The Maximum-Flow problem in undirected graphs

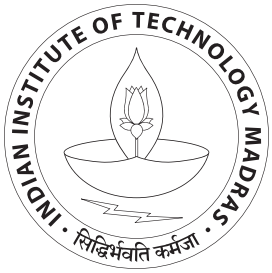
A Project Report

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 $f \in \mathbb{R}^{\mathsf{E}}$ 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,

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

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 $f \in \mathbb{R}^{\mathsf{E}}$ 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 E and V 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:

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

Here, f is a vector from $\mathbb{R}^{\mathsf{P}}$, where P is the number of s-t paths in the graph. F is a vector from $\mathbb{R}^{\mathsf{V}}$ and γ is a vector from $\mathbb{R}^{\mathsf{E}}$. $f(i)$ 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, $y \in \mathbb{R}^{\mathsf{E}}$. 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.

Ford-Fulkerson algorithm to compute the maximum flow

begin

for every edge (i,j) $\in E$ do

Initialise f((i,j)) = 0

od

while $\exists$ a path p from s to t in the residual network G_f

$c_f(p) = \min\{c_f((i,j)) : (i,j) \in p\}$

for each edge (i,j) $\in p$ do

if $(i,j) \in E$

then

$f((i,j)) = f((i,j)) + c_f(p)$

else

$f((i,j)) = f((i,j)) - c_f(p)$

fi

od

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.

Dinic's algorithm to compute the maximum flow

begin

for every edge (i,j) $\in E$ do

Initialise f((i,j)) = 0

od

while $\exists$ a **shortest unweighted path p** from s to t in the residual network G_f

$c_f(p) = \min\{c_f((i,j)) : (i,j) \in p\}$

for each edge (i,j) $\in p$ do

if $(i,j) \in E$

then

$f((i,j)) = f((i,j)) + c_f(p)$

else

$$f((i,j)) = f((i,j)) - c_f(p)$$

fi

od

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.

proc $INIT\_PREFLOW(G, s)$:

begin

for every vertex v $\in V$ do

$potential(v) := 0$

$reservoir(v) := 0$

od

for each edge $(i, j) \in E$ do

$f((i, j)) := 0$

od

potential(s) = n

for each vertex v $\in N(s)$ do

$f((s, v)) = c((s, v))$

$reservoir(v) = c(s, v)$

$reservoir(s) = reservoir(s) - c((s, v))$

od

end

The above procedure initialises 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 pusing 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.

Push Operation

proc $PUSH(u, v)$:

begin

$\Delta_f((u, v))$ = 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

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

fi

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

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

end

Relabel Operation

proc $RELABEL(u)$:

begin

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:

Push Relabel Template

proc $MAX - FLOW(G,s,t)$:

begin

INIT_FLOW(G, s)

while exists an applicable push or relabel operation do

Call the appropriate push or relabel operation

od

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. This algorithm uses the technique of blocking flow

CHAPTER 3

Spectral Graphs

CHAPTER 4

Cristiano-Kelner-Madry-Spielman-Teng Algorithm

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.