–592. IEEE, 2016.

[13] Michael B Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 410–419, 2017.

[14] Peter Elias, Amiel Feinstein, and Claude Shannon. A note on the maximum flow through a network. *IRE Transactions on Information Theory*, 2(4):117–119, 1956.

[15] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

[16] Shimon Even and R Endre Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518, 1975.

[17] Andrew V Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM (JACM)*, 45(5):783–797, 1998.

[18] James B. Orlin. Max flows in o(nm) time, or better. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 765–774, New York, NY, USA, 2013. Association for Computing Machinery.

[19] Ali Kemal Sinop, Lisa Fawcett, Sreenivas Gollapudi, and Kostas Kollias. Robust routing using electrical flows. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '21, page 282–292, New York, NY, USA, 2021. Association for Computing Machinery.