The Maximum-Flow problem in undirected graphs

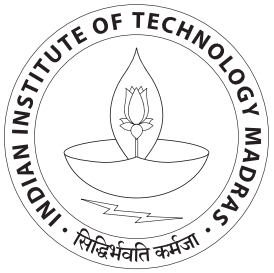
A Project Report

submitted by

KARTHIK ABINAV S

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

May 2014

THESIS CERTIFICATE

This is to certify that the thesis entitled **The Maximum-Flow problem in undirected graphs**, submitted by **Karthik Abinav S**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. N. S. Narayanaswamy
Research Guide
Associate Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

Acknoledgements

ABSTRACT

KEYWORDS: Graph Theory, Maximum Flow, Laplacian solvers

Maximum flow problem has been a very important optimization problem in computer science and mathematics. This problem has a lot of practical relevance. Some of the age-old applications involving maximum flow have been in electrical circuits, water supply networks, etc. With the advent of social media and social networks, this problem has found a newer practical relevance. And since the graphs in these networks are typically very large, researchers are sought after creating faster and more efficient algorithms for this problem.

Some of the classical algorithms to solve this problem are the Ford-Fulkerson's augmenting path algorithm and the Dinic's Algorithm. These algorithms compute the exact value of the maximum flow. It is also fairly straightforward to obtain the optimal flow vector after the termination of the algorithm. The main drawback with this algorithm is that the running time, though polynomial, is very high and is expensive to use in many practical situations. Following these algorithms a series of algorithms based on push-relabel and blocking flow techniques were devised which had a slightly better running time as compared to the classical algorithms. The series of algorithms terminated with the algorithm by Goldberg-Rao which gave a O($\min(n^{\frac{2}{3}}, m^{\frac{1}{2}}) \log(\frac{n^2}{m}) \log U$)-time algorithm for edge capacities in $\{1, 2, \ldots, U\}$. For extremely large graphs, as in the case of social networks, this algorithm is still far from being practical.

In 2008, the breakthrough result by Spielman and Teng gave an algorithm to solve the Symetric Diagonally Dominant(SDD) system of equations in near-linear time. This work was immediately extended by Koutis-Miller-Peng which gave an efficient algorithm to produce an incremental graph

sparsifier. Development of these techniques led to the development of an almost linear time algorithm to the maximum flow problem by Cristiano-Kelner-Madry-Spielman-Teng. This involved looking at the graph as a resistive network, approximating the electrical flow and producing an approximate s-t flow from this approximate electrical flow. This algorithm broke the running time barrier of Golderg-Rao and gave the first almost-linear time algorithm.

In this thesis, we give a survey of the above algorithms for the maximum flow problem. We first start off by giving a brief description of the classical algorithms. We then conclude this by giving a completely recursive specification for the push-relabel algorithm. We then present the required tools from spectral graph theory and linear algebra that is required to understand the almost linear time algorithm. Finally, we present the Cristiano-Kelner-Madry-Spielman-Teng algorithm in detail. We will also present some of the key theorems from the Koutis-Miller-Peng SDD solver. Finally, we give some arguments to justify the requirement of the weight updates and also show some steps that can possibly lead to making the algorithm parallel.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
BFS	Breadth First Search
DFS	Depth First Search
SDD	Symmetric Diagonally Dominant

NOTATION

m	The number of edges in a graph G
n	The number of vertices in a graph G
u	A $m * 1$ matrix containing the capacities of the edges
U	This defines the ratio between the highest and lowest capacity in the graph, i.e. $\frac{\max\limits_{e} u_e}{\min\limits_{e} u_e}$

CHAPTER 1

INTRODUCTION

Maximum-Flow problem is an age old optimization problem in mathematics and computer science. Hundreds of researchers have investigated this problem and have given some interesting results. This problem has far-reaching applications in almost every field of engineering and sciences. Hence, this problem is of great importance, not just theoretically, but also in real-world applications. Devising better and more efficient algorithms for this problem and its variants has always been the pursuit. Inspite, of having had so much research into this problem, for many years we were far from getting an algorithm that runs in time that is good for practical purposes. This gives an indication of the hardness of this problem.

1.1 Basics

Graph

A *graph* G(V, E, ρ) is defined as a mathematical structure on the set of vertices V and the set of edges E and an adjacency function $\rho : V \times V \rightarrow E$, such that $\rho(a, b) = e$ for $a, b \in V$ and $e \in E$ tells that there exists an edge e between the vertices a and b.

Undirected Graph

An *undirected graph* is a graph where the function ρ is symmetric, i.e. $\rho(a, b) = \rho(b, a)$.

Directed Graph

An *directed graph* is a graph where the function ρ is not necessarily a symmetric function.

Capacitated Graph

A *capacitated graph* is a graph defined as G(V, E, ρ, ψ), where the first three values in the tuple mean the same as before. The ψ in the definition is a function $\psi : E \to \mathbb{R}$, which assigns a real number corresponding to every edge e in the graph. This real number is called the *capacity* of the graph.

s-t Flow

A *s-t flow* is a vector $f \in \mathbb{R}^{|E|}$ such that the following two criteria hold:

- Flow Conservation:

$$\sum_{e \in E^+(v)} f(e) - \sum_{e \in E^-(v)} f(e) = 0 \quad \forall v \in V \setminus \{s, t\}$$

  where $E^-(v)$ denotes the set $\{a : \rho(a, v) \text{ is defined}\}$ and $E^+(v)$ denotes the set $\{a : \rho(v, a) \text{ is defined}\}$

- Capacity maintenance:

$$f(e) \leq \psi(e) \quad \forall e \in E$$

Additionally, the *value* of the flow f is a real number F such that,

$$| f | = F = \sum_{e \in E^+(s)} f(e) - \sum_{e \in E^-(s)} f(e)$$

s-t Cut

A *s-t cut* is defined as a partition of the vertex set V into U_1 and U_2 such that, $s \in U_1$ and $t \in U_2$ and the vertices in $V \setminus \{U_1, U_2\}$ is present exactly in one of U_1 and U_2.

Minimum s-t Cut Problem

Given a weighted graph G, let $\phi(\{U_1, U_2\})$ denote the sum of weights of edges such that one of its endpoints is in U_1 and the other end point is in U_2. The *minimum s-t cut problem* is to select that cut, among all possible s-t cuts, whose value of ϕ is minimized.

1.2 Maximum Flow Problem

Given a capacitated graph G(V, E, ρ, ψ), a source vertex s and a sink vertex t, the goal of the problem is to find a s-t flow such that the value of the flow is maximized among all possible s-t flows.

1.2.1 Simple Linear Program definition

This problem can be posed as a linear programming problem. The first observation is that since LP's are efficiently solvable in polynomial time, this will already give us a polynomial time algorithm. Though, this is helpful, this is not the main goal. The dual to the linear program has a specific interpretation in graph theory. Hence, one can devise potentially more faster algorithms by using some primal-dual methods.

$$\max \sum_{u:(s,u)\in E} f((s,u))$$

subject to

- $\forall v \in V \setminus \{s,t\} \sum_{(u,v)\in E} f((u,v)) = \sum_{(v,w)\in E} f((v,w))$
- $\forall (u,v) \in E\ f((u,v)) \leq c((u,v))$
- $\forall (u,v) \in E\ f((u,v)) \geq 0$

Note that a tuple (a, b) represents an edge e whose end points are a and b.

Clearly, the vector $f \in \mathbb{R}^{|E|}$ forms the set of variables in this LP. The constraints are given to ensure that the set of solutions are the set of valid s-t flows. hence, a constraint on the flow conservation on f and the capacity maintenance on f. Hence, for the given problem instance, the number of variables are polynomial in $\mid E \mid$ and $\mid V \mid$ and the number of constraints are also polynomial in

$| E |$ and $| V |$.

Sometimes, an alternative formulation of the LP is given for this problem.

$$\max \sum_{i \in P} f(i) - \lambda(\sum_{v \in V} F(v))$$

subject to:

- $\forall (j,k) \in E \; \gamma((j,k)) \leq c((j,k))$
- $\forall i \in P$,
$$f(i) \leq \gamma((j,k)) \quad \forall (j,k) \in i$$
- $\forall v \in V$,
$$F(v) = \sum_{(j,k) \in E_{in}} \gamma((j,k)) - \sum_{(j,k) \in E_{out}} \gamma((j,k))$$
- $f \succeq 0$, $F \succeq 0, \gamma \succeq 0, \lambda \geq 0$

Here, f is a vector from $\mathbb{R}^{|P|}$, where $| P |$ is the number of s-t paths in the graph. F is a vector from $\mathbb{R}^{|V|}$ and γ is a vector from $\mathbb{R}^{|E|}$. $f(i)$ denotes the flow that goes through the path i. $F(v)$ denotes the flow that is accumulated in the vertex v and $\gamma((i,j))$ denotes the flow that flows through the edge (i,j).

Clearly, the number of variables and the number of constraints are both exponential. Hence, we can't really hope to solve this in polynomial time. The only advantage of this formulation is that a optimal solution to this LP, helps one visualize the amount of flow that goes through each path independently. This formulation helps understanding of some of the path augmenting algorithms easier.

1.2.2 Dual of the Maximum Flow LP

The dual to the maximum flow LP has a very interesting connection to graph theory. In fact, as observed ahead, the dual is a relaxation to the *Minimum Cut Problem*.

$$\min \sum_{(u,v)\in E} c((u,v)) * y((u,v))$$

subject to:

- $\sum_{(u,v)\in p} y((u,v)) \geq 1 \qquad \forall p \in P$
- $y((u,v)) \geq 0 \qquad \forall (u,v) \in E$

Here, y is the dual vector corresponding to the constraints in the primal. Hence, $y \in \mathbb{R}^{|E|}$. Here P is the set of all s-t paths in the graph.

From the constraints, we can now interpret this dual as the relaxation to the s-t minimum cut problem. The vector y can be interpreted as the weight assigned to each edge in the graph. For every s-t path, the sum of edge weights in that path should be atleast one. And the objective is to minimize the weighted sum of these edge weights given by the vector y. Hence, one optimal solution to this above LP will consist of assigning a weight of 1 to the edge with minimum c in every s-t path and 0 to the other edges. This, in fact, gives the value of the minimum s-t cut in the graph and the edge with weights 1 forms the edge across the cut.

Now, consider the following randomized rounding procedure to find a cut (U_1, U_2) such that $\phi((U_1, U_2)) \leq OBJ$ where OBJ is the value of the objective for the dual, for a given vector y.

Let the values given by the vector y be weights of the edge. Now, consider any shortest path from s to v, and let d(v) be that shortest distance for vertex v. Choose a value of h uniformly at random from $[0, 1)$. Consider the set $R = \{v : d(v) \leq h\}$. Clearly, from the constraints above, corresponding to any feasible solution to the dual, t will not be contained in the set R and s will be contained in the set R. Hence, R forms a valid s-t cut.

From linearity of expectation, we get

$$\mathbb{E}[\phi(R)] = \sum_{(i,j)\in E} c((i,j))\mathbb{P}[i \in R \wedge j \notin R]$$

$$\mathbb{P}[i \in R \wedge j \notin R] = \mathbb{P}[d(i) \leq h < d(j)] = d(j) - d(i)$$

Also, from triangle inequality note that

$$d(j) \leq d(i) + y((i,j))$$

Hence, $\mathbb{E}[\phi(R)] \leq OBJ$ and therefore, there exists a cut (U_1, U_2), such that $\phi((U_1, U_2)) \leq OBJ$.

Hence, the dual to the maximum flow problem is a relaxation of the s-t minimum cut problem.

1.3 Application of Maximum Flow problem in Computer Vision

In this section, we consider the problem of Energy Minimization, which is a problem very well studied in the computer vision community. Following the lines of [insert link to paper], we show how this particular problem can be posed as a maximum flow problem. Hence, this will strengthen the importance of this problem outside the theory community.

The energy function considered in the computer vision community [Link to grieg, et al paper] is usually represented as :

$$E(L) = \sum_{p\in P} D_p(L_p) + \sum_{(p,q)\in N} V_{p,q}(L_p, L_q)$$

where $L = \{L_p : p \in P\}$ is a labeling of the image P, D_p is a data penalty function, $V_{p,q}$ is an interaction potential, and N is a set of all pairs of neighboring pixel to the current pixel.

Assume the case of binary coloring. We want to colour each pixel with either a black or a white label. The graph is constructed as follows:

- Each pixel in the image forms a vertex in this graph.
- Additionally, two vertices are added, the vertex corresponding to a black label(s-vertex) and the vertex corresponding to a white vertex(t-vertex).
- The edges are classified into two types - the T-links and the N-links.
  - The N-links are edges among neighboring pixels in the image. The cost of these N-links are the penalty assigned for the discontinuity between the pixels. This can be obtained from the pixel interaction term $V_{p,q}$ in the energy equation.
  - The T-links, are the edges between the pixel and the terminal node which corresponds to either of the two colours. It signifies the penalty of assigning the label corresponding to the terminal to that pixel. This is obtained from the term D_p in the energy equation.

  Any s-t cut partitions the pixels into two disjoint groups. Some vertices present in the same partition as the black label vertex and other vertices present in the same partition as the white label vertex. Hence, for an appropriate setting of the weights based on the parameters of an energy, a minimum s-t cut cost will correspond to a labeling with the minimum value of this energy. [link to that paper]

CHAPTER 2

Classical Algorithms

In this chapter, we describe some of the classical algorithms to find the exact value of the maximum flow in an undirected graph. These algorithms have been widely used and give a lot of insight into the nature of the problem.

2.1 Ford-Fulkerson Augmenting Path Algorithm

This is the probably the most widely known algorithm for this problem. This algorithm falls into the class of primal-dual algorithms.

Algorithm 2.1: Ford-Fulkerson algorithm to compute the maximum flow

for *every edge (i,j)* $\in E$ **do**
    Initialize f((i,j)) = 0;
end
while $\exists$ *a path p from s to t in the residual network* G_f **do**
    $c_f(p) = \min\{c_f((i,j)) : (i,j) \in p\}$;
    for *each edge (i,j)* $\in p$ **do**
        if *(i,j)* $\in E$ **then**
            f((i,j)) = f((i,j)) + $c_f(p)$;
        else
            f((i,j)) = f((i,j)) - c_f(p);
        end
    end
end

The running time of this algorithm mostly depends the augmentation method to obtain the residual network. We can show that for a suitably ill-set of edge capacities, the number of iterations taken by the above algorithm can be very large if the augmenting paths are not chose carefully. Dinic's algorithm precisely solves this part by choosing the shortest augmenting path in each iteration and hence, bounding the overall running time to $O(V^2E)$.

2.2 Dinic's Algorithm

This algorithm is almost identical to the Ford-Fulkerson algorithm, except for the fact that it chooses the shortest augmenting path using a Breadth-First search.

Algorithm 2.2: Dinic's algorithm to compute the maximum flow

for *every edge (i,j)* $\in E$ **do**
    Initialize f((i,j)) = 0 ;
end
while $\exists$ *a* ***shortest unweighted path p*** *from s to t in the residual network* G_f **do**
    $c_f(p) = \min\{c_f((i,j)) : (i,j) \in p\}$;
    for *each edge (i,j)* $\in p$ **do**
        if *(i,j)* $\in E$ **then**
            f((i,j)) = f((i,j)) + c_f(p);
        else
            f((i,j)) = f((i,j)) - c_f(p);
        end
    end
end

The part of the algorithm that is in bold is the only difference between this algorithm and Ford-Fulkerson's algorithm. The advantage however is that now, the number of residual graphs generated is bounded independent of the edge capacities.

The above algorithms were the most efficient known for a long time in the literature until, goldberg and others subsequently introduced the class of algorithms known as the push-relabel algorithms. The algorithms above always maintained edge capacities, while violating the reservoir capacity at the vertices. Push-relabel algorithms on the other hand, do the converse. In every iteration, the flow is conserved at every vertex, whereas the edge capacity may be violated. And hence, the algorithm progresses towards "correcting" these violations at the edges.

2.3 Push-Relabel Algorithms

In this section, we will describe the general template of any push-relabel algorithm and present a fully recursive specification of the relabel-to-front algorithm. A completely recursive specification hence, allows programmers to directly to adapt it into languages such as prolog.

Generic Template of the push-relabel algorithm

In this section we will describe a generic template for any push-relabel algorithm. This is adapted from the text given by [bibreference to cormen]

Associate with every vertex in the graph two parameters, namely potential and reservoir.

- The potential is a measure of knowing the direction in which particular flow should be sent. In some sense, it is used to measure the progress of the algorithm between two iterations of the algorithm. The flow is always sent from a vertex of higher potential to a vertex of lower potential.

- The reservoir stores the excessive flow that has been sent to this vertex. Since, in this algorithm the flow conservation is maintained, some flow that has already been sent to this vertex cannot be routed further ahead. Hence, this keeps track of the excessive flow that accumulates at every vertex.

Algorithm 2.3: Initializing the preflow

```
begin INIT_PREFLOW (G,s):
    for every vertex v ∈ V do
        potential(v) := 0 ;
        reservoir(v) := 0 ;
    end
    for each edge (i,j) ∈ E do
        f((i,j)) := 0 ;
    end
    potential(s) = n ;
    for each vertex v ∈ N(s) do
        f((s,v)) = c((s,v)) ;
        reservoir(v) = c(s,v) ;
        reservoir(s) = reservoir(s) - c((s,v)) ;
    end
end
```

The above procedure initializes the required variables and vectors before the start of the algorithm.

Overflowing vertex : We say a vertex v is *overflowing*, if the reservoir at vertex v has positive flow accumulated in it, i.e. reservoir(v) > 0.

As the name suggests, at every iteration we associate two types of operations, a push operation and the relabel operation.

- *Push operation*: This operation is performed on vertices u and v by pushing some of the accumulated flow at u to vertex v. This is applicable only when the vertex u is overflowing, $c_f((u, v)) > 0$ and potential(u) = potential(v) + 1. Here, c_f refers to the capacities in the residual network with respect to the flow f. The operation involves pushing $\min$(reservoir(u), $c_f((u, v))$) units of flow from u to v.

- *Relabel*: This operation involves re-assigning the potential of the vertex u. This is applicable only when u is overflowing, and $\forall v \in V$ such that $(u, v) \in E_f$, we have potential(u) $\leq$ potential(v). Here, E_f = $\{(i, j) \in E : f((i, j)) < c((i, j))\}$. The operation involves re-assigning the potential of this vertex u with one more than the minimum of all such v.

Algorithm 2.4: Push Operation

begin PUSH(u, v):

$\Delta_f((u, v)) = \min($reservoir(u), $c_f((u, v)))$;

if *(u,v)* $\in E_f$ **then**

f((u,v)) = f((u,v)) + $\Delta_f((u, v))$;

else

f((v,u)) = f((v,u)) - $\Delta_f((u, v))$;

end

reservoir(u) = reservoir(u) - $\Delta_f((u, v))$;

reservoir(v) = reservoir(v) + $\Delta_f((u, v))$;

end

Algorithm 2.5: Relabel Operation

begin RELABEL(u):

potential(u) = 1 + $\min\{$potential(v) : $(u, v) \in E_f\}$;

end

With the above two functions available, a generic push-relabel algorithm looks as follows:

Algorithm 2.6: Push Relabel Template

begin MAX-FLOW(G, s, t):

INIT_FLOW(G, s) ;

while $\exists$ *an applicable push or relabel operation* **do**

Call the appropriate push or relabel operation ;

end

end

The proof of correctness of this algorithm can be shown by using the potential function on the

vertices. We omit the proof here. The reader is encouraged to read [bib to cormen] for more details.

Based on the specific implementation of the template i.e. the order of push and relabel operations performed, we can bound the running time of the algorithm. In the following sub-section, we give a fully recursive specification of the relabel-to-front algorithm and bound the running time of this algorithm.

2.4 Goldberg-Rao Blocking Flow Algorithm

In this section, we briefly describe the idea behind the goldberg-rao's algorithm. The description of this algorithm is adapted from [link to the lecture notes containing the algorithm]

Before describing the main algorithm, let us first have some definitions in place.

Delta indicator: We define a vector L, which is a delta indicator for a particular flow f i.e.

$$L((i,j)) = \begin{cases} 1 & u_f((i,j)) \leq \Delta \\ 0 & \text{otherwise} \end{cases}$$

Admissible Edge: We call an edge $(i,j) \in E_f$ admissible with respect to the flow f, if dist(i) = dist(j) + L_f((i,j)). Here, the dist(v) refers to the shortest distance of vertex v to sink t.

Blocking flow: A flow f is *blocking* if every s-t path in the graph G has atleast one completely saturated edge.

Algorithm 2.7: Goldberg-Rao Algorithm

begin Goldberg-Rao(G, s, t):

Initialize F as mU, where U = $\max_{(i,j)\in E} u((i,j))$;

Initialize f as 0 ;

Define Λ as $\min(m^{\frac{1}{2}}, n^{\frac{2}{3}})$;

while $F \geq 1$ **do**

Assign Δ as $\frac{F}{2*\Lambda}$;

for i:=1 to $5*\Lambda$ **do**

- Compute the Delta indicator vector L ;

- Compute dist(i), using the vector L as the vector of edge lengths ;

- Shrink the strongly-connected components of admissible edges ;

- Find a flow $\tilde{f}$ in the shrunken graph s.t. it is either a blocking flow or value of the flow is $\frac{\Delta}{4}$;

- Assign $\bar{f}$ as $\tilde{f}$ with the flows completely routed within the edges of the shrunken components.;

- $f = f + \bar{f}$;

end

F = $\frac{F}{2}$;

end

end

In the above algorithm, *shrinking* a strongly connected component(SCC) means, replacing the entire SCC by a single vertex, whose incoming edges are the incoming edges of all vertices of the SCC, and whose outgoing edges are outgoing edges of all vertices of the SCC.

As again, we encourage the reader to refer to [link to the notes] for a complete analysis of the correctness of the above algorithm and its running time.

CHAPTER 3

Spectral Graphs

In this chapter, we give some requisite mathematics from spectral graph theory. The definitions and theorems stated in this chapter, will be the key to understanding the breakthrough Cristiano-Kelner-Madry-Spielman-Teng algorithm and our attempts and observations. We only give the required amount of definitions and would avoid most proofs. This section is adapted from the monograph [link to lx=b]. For a detailed explanation of some of these topics, the reader is advised to refer to [link to Lx=b].

3.1 Eigen Values

Throughout this report, we will be dealing with square matrices which are symmetric unless explicitly mentioned otherwise.

Eigen value: A value λ is called the *eigen value* of the matrix A if there exists a vector x such that $A.x = \lambda.x$. This vector x is called the *eigen vector* of the matrix A.

Proposition: Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigen values of a $n*n$ matrix A. Let $v_1, v_2, \ldots, v_n$ be the corresponding eigen vectors. Then, the matrix A can be written as $\sum_{i=1}^{n} \lambda_i v_i v_i^T$

Alternate characterization of eigen values:

The k^{th} smallest eigen value can be expressed as

$$\lambda_k(A) = \min_{x \in \mathbb{R}^n \setminus \{0\} \,:\, x^T.v_i = 0 \;\; \forall i \in \{1,2,\ldots,k-1\}} \frac{x^T A x}{x^T x}$$

Alternatively, it can also be expressed as:

$$\lambda_k(A) = \max_{x \in \mathbb{R}^n \setminus \{0\} \,:\, x^T.v_i = 0 \;\; \forall i \in \{k+1,k+2,\ldots,n\}} \frac{x^T A x}{x^T x}$$

3.2 Graph Laplacian

Graph Laplacian is a $n * n$ matrix which gives lot of algebraic properties of a graph. This matrix is useful for testing many properties of a graph algebraically. For example, one can test connectivity of a graph using the eigen values of this matrix. In fact, we can also extend this and find the number of connected components in the graph by using the eigen values.

Let A be the adjacency matrix of the graph and let D be a diagonal matrix which contains the degree of the vertex i in the i^{th} entry of the principal diagonal and 0 elsewhere. The *graph laplacian* L is defined as $D - A$.

Proposition: The graph Laplacian L is a positive semi-definite matrix i.e. $\lambda_1(L) \geq 0$.

In fact, we can also claim that graph laplacian is not positive definite. It can be easily shown that $\lambda_1(L) = 0$.

The above definition also translates to a weighted graph, where every edge of weight w is replaced by w parallel edges and use the same definition as above.

One important advantage of this laplacian is that we can test connectivity properties using the eigen values of this matrix. The following theorem gives the exact relationship.

Theorem:

A graph G is connected if and only if the second smallest eigen value of its laplacian is strictly greater than 0.

In fact, a more generalized version of the above theorem can be shown.

Theorem:

Let G be a graph and L be the corresponding laplacian matrix. Let $\lambda_1(L) \leq \lambda_2(L) \leq \ldots \leq \lambda_n(L)$ be the eigen values of L. Let i be such that $\lambda_i(L) = 0$ and $\lambda_{i+1}(L) > 0$. Then, the graph G has exactly i connected components.

Plugging $i = 1$ in the above equation gives the theorem on connectivity of the graph.

3.3 Laplacian system of equations

A special subset of system of linear equations is a laplacian system of equations. In this case $A.x = b$ system, has the matrix A as a graph laplacian corresponding to some graph G. Hence, these are equations of the form $L.x = b$. It turns out that some optimization problems over graphs can be posed as a laplacian system of equations. In fact, one such procedure is used in the Cristiano-Kelner-Madry-Spielman-Teng algorithm and hence, being able to solve this system effectively is a requirement.

We know that a gaussian procedure to solve $A.x = b$ takes about $O(n^3)$ steps. This is not prac-

tically relevant since we are looking for flow algorithms that are close to linear time. But, it turns out that for the case of laplacian systems there exists nice solvers which can solve it efficiently. The following theorem states this more precisely.

Theorem:

There exists an algorithm for solving the Laplacian system which takes as input a graph laplacian L, a vector b and an error parameter ϵ and returns a solution x to the system such that:

$$\|x - L^{-1}b\|_L \leq \epsilon \|L^{-1}b\|_L$$

Additionally, this algorithm runs in time $\tilde{O}(m \log(\frac{1}{\epsilon}))$ where m is the number of non-zero entries in the laplacian matrix.

We would like to refer to the reader [link to olivia simpson's research exam] for a brief survey on the development of various Laplacian solvers.

3.4 Electrical Flows

The key idea leading to the breakthrough of the newer algorithm was to visualize the graph as an electrical network. Every edge corresponds to a resistor of some appropriate value. A voltage of one unit is applied across the vertices s and t. The network thus formed is an electrical network.

In this section, we will formalize the above notion and provide some useful definitions and theorems.

Incidence Matrix:

Consider an arbitrary orientation of the edges in an undirected graph G. We define a $m * n$ matrix

B, which is called the *incidence matrix*, as follows:

$$B(i,j) = \begin{cases} -1 & \text{if edge i leaves vertex j} \\ 1 & \text{if edge i enters vertex j} \\ 0 & \text{otherwise} \end{cases}$$

The graph laplacian can now be related to this incidence matrix B by the following theorem.

Theorem:

For a graph G having an incidence matrix B and a laplacian L, $B^T B = L$.

Note in the above theorem, there is no mention of the orientation of the edges for B. Hence, it is true for any arbitrary orientation of the edges.

Current vector: As stated before, we can look at a graph as an electrical network. Replace every undirected edge by a resistor of resistance 1. Now, associate a voltage difference of v across the vertices s and t. The current flowing through each of the resistors(which are edges in the original graph) is given by the *current vector*. It is denoted as c.

Since, the above is an electrical network, we know that the current and voltage satisfies the Kirchoff's current and voltage laws. Using the laws, we can easily show that, $B^T B.v = c$ and hence, $L.v = c$. Therefore, given the current flowing through each of the edges and the laplacian, finding the vertex potentials is essentially solving a laplacian system of equations.

Effective Resistance: Given a particular current vector c, we define the effective resistance of an edge $e = (i, j)$ as the potential difference across e when a current of value 1A is sent across i to j.

Proposition: The effective resistance $R_{eff}(e) = (e_i - e_j)^T L^{-1}(e_i - e_j)$. Here, e_i is a vector of length $\mid V \mid$ which has 1 at the i^{th} location and 0 elsewhere.

We would like to know what is the voltage across a particular edge, when a current of 1A is put across a fixed set of two vertices s and t. To do that we will now define the following matrix called the Π *matrix*.

Π **matrix**:

This matrix is defined on $\mathbb{R}^{|E|*|E|}$.

$$\Pi(e, f) = b(e)^T L^{-1} b(f)$$

$b(e)$ is the e^{th} row in the matrix B.

Hence, $\Pi = B^T L^{-1} B$. It is easy to check the following properties of this matrix

- Π is symmetric
- $\Pi^2 = \Pi$
- All eigen values of Π are 0 or 1

Electrical flow: Let 1A of current be inducted at vertex s and removed at vertex t. We define f^* as the electrical flow and is given by $BL^{-1}(e_s - e_t)$.

Electrical energy: Let r be the vector of resistances of every edge(Here it is the all 1's vector). Let f be the vector of electrical current through each edge. The electrical energy is

$$\epsilon_r(f) = \sum_e r_e f_e^2$$

Lemma:

Among all valid flows of value F, the one that minimizes the energy is the electrical flow.

Proof:

Consider the case of two resistors in series and two resistors in parallel. If we prove the lemma for these two cases, it applies for any general circuit.

Series: In this case, there is just one possible flow, which is the electrical flow. Hence, statement trivially holds true.

Parallel: Let us start with an electrical flow. We will show that any change to the flows will only increase the energy. Let i_1 and i_2 be the electrical flow through resistors r_1 and r_2. From Kirchoff's law we have $i_1 * r_1 = i_2 * r_2$. The energy now is $\epsilon = r_1 * i_1^2 + r_2 * i_2^2$. Let us now redistribute the flow as $i_1 + \delta$ and $i_2 - \delta$. The value of the flow remains the same. The energy now becomes $\epsilon' = \epsilon + \delta^2(r_1 + r_2) \geq \epsilon$. Hence the lemma.

Proposition: For r being the all 1's vector and an electrical flow f^*, the energy of this flow is given by

$$\epsilon(f^*) = (e_s - e_t)^T L^{-1}(e_s - e_t)$$

CHAPTER 4

Cristiano-Kelner-Madry-Spielman-Teng Algorithm

In this chapter we describe the Cristiano-Kelner-Madry-Spielman-Teng algorithm[CKMST] in full detail. This will help the reader understand this new approach which manages to get to a near linear running time for the maximum flow problem in undirected graphs. Later in the section we will also provide some observations and give a method which could possibly lead to a parallel version of the same. This might help in making the algorithm run a even larger graphs.

4.1 Overview

The key idea of the CKMST algorithm is that given an electrical circuit, an approximate set of currents and node potentials can be calculated very efficiently using some of the fast SDD solvers. The research on algorithms for laplacian solvers is a closely related topic. But, in this report we will avoid stating them in great detail and just state the necessary theorems whenever needed. The CKMST algorithm in particular, uses the Koutis-Miller-Peng algorithm[Link to the Koutis, Miller, Peng paper] as a subroutine. Given, an approximation to the electrical currents and potentials(in short we call this the electrical flow), the algorithm, then constructs two subroutines.

- (ϵ, ρ) **- oracle**: The authors call this an oracle since, given a flow value F and a graph G whose every edge capacity is 1, this subroutine either returns a flow f whose edge capacities are violated by atmost ρ on each edge and whose average is atmost ϵ or returns a fail which indicates that the required flow value F is too large. Note, that having an upperbound on the average makes the sub-routine do more work.
- **Weights update routine**: This function assigns different weights to each edge in each iteration and then uses the flow returned by the oracle for a suitable ϵ and ρ to finally return a flow which does not violate edge capacity at all edges. The main intuition behind why a flow that has violated edge capacities in some iteration can be modified to give a flow that does not violate any capacity is the fact that the oracle maintains the average capacity as a weighted average over the weights used by the weights update routine. Hence, by suitably modifying the weights across iterations, one can obtain a required flow.

This algorithm achieves a $(1-\epsilon)$- approximation to maximum flow problem in undirected graphs. The main drawback with this algorithm, as we will see ahead, is that it doesn't actually give a flow vector, but merely gives the value of the flow. This issue was resolved in a follow-up work by Orecchia, Kelner et al[link to paper], which adapts a completely different approach using a variety of new techniques such as non-euclidean gradient descent, construction of oblivious routing and flow sparsifiers.

An important attempt has been to make this algorithm parallel or atleast "less" serial. To do that efficiently, a two step method is required. Firstly, the update routine itself should be modified to ensure that some updates can be done in parallel. In this regard, we make certain observations on the update procedure which will lead to a modification in making every two iterations into two parallel iterations. Although the main crux of the algorithm is in the weights update method, it is also helpful to try and make the subroutine that computes the approximate electrical flow parallel. This requires the SDD solver itself to have a parallel implementation. In this regard some progress has been made recently by [link to the new STOC paper].

CHAPTER 5

A SAMPLE APPENDIX

Just put in text as you would into any chapter with sections and whatnot. Thats the end of it.

Publications

1. S. M. Narayanamurthy and B. Ravindran (2007). Efficiently Exploiting Symmetries in Real Time Dynamic Programming. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2556–2561.

REFERENCES

Amarel, S., On representations of problems of reasoning about actions. *In* **D. Michie** (ed.), *Machine Intelligence 3*, volume 3. Elsevier/North-Holland, Amsterdam, London, New York, 1968, 131–171.

Barto, A. G., **S. J. Bradtke**, and **S. P. Singh** (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, **72**, 81–138.

Bellman, R. E., *Dynamic Programming*. Princeton University Press, 1957.

Crawford, J. (1992). A theoretical analysis of reasoning by symmetry in first-order logic. URL `citeseer.ist.psu.edu/crawford92theoretical.html`.

Griffiths, D. F. and **D. J. Higham**, *Learning LaTeX*. SIAM, 1997.

Knoblock, C. A., Learning abstraction hierarchies for problem solving. *In* **T. Dietterich** and **W. Swartout** (eds.), *Proceedings of the Eighth National Conference on Artificial Intelligence*. AAAI Press, Menlo Park, California, 1990. URL `citeseer.ist.psu.edu/knoblock90learning.html`.

Kopka, H. and **P. W. Daly**, *Guide to LaTeX (4th Edition)*. Addison-Wesley Professional, 2003.

Lamport, L., *LaTeX: A Document Preparation System (2nd Edition)*. Addison-Wesley Professional, 1994.

Manning, J. B. (1990). *Geometric symmetry in graphs*. Ph.D. thesis, Purdue University.

Ravindran, B. and **A. G. Barto** (2001). Symmetries and model minimization of markov decision processes. Technical report, University of Massachusetts, Amherst.