Max Flow in a Directed Planar Graph

Jason Hoch and John Wang
6.854 Final Paper

December 10, 2012

Abstract

We implement an max flow algorithm presented by Erikson [1].

Contents

1 Introduction **2**

2 Motivation **2**
2.1 Supply Chain Example 2
2.2 Computer Vision Example 3

3 Overview of Erikson Algorithm **3**
3.1 Definitions 3
3.2 Parametric Shortest Paths 4

4 Framework and Infrastructure **4**

1 Introduction

The general maximum flow problem has many applications in operations research and supply chain management. The classic algorithms used to solve the problem are polynomial, but relatively slow. The Edmonds-Karp algorithm requires $O(nm^2)$ time, while Ford Fulkerson requires $O(m \max |f|)$. Improvements made throughout the last few decades have decreased these runtimes by using concepts from blocking-flow and push-relabel algorithms. Dinitz's blocking flow algorithm improved the runtime to $O(n^2m)$ by using a breadth-first-search in the residual graph to build a layered graph [2]. Goldberg and Tarjan [3] introduced a general framework for solving max-flow problems by using the idea of push relabel. Many of the fastest algorithms for maximum flow are variants of this general framework. Recently, Orlin presented an algorithm which runs in $O(nm)$ time.

A practical application of a maximum flow algorithm, however, would find many of these algorithms prohibitively slow. Even the best known algorithm of Orlin for general graphs is $o(n^2)$ for reasonably connected graphs. However, these algorithms perform poorly due to their generality. In practice, only a certain subset of graphs are likely to arise. For instance, in many applications of maximum flow, it is reasonable to assume that the graphs will be planar.

This is the case for maximum flow networks on any type of two dimensional map. In these cases, solving the problem in its full generality does too much work, and enforcing constraints on the assumptions of the graph can improve runtime. This paper focuses on the algorithm proposed by Erikson 2010 [1], which exploits planarity to achieve $O(n \log n)$ runtime. This significantly reduces the execution time of the algorithm, at least theoretically.

This paper implements the Erikson algorithm and presents the observed runtime on a set of planar graphs. The set of planar graphs generated contains graphs of varying sizes. Our implementation of the Erikson algorithm is tested for how it scales with n, the number of vertices in a graph. We attempt to determine an empirical function for how the observed runtime scales with n, and compare it with standard library implementations of max-flow algorithms.

The rest of the paper presents a brief overview of the Erikson algorithm, the unique features in our implementation, and the object oriented infrastructure we created. We present the results from our implementation on a set of planar graphs and compare it with standard maximum flow algorithms.

2 Motivation

In this section, we provide motivation for our implementation. In particular, we motivate the use of an algorithm specific to planar graphs. We argue that most applications of maximum flow are constrained to planar graphs, and hence, that the Erikson algorithm provides a non-trivial improvement upon past algorithms.

2.1 Supply Chain Example

First, suppose a chain of department stores has a supply chain and is attempting to find the best way to move goods from its production facilities to its stores. The store must transport the goods across the network to its stores, and can stop at any intermediate warehouse in its network. There are a discrete number of warehouses that its trucks can stop at, but there are only a discrete number of trucks that it sends over any route. The goal is to achieve the highest throughput of goods possible from production facilities to its stores.

This problem can be reduced to a single source, single sink maximum flow problem by created a supernode s representing all the production facilities and a supernode t representing

the stores. Each production facility will be connected to s and each store will be connected to t with an infinite capacity edge. Finding the maximum flow between s and t will then find the total throughput for a given timespan. Notice that since this transportation problem is defined on land, it is confined to a planar network (assuming that warehouses or rest-stops are located at each path intersection).

Notice that this type of transportation problem occurs with incredible frequency. Every large chain of stores with a need to move goods across land has a similar problem. Retailers such as Amazon, Walmart, and Target have a large incentive to optimize their transportation networks, as do large shipping companies such as FedEx and UPS.

2.2 Computer Vision Example

There are also many examples of planar max-flow networks which are relevant in computer vision. In particular, finding graph cuts in an image is a widely used industrial application of max-flow. Greig, Porteous, and Seheult [4] provide an example of using max-flow for smoothing noisy images. They show that the maximum a posteriori estimate of a noisy image is given exactly by finding a max-flow in a specifically defined image network.

The authors in [4] create a network of pixels i, each with value $x_i \in \{0, 1\}$ corresponding to white and black. Each pixel i in the flow network has neighbors which are the pixels touching it in the actual image. The capacity of the edge between pixels i and j is defined as a log-likelihood ratio corresponding to the probabilitiy that the values of pixels i and j are correct. The paper shows that finding a maximum flow through the network results in a maximum a posteriori estimate of the image.

The network is clearly planar because the graph can be embedded onto a two dimensional plane (the image). Moreover, [4] spawned a large literature in the computer vision field using max-flows as a means to separate out nosie from images. Other max-flow algorithms in the computer vision field follow in a similar vein to [4].

3 Overview of Erikson Algorithm

This section provides an overview of the Erikson algorithm. We summarize the main results from the paper in [1] and sketch the proofs of the main theorems. The Erikson algorithm relies on the observation that max-flow in directed planar graphs can be reformulated as a parametric shortest paths problem. The algorithm finds a shortest paths spanning tree which whose parameter is perturbed in multiple iterations of the algorithm. Eventually, the resulting tree will arrive at a state where the max-flow can be computed with simple arithmetic.

3.1 Definitions

First, we shall present the fundamental definitions and terminology used throughout the this paper and the Erikson [1] paper. Note that the notation is the same as in [1] to reduce confusion. A graph $G = (V, E)$ will be a directed planar graph where each edge e denoted as $u \to v$ contains a head and tail defined as $head(e) = v$ and $tail(e) = u$ respectively. The reversal of the graph is defined as $rev(e) = head(e) \to tail(e)$.

Each primal graph G has a corresponding dual G^* where each primal face is converted into a dual vertex. Two dual vertices are connected by a dual edge if and only if there is a corresponding primal edge connecting the two faces represented by the two dual vertices. In particular, consider faces a and b in the primal graph, and consider some edge e which separates these two faces. Then the dual graph will contain an edge e^* (the dual of the edge e) which is oriented 90° counterclockwise to e and connects a^* and b^* in the dual.

A flow between source s and sink t is defined as a function $\phi : E \to \mathrm{R}$. The flow satisfies antisymmetry and conservation. In particular, the antisymmetric constraint specifies that $\phi(e) = -rev(\phi(e))$ while the conservation constraint specifies that $\sum_{e_w} \phi(e) = 0$ for all edges e such that $w = head(e)$. We define a feasible flow by defining a non-negative capacity function $c : E \to \mathrm{R}$ on each edge e. For a flow to be feasible on a graph, we must have $\phi(e) \leq c(e)$ for all edges e in the graph.

3.2 Parametric Shortest Paths

Before moving on to Erikson's algorithm, it is instructive to examine the parametric shortest paths problem, since Erikson's algorithm is fundamentally based on the idea of using parametric shortest paths to compute the max-flow. For this problem, consider a graph $G = (V, E)$ and an additional parameter λ. We define a new cost function $c : E \to \mathrm{R}$ and obtain a subset $E' \subset E$ of the edges of the graph. The cost function will be set to $c(\lambda, e) = c(e) - \lambda$ for all edges $e \in E'$ and $c(\lambda, e) = c(e)$ for all edges $e \notin E'$. The parametric shortest paths problem asks to compute the largest value of λ for which the resulting graph with weights $c(\lambda, e)$ has no negative-weight cycles.

A number of algorithms that solve this problem use the concept of a pivot tree. Young, Tarjan, and Orlin [**?**] solve the problem in $O(nm + n^2 \log n)$ by constructing a single-source shortest paths tree. The distance used in the shortest paths tree is equal to the cost $c(\lambda, e)$ for each edge e, given a parameter λ. The Young, Tarjan, and Orlin algorithm starts increasing λ from $-\infty$. At some point, $d(\lambda, v) = d(\lambda, u) + c(\lambda, v)$ for some edge $u \to v$ which is not in the shortest paths tree. When this occcurs, the shortest paths tree incorporates the $u \to v$ edge and removes an appropriate edge e where $head(e) = v$ so that the tree remains a valid shortest paths tree. The algorithm continues until a cycle appears in the tree. Notice that at this point, increasing λ by any amount will cause a negative-weight cycle in the original graph because the cycle must be of weight 0.

This algorithm therefore computes the maximum λ for which the graph G has no negative-weight cycles. Erikson's algorithm will use the same concept and apply it to max-flow. In particular, Erikson uses the parameter λ to pivot edges in a shortest paths tree. Using a well defined set of weights, one can show that the resulting algorithm solves the max-flow problem.

3.3 Erikson's Algorithm

The algorithm that Erikson provides in [1] first creates a shortest-paths tree, then runs a parameterized shortest-paths algorithm. It uses dynamic tree data structures in order to find pivots and remove them from the tree.

4 Framework and Infrastructure

The Erikson algorithm was implemented in Java 1.7 and is available on `https://github.com/jrshoch/msmsmaxflow`.

References

[1] Jeff Erikson. "Maximum Flows and Parametric Shortest Paths in Planar Graphs." *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*. 794-804. 2010.

[2] Yefim Dinitz. "Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation." *Doklady Akademii nauk SSSR*. 11: 1277-1280. 1970.

[3] Andrew Goldberg and Robert Tarjan. "A New Approach to the Maximum Flow Problem." *Annual ACM Symposium on Theory of Computing*. Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing. 136-146. 1986.

[4] D.M. Greig, B.T. Porteous, and A.H. Seheult. "Exact Maximum A Posteriori Estimation for Binary Images." *Journal of the Royal Statistical Society. Series B (Methodological).* 51(2): 271-279. 1989.

[5] N.E. Young, R.E. Tarjan, and J.B. Orlin. "Faster Parametric Shortest Path and Minimum Balance Algorithms." *Networks.* 21(2): 205-221. 1991.