Discrete Applied Mathematics 159 (2011) 2187–2203

Contents lists available at SciVerse ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Communication

Simplifying maximum flow computations: The effect of shrinking and good initial flows☆

F. Liers, G. Pardella*

Universität zu Köln, Institut für Informatik, Pohligstraße 1, 50969 Köln, Germany

ARTICLE INFO

Article history:
Received 16 June 2011
Accepted 18 June 2011
Available online 27 July 2011
Communicated by Miguel F. Anjos

Keywords:
Maximum flow
Minimum cut
Subgraph shrinking
Hybrid algorithm

ABSTRACT

Maximum flow problems occur in a wide range of applications. Although already well studied, they are still an area of active research. The fastest available implementations for determining maximum flows in graphs are either based on augmenting path or on push–relabel algorithms. In this work, we present two ingredients that, appropriately used, can considerably speed up these methods. On the theoretical side, we present flow-conserving conditions under which subgraphs can be contracted to a single vertex. These rules are in the same spirit as presented by Padberg and Rinaldi (1990) [12] for the minimum cut problem in graphs. These rules allow the reduction of known worst-case instances for different maximum flow algorithms to equivalent trivial instances. On the practical side, we propose a two-step max-flow algorithm for solving the problem on instances coming from physics and computer vision. In the two-step algorithm, flow is first sent along augmenting paths of restricted lengths only. Starting from this flow, the problem is then solved to optimality using some known max-flow methods. By extensive experiments on instances coming from applications in theoretical physics and computer vision, we show that a suitable combination of the proposed techniques speeds up traditionally used methods.

1. Introduction

Determining maximum flows in networks is a classical problem in the area of combinatorial optimization, with many applications abound in different fields. Elegant algorithms and fast implementations are available. They allow the determination of a solution in a time growing only polynomially in the size of the input in the worst case, and large instances can be solved in practice.

We formulate the maximum s–t flow problem in the following. We are given a network, that is a (directed or undirected) graph $G = (V, E)$. For an edge $e \in E$ capacities $c_e \geq 0$ are present. Further, we are given two vertices, called the source s and the sink t. Let $f : E \to \mathbb{R}$ be a flow function on the set of edges, and denote by f_e the amount of flow on edge e. In the following, we say flow and mean the corresponding flow function. We also do not distinguish between undirected edges and directed arcs, if not necessary. The value of a flow is the net amount of flow out of the source. The objective is to determine a maximum flow, that is a flow with maximum value $\sum_{e\in\delta^+(s)} f_e = \sum_{e\in\delta^-(t)} f_e$, with $\delta(v) = \delta^+(v) \cup \delta^-(v) = \{e \in E \mid e = (v, w)\} \cup \{e \in E \mid e = (w, v)\}$. The flow has to respect the following constraints:

$$\text{(capacity constraints) } 0 \leq f_e \leq c_e, \ \forall e \in E \tag{1}$$

$$\text{(flow conservation) } \sum_{w\in\delta^-(v)} f_{(w,v)} = \sum_{w\in\delta^+(v)} f_{(v,w)}, \quad \forall v \in V \setminus \{s, t\}. \tag{2}$$

☆ Financial support from the German Science Foundation is acknowledged under contract Li 1675/1.

* Corresponding author. Fax: +49 2214705317.
E-mail addresses: liers@informatik.uni-koeln.de (F. Liers), pardella@informatik.uni-koeln.de (G. Pardella).

0166-218X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2011.06.030

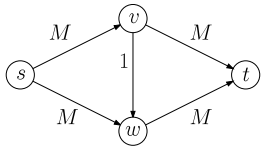

Fig. 1. Simple (directed) example such that augmenting path strategies perform many unnecessary augmentations in the worst case. Capacities are set to a very large value (indicated by M), except for the arc connecting v and w. In the worst case, one repeatedly augments flow along the residual path s, v, w, t, then along s, w, v, t, and so on. Thus, one needs $2M$ augmentations, while only two augmentations are needed if augmented along the paths s, v, t and s, w, t.

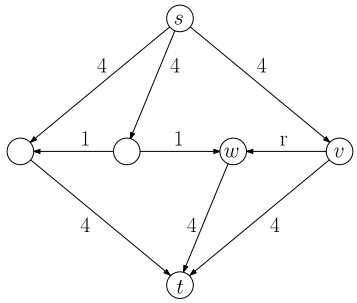

Fig. 2. Minimum counterexample from [16] for the non-termination of augmenting path strategies if using real-valued capacities $\left(r = \frac{\sqrt{5}-1}{2}\right)$. At least one augmenting path always exists that uses the (residual) edge (v, w).

The set of capacity constraints (1) ensures that any feasible flow respects the capacities. The flow conservation constraints (2) avoid flow generation or elimination other than at the source and the sink. In other words, every unit of flow that enters a vertex must leave it and vice versa. A flow is feasible if and only if the above constraints are satisfied. A flow function violating the flow conservation constraints (2) is called *preflow*. For a vertex v that violates the flow conservation constraint the difference of incoming flow and outgoing flow is called its *excess*.

The first algorithm for determining maximum s–t flows has been presented in 1956 in a seminal work by Ford and Fulkerson [5]. Flow is iteratively improved by increasing it along *augmenting paths* between the source and the sink. Given a feasible flow, an augmenting path is a path between source and sink such that additional flow can be sent to the sink. In other words, the residual capacities of all edges on the path must be strictly positive. Most maximum s–t flow algorithms rely on the concept of an auxiliary directed graph, the so-called *residual graph* $R = (V, A)$. The idea is the following. Suppose we have sent f_e units flow along edge e. Thus, we can additionally send $c_e - f_e$ units flow along edge e. On the other hand, we can also cancel f_e units of flow on edge e. Hence, for a given flow f the residual graph consists of the same vertex set. Each original edge $e = (v, w)$ is replaced by two arcs $a_1 = (v, w)$ and $a_2 = (w, v)$. The arc a_1 has residual capacity $c_e - f_e$ while arc a_2 has residual capacity f_e. One restricts the residual arcs to those with positive residual capacities. Therefore we have to be more specific about augmenting path strategies. Flow is sent along augmenting paths in the residual graph R until no such path exists. For rational capacity choices this basic augmenting path algorithm terminates. However, its running time is pseudo-polynomial. Choosing 'bad' augmenting paths yields unnecessary augmentations, see Fig. 1.

If capacities are real-valued there is no termination guarantee. In fact, Zwick [16] proposed a minimum counterexample on which augmenting path algorithms do not terminate, see Fig. 2.

Recently, Boykov and Kolmogorov [2] presented a fast implementation for determining maximum s–t flows based on augmenting paths in a more elaborate way. Two search trees are used simultaneously to determine augmenting paths. One search tree starts at the source while the other is a backward tree starting at the sink. The trees are updated dynamically in each search step. This 'double tree strategy' yields the currently fastest implementation for instances coming from computer vision.

Dinic [3] proposed a so-called 'blocking flow' algorithm that can be interpreted as a clever augmenting path strategy avoiding unnecessary augmentations. Applied to the worst case from Fig. 1 only two augmentations are made. This idea was independently proposed by Edmonds and Karp [4]. We note that these algorithms have strongly polynomial running time. Finally, the method with the best practical performance on general instances is the push–relabel algorithm by Goldberg and Tarjan [8]. We describe the general idea in the 'highest label push–relabel' form. The algorithm maintains vertex labels that correspond to the distance of a vertex to the sink in the residual graph. Initially, all vertices are labeled by a BFS starting at the sink. Next, the algorithm pushes all possible flow from the source to its adjacent vertices, which results in a preflow. While the preflow is not a flow, a vertex v with highest label is chosen among all vertices having positive excess. If possible, the excess at v is pushed along its incident edges toward the sink. If there is then still some excess left at v, the distance label of v is updated accordingly. The push–relabel algorithm proposed by Goldberg and Tarjan [8] has strongly polynomial