Universidade Federal do Paraná

Departamento de Informática

Jonatan Schroeder
André Pires Guedes
Elias P. Duarte Jr.

Computing the Minimum Cut and Maximum Flow of Undirected Graphs

Relatório Técnico
RT-DINF 003/2004

Curitiba, PR
2004

Computing the Minimum Cut and Maximum Flow of Undirected Graphs

Jonatan Schroeder
André Pires Guedes
Elias P. Duarte Jr.
Federal University of Paraná - Dept. of Informatics
P.O. Box 19081 CEP 81531-990
Curitiba PR Brazil
{jonatan,andre,elias}@inf.ufpr.br

Abstract

This work presents an algorithm for computing the maximum flow and minimum cut of undirected graphs, based on the well-known algorithm presented by Ford and Fulkerson for directed graphs. The new algorithm is equivalent to just applying Ford and Fulkerson algorithm to the directed graph obtained from original graph but with two directed arcs for each edge in the graph, one in each way. We present a proof of correctness and experimental results.

Keywords: Graph Theory, Maximum Flow, Minimum Cut

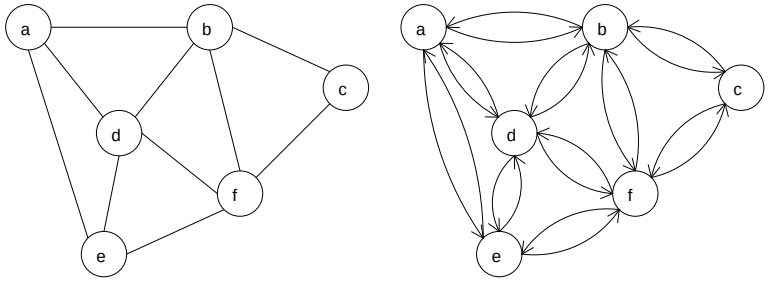
1 Introduction

This work presents an algorithm for computing the maximum flow of undirected graphs. This algorithm differs from those applied to directed graphs (or digraphs), since in undirected graphs an edge can be used in both ways, and if it's used in a way, it cannot be used in the other way. One of the applications of the new algorithm is the computation routes between two hosts in a computer network, as presented by [6].

The algorithm is based in the same concept used by Ford and Fulkerson in [4], which is based on the computation of flow augmenting paths. These paths take an existing flow and construct a new flow that is greater than the original flow.

The new algorithm is equivalent to just applying Ford and Fulkerson algorithm to the directed graph obtained from original graph but with two directed arcs for each edge in the graph, one in each way, as shown in figure 1. This figure contains an example of undirected graph (A) and its corresponding directed graph (B).

The reason that the original algorithm by Ford and Fulkerson can be applied is that, considering the two arcs between a given pair of vertices, when an arc is used in a flow, the other arc cannot be used, otherwise it would not satisfy the skew simmetry, which is described in section 2.



(A) (B)

Figure 1: Example of an undirected graph and its corresponding directed graph.

As the proposed algorithm is based on Ford and Fulkerson's algorithm, we make a brief description of that algorithm in section 2. In section 3 we present the new algorithm considering undirected edge-weighted graphs. In section 4 we present the proof of correctness of the proposed algorithms. In section 5, we briefly describe a Java implementation of the algorithm, and finally section 6 contains our conclusions.

2 A Brief Description of Ford & Fulkerson Algorithm

One of the most well known algorithms for computing the maximum flow of a digraph is the algorithm presented by Ford and Fulkerson [4]. This algorithm uses flow-augmenting paths to increase existing flows in the digraph, so that in each iteration the flow is greater. We give a brief description of the problem and the algorithm below.

2.1 Algorithm Specification

Definition: Given a connected digraph $G = (V, E)$, and a pair of nodes $s, t \in V$, called respectivelly source and sink, let $c : E \to \mathbb{R}^*_+$ be the capacity of the edges in G. A *flow* in G is a function $f : V \times V \to \mathbb{R}$ such that:

$$\forall u, v \in V, f(u, v) \leq c(u, v) \tag{1}$$

$$\forall u, v \in V, f(u, v) = -f(v, u) \tag{2}$$

$$\forall u \in V - \{s, t\}, \sum_{v \in V} f(u, v) = 0 \tag{3}$$

In other words, equation 1, known as the capacity constraint property, indicates that the resources used by a flow in an edge cannot be greater than the capacity of that edge. Equation 2, known as the skew simmetry property, says that the net flow from node u to node v is the negative of the net flow in the reverse direction. Thus, the net flow from a node to itself is 0, since for all nodes, $f(u, u) = -f(u, u)$ implies that $f(u, u) = 0$. Equation 3, known as the flow conservation property, says that the sum of all flows that enter in a node (negative flows) plus the sum of all flows that leave that node (positive flows) is 0.

Definition: Given a connected digraph $G = (V, E)$. Let f be a flow in G. The *value* of the flow f is defined as:

$$|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$$

for each edge $(u, v) \in E$ **do**

$f[u, v] \leftarrow 0$

$f[v, u] \leftarrow 0$

while there exists a path p from s to t with no cycles in the residual network G_f **do**

$\Delta \leftarrow min\{c_f(u, v) : (u, v) \in p\}$

for each edge (u, v) in p **do**

$f[u, v] \leftarrow f[u, v] + \Delta$

$f[v, u] \leftarrow -f[u, v]$

Figure 2: The algorithm proposed by Ford and Fulkerson.

The value of a flow is the total flow that leaves s and, as proved in [2], is equal to the total flow that enters t.

Definition: Given a connected digraph $G = (V, E)$, and considering a pair of nodes $u, v \in V$. Let f be a flow in G. The *residual capacity* of the pair (u, v), or $c_f(u, v)$, is given by:

$$c_f(u, v) = c(u, v) - f(u, v)$$

The residual capacity is used by the algorithm to determine how much flow can pass through a pair of nodes. It is used basically in the definition of the so called residual network, given below.

Definition: Given a connected digraph $G = (V, E)$. Let f be a flow in G. The *residual network* of G induced by f is graph $G_f = (V, E_f)$, where:

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$$

The residual network is an auxiliar graph used by the algorithm descripted below.

Problem: Given a connected digraph $G = (V, E)$, where V is a set of nodes and E is a set of edges between these nodes ($E \subseteq V \times V$), a capacity $c : E \rightarrow \mathbb{R}^*_+$ and two nodes s and t, find a maximum flow (a flow whose value is maximal) from source s to sink t in G. Let f be this flow[1].

Initially, the algorithm presented in figure 2 creates an initial flow that is empty, i.e., there is no flow between any pair of nodes. Next, a randomly chosen path is obtained from the residual network. Since the residual network contains edges only where the residual capacity is positive, the path obtained will be composed by nodes that can increase the flow. In this path, the minimum residual capacity of all pairs corresponds to the amount that can be increased in all node pairs so that the capacity is not overrun. Finally, the flow amount in the edges is increased, and the algorithm searches a new path in the residual network. When no path is found, than the flow cannot be increased, so it is the maximum flow.

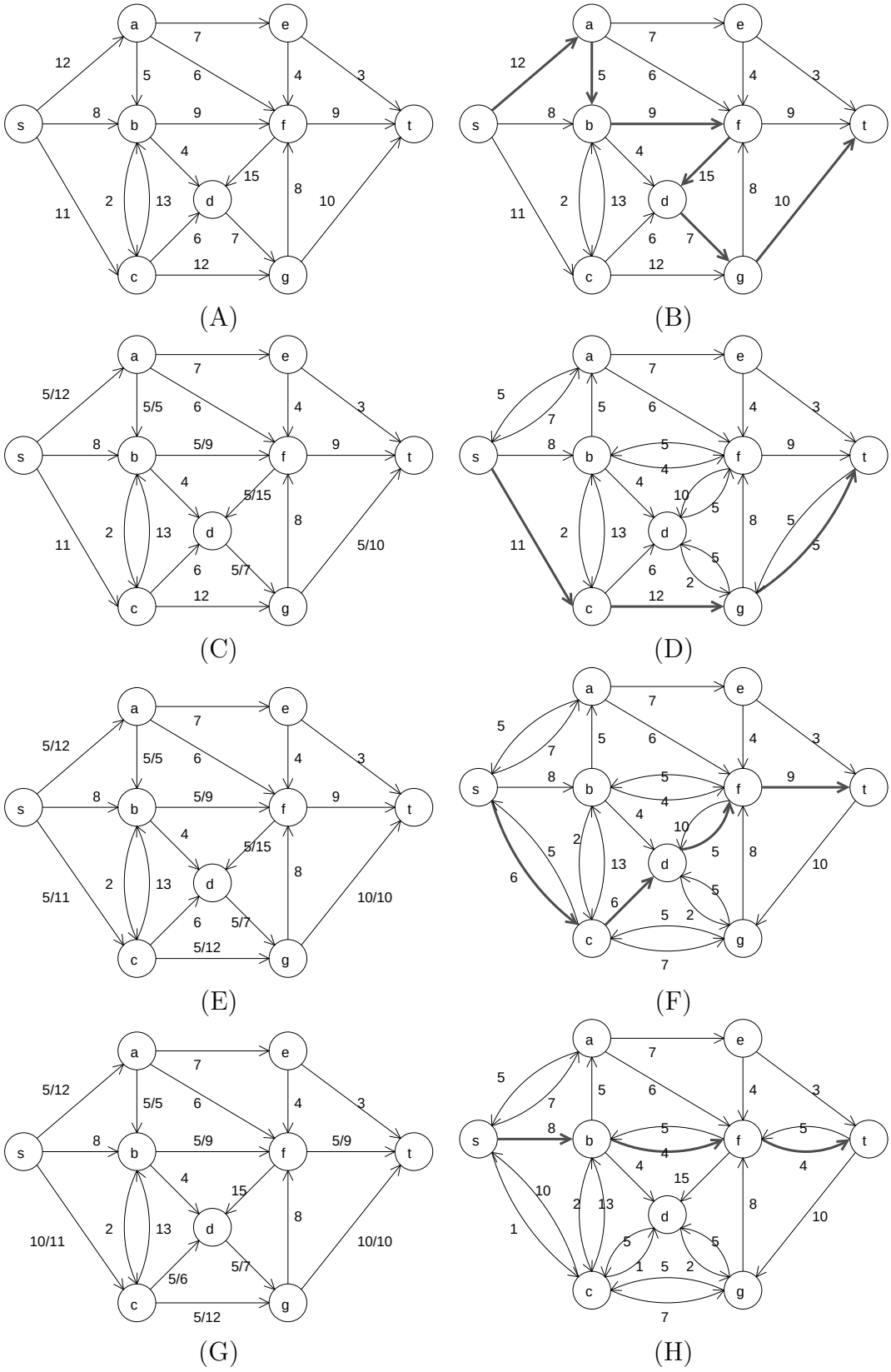


Figure 3: Example execution of the algorithm by Ford and Fulkerson.

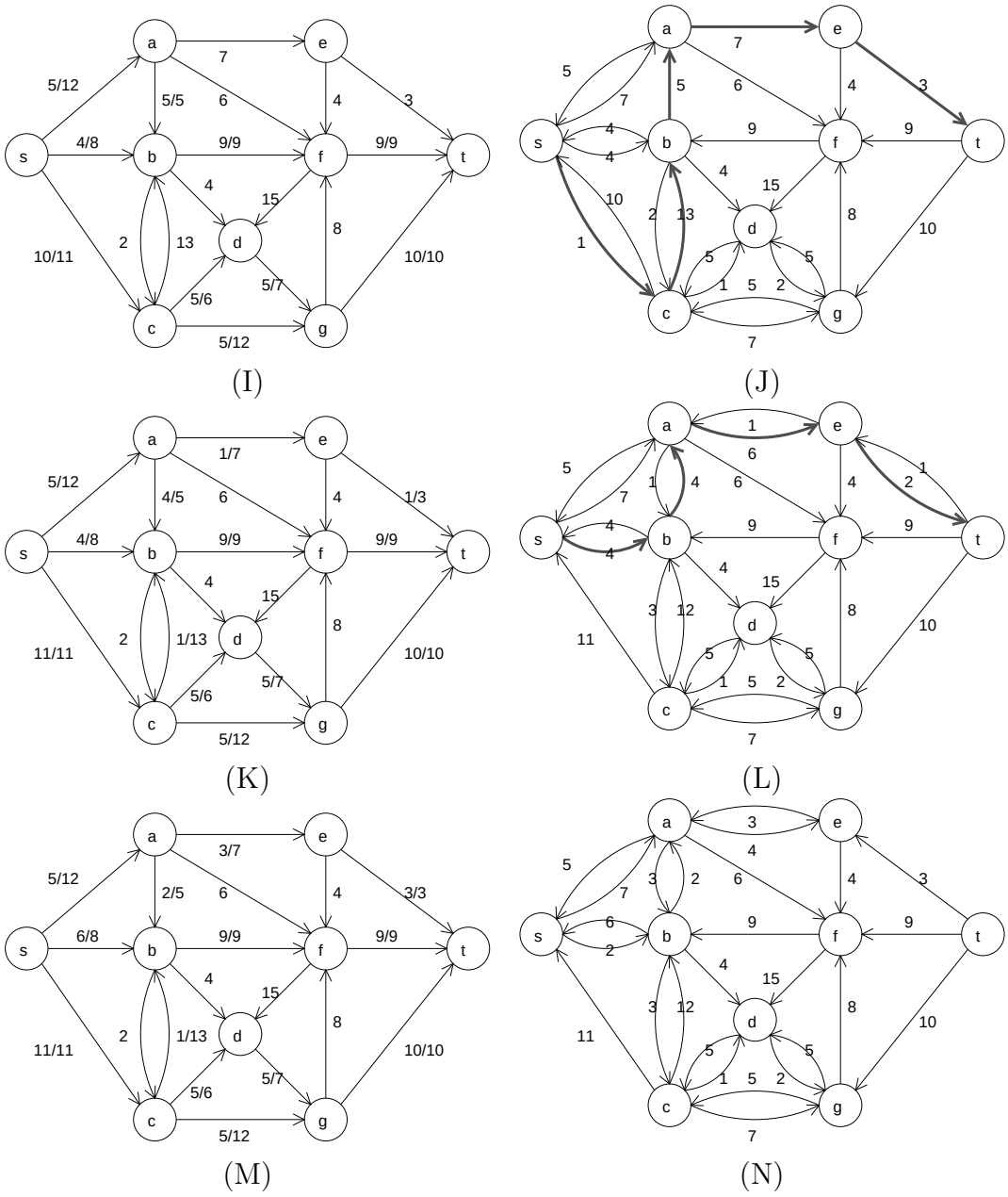


Figure 3: (cont.) Example execution of the algorithm by Ford and Fulkerson.

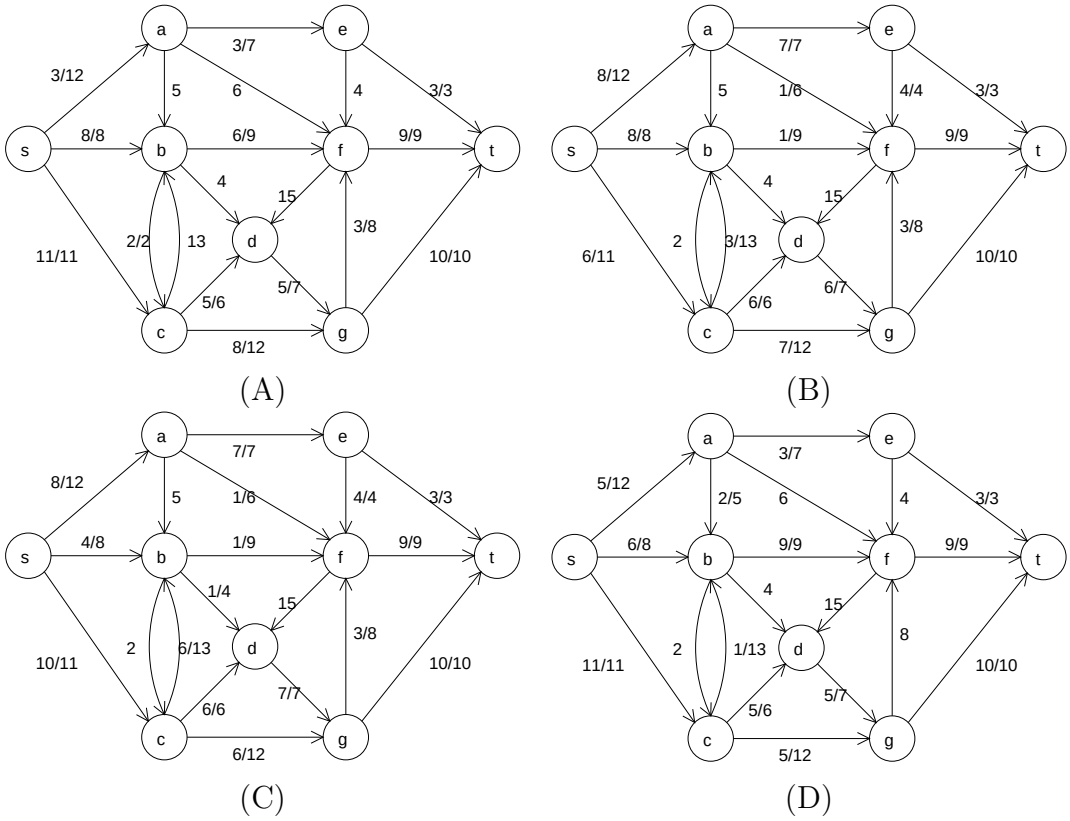


Figure 4: Feasible maximum flows for graph shown in figure 3.

2.2 Example Execution

An example showing the execution of this algorithm is shown in figure 3. This figure contains a sequence of graphs that represent the obtained flow step by step. In these graphs, the labels of the arcs contain, in this order, the flow used in that arc and its capacity. When only one number is found, than there is no flow passing through the arc.

The initial graph is presented in (A), and its corresponding residual network is the graph shown in (B). The first augmenting path found is the path (s, a, b, f, d, g, t), also shown in (B). In this path, the smallest capacity is the capacity of arc (a, b), which is equal to 5. So, the flow used in pairs (s, a), (a, b), (b, f), (f, d), (d, g) and (g, t) increases in 5 units, as shown in (C), and in the new residual network in (D). Next the same procedure is repeated for path (s, c, g, t). The result is shown in (E), and the new residual network in (F).

In the residual network, shown in (F), path (s, c, d, f, t) is found. In this case, the flow used between pair (d, f), equal to -5 (by the skew simmetry), is increased in 5 units, becoming equal to 0. So, the new flow formed by this augmenting path does not use pair (f, d), that had been used before. The new flow is presented in (G), and the corresponding residual network is shown in (H).

The same process runs with path (s, b, f, t), resulting in the flow shown in (I). Its corresponding residual network is shown in (J). Next, path (s, c, b, a, e, t) is considered, resulting in the flow in (K). Note that, in this case, the flow of the pair (c, b) is +1, and, by skew simmetry, the flow of the pair (b, c) is -1. Consequently, the residual capacity $c_f(c, b)$ is equal to

[1]We use brackets when we treat an identifier – such as f – as a mutable field, and we use parenthesis when we treat it as a function.

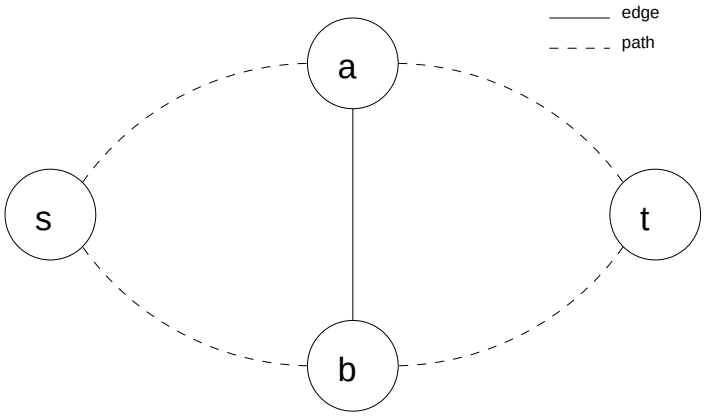


Figure 5: Example of the use of algorithm.

$13-(+1)=12$, and the residual capacity $c_f(b,c)$ is equal to $2-(-1)=3$. These results in the residual network shown in (L).

Finally, path (s,b,a,e,t) is taken, resulting in the flow shown in (M), whose residual network is shown in (N). This residual network contains no path between s and t, so that the flow shown in (M) is the maximum flow.

The maximum flow is not always unique. Some graphs can contain more than one flow that has a maximal value. As an example, for the graph shown in figure 3(A), all flows shown in figure 4 are feasible maximum flows, among other possible for that graph. The result will depend on the augmenting paths used, since there are, in many cases, more than one possible augmenting path for each step of the algorithm.

3 Finding the Maximum Flow of Undirected Graphs

The main concept behind the algorithm is to obtain individual augmenting paths that can be used to increment an existing flow. Augmenting paths are paths from the source to the sink of the flow that increase the existing flow. Using edges not used in the flow, or edges partially used, we can obtain paths that increase the flow. But we can also decrease the used capacity for a used edge, so that it can be re-used in another augmenting path.

Consider as an example the graph shown in figure 5. In order to simplify the example, only a portion of the graph is represented. The dashed edges can contain more nodes not visible in the figure. There is a path between nodes s and a, and between nodes s and b. There is also a path between nodes a and t, and between nodes b and t. On the other hand, there is only an edge between nodes a and b.

Consider initially an empty flow between s and t. We can use the path formed by the path between s and a, the edge between a and b and the path between b and t. This way, we will obtain an augmenting path, since none of the edges in the path is already used in the flow. The flow is then formed uniquely by this path.

Next, we consider the path formed by the path between s and b, the edge between b and a and the path between a and t. The edge between b and a has been already used but in the other direction (from a to b). In the algorithm, this edge is marked as unused, since we can find a flow that combines both paths. In the example, the resulting flow will be composed by the path between s and a, the path between a and t, the path between s and b and the path

between b and t. The edge between a and b can then be used again by the algorithm to compose a new flow augmenting path, considering the nodes and edges not shown in figure.

In other words, consider, as an example, that node s is a water producer and node t is a water consumer. The other nodes of the graph, say a and b, are the connection points, and the edges are the connections. Suppose that initially s sends water to a, who sends it to b, who finally sends it to t. But someone deduces that s can also start sending water to b, and then a stops to send water to b and starts sending to t. With this modification, the volume of water that is sent increases, as water is flowing from s to t both through a and b. The connection between a and b can be used again by another path.

3.1 Algorithm Specification

These definitions are similar to the corresponding definitions shown in section 2 for digraphs, but are adapted to undirected graphs.

Definition: Given a connected undirected graph $G = (V, E)$, and a pair of nodes $s, t \in V$, called respectivelly source and sink, let $c : E \to \mathbb{R}^*_+$ be the capacity of the edges in G. A *flow* in G is a function $f : V \times V \to \mathbb{R}$ such that:

$$\forall u, v \in V, f(u, v) \leq c(\{u, v\}) \tag{4}$$

$$\forall u, v \in V, f(u, v) = -f(v, u) \tag{5}$$

$$\forall u \in V - \{s, t\}, \sum_{v \in V} f(u, v) = 0 \tag{6}$$

Definition: Given a connected undirected graph $G = (V, E)$. Let f be a flow in G. The *value* of the flow f is defined as:

$$|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$$

Definition: Given a connected undirected graph $G = (V, E)$, and considering a pair of nodes $u, v \in V$. Let f be a flow in G. The *residual capacity* of the pair (u, v), or $c_f(u, v)$, is given by:

$$c_f(u, v) = c(\{u, v\}) - f(u, v)$$

Definition: Given a connected undirected graph $G = (V, E)$. Let f be a flow in G. The *residual network* of G induced by f is digraph $G_f = (V, E_f)$, where:

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$$

The residual network is always directed, either for directed or undirected graphs.

Problem: Given a connected undirected graph $G = (V, E)$, where V is a set of nodes and E is a set of edges between these nodes ($E \subseteq \{\{u, v\} : u, v \in V\}$), a capacity $c : E \to \mathbb{R}^*_+$ and two nodes s and t, find a maximum flow (a flow whose value is maximal) from source s to sink t in G. Let f be this flow.

The algorithm presented in figure 6 is similar to the algorithm by Ford and Fulkerson, but considering undirected graphs, i.e., the order of nodes connected by an edge does not matter, as well as the associated capacity.

for each edge $\{u, v\} \in E$ **do**

$f[u, v] \leftarrow 0$

$f[v, u] \leftarrow 0$

while there exists a path p from s to t with no cycles in the residual network G_f **do**

$\Delta \leftarrow min\{c_f(u, v) : (u, v) \in p\}$

for each edge (u, v) in p **do**

$f[u, v] \leftarrow f[u, v] + \Delta$

$f[v, u] \leftarrow -f[u, v]$

Figure 6: The proposed algorithm.

3.2 Example Execution

An example showing the execution of this algorithm in shown in figure 7. In this figure, the representation of the flow is the same used in directed graphs, but there is also a half head showing the direction of the flow.

The execution is based on the original graph shown in (A). The residual network of the original graph is shown in (B). The first path that is found is path (s, b, d, c, g, f, t), also shown in (B). In this path, the minimum capacity is the capacity of edge (b, d), that is equal to 3. The use of the path as an augmenting path results in the flow shown in (C). The corresponding residual network is shown in (D).

The next augmenting path considered is path (s, c, d, f, e, t), also shown in (D). The minimum capacity in this case is equal to 6, from edge (d, f). The new flow is shown in (E). Note the edge (c, d), that had been already used in previous path, but in the other direction. In this node, 3 units are removed from the direction $(d \rightarrow c)$, so that the node is not more used in the flow. The remaining 3 units are than used in the direction $(c \rightarrow d)$, and the node is now used in a new direction. The new residual network is shown in (F).

The execution proceeds with path (s, b, f, d, c, g, t), resulting in the flow shown in (G) and in the residual network shown in (H). Note that, in this step, the process for node (c, d) is repeated in the other direction. In the sequence, the path (s, b, a, e, f, g, t) is taken, resulting in the flow shown in (I) and in the residual network shown in (J). Next, the path (s, c, d, f, t) is considered, resulting in the flow shown in (K) and in the residual network shown in (L). The next path that is considered is path (s, c, d, g, f, t), and the new flow is shown in (M), with the residual network shown in (N). In the sequence, path (s, b, f, t) is considered, resulting in flow shown in (O), whose residual network is shown in (P). Finally, The path (s, a, f, t) is the last path found, resulting in the final flow shown in (Q). As shown in the residual network in (R), there is no feasible augmenting path, so the found flow is a maximum flow.

4 Proofs

To show that this algorithm constructs a maximum flow from s to t, we must show that:

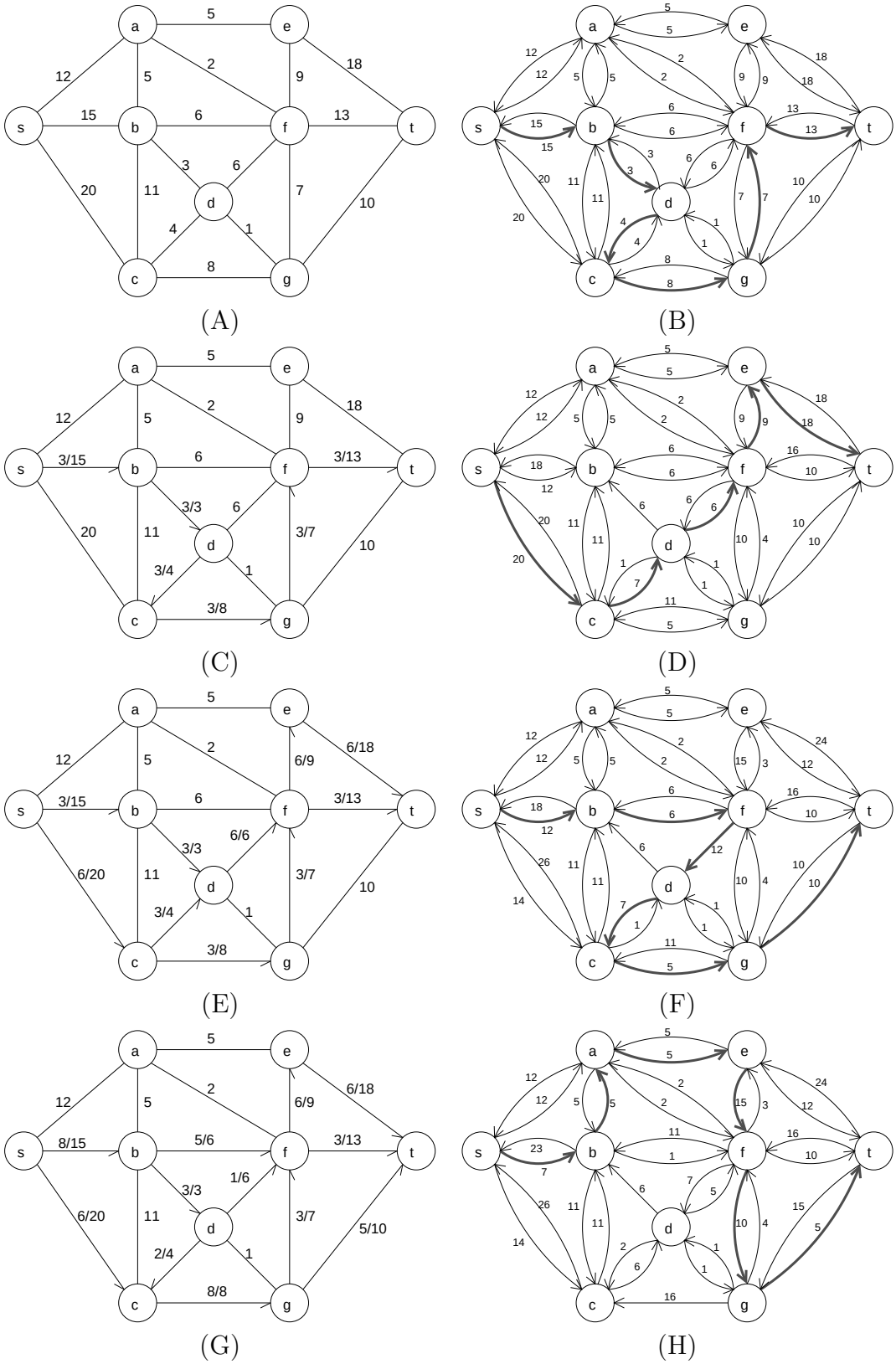


Figure 7: Example execution of the algorithm proposed by this paper.

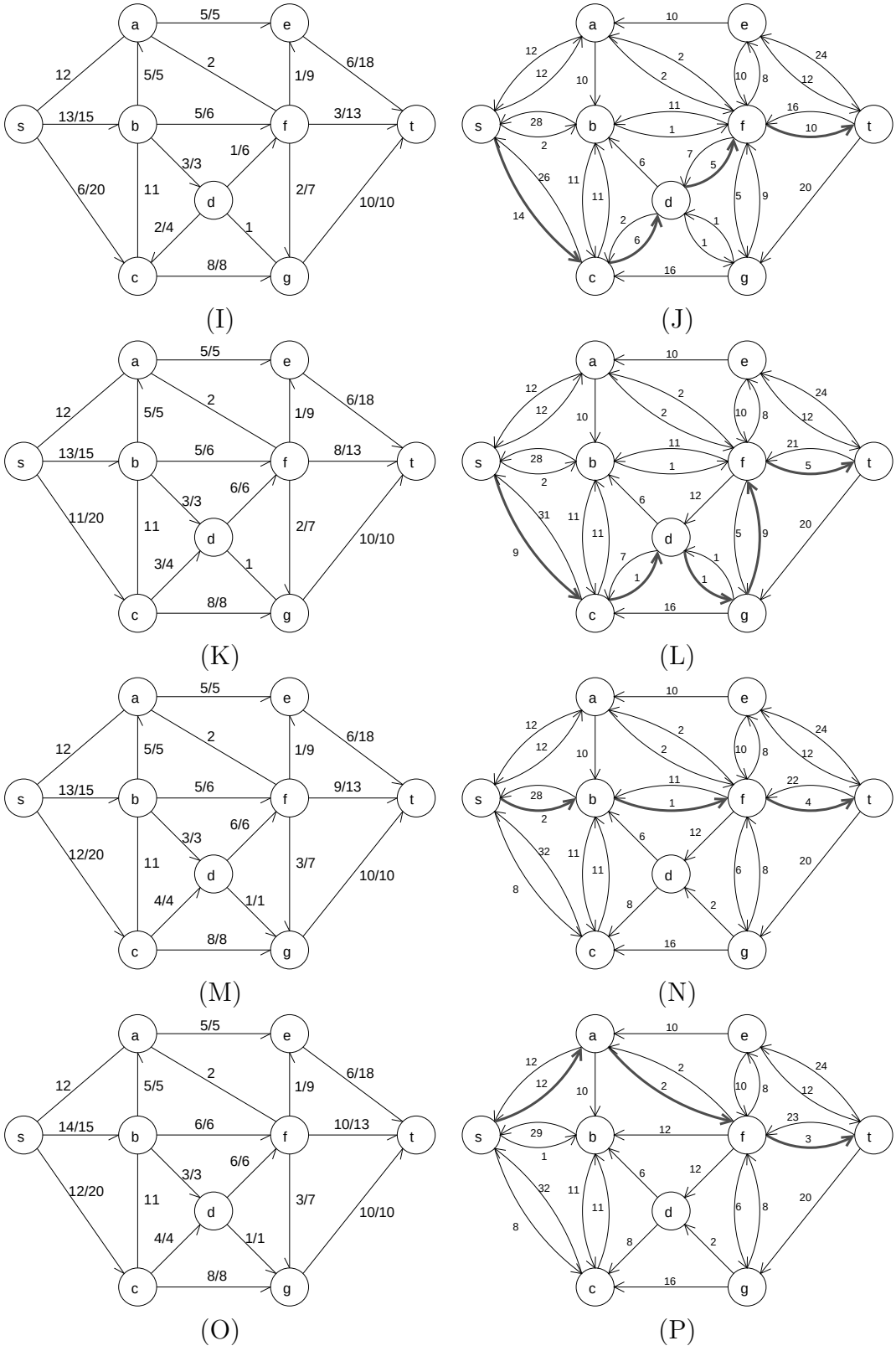
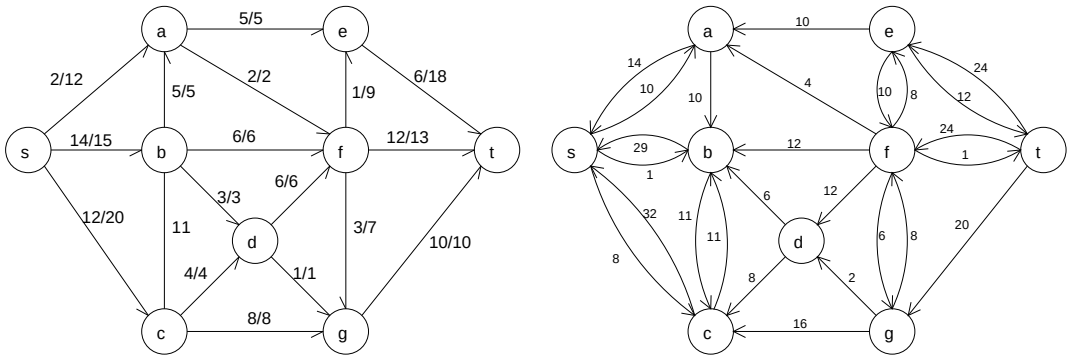


Figure 7: (cont.) Example execution of the algorithm proposed by this paper.



(Q) (R)

Figure 7: (cont.) Example execution of the algorithm proposed by this paper.

1. The algorithm constructs a flow;
2. This flow is a maximum flow;
3. The algorithm terminates after a finite number of steps.

The proofs of each one of these items will be discussed in this section.

4.1 The Algorithm Constructs a Flow

As presented in section 3.1, a flow must satisfy some properties. The proof consists in demonstrating each one of the properties in the resulting flow. In each iteration of the algorithm, the result is a flow, so the final result is also a flow. In the following, we call f_i the flow in step i.

Capacity constraint property: For nodes that compound the augmenting path, we have $f_i(u,v) = f_{i-1}(u,v)+\Delta$. But Δ is the minimum value of all residual capacities in the augmenting path. So, $\Delta \leq c_{f_{i-1}}(u,v) = c(\{u,v\}) - f_{i-1}(u,v)$. Applying this property to the equation above, we have $f_i(u,v) \leq f_{i-1}(u,v) + c(\{u,v\}) - f_{i-1}(u,v)$, and $f_i(u,v) \leq c(\{u,v\})$, satisfying the capacity constraint.

For nodes in the opposite direction, we have $f_i(u,v) = -f_i(v,u) = -\big(f_{i-1}(v,u) + \Delta\big) = f_{i-1}(u,v) - \Delta$. As $\Delta > 0$ (by the definition of residual network), $f_i(u,v) < f_{i-1}(u,v) \leq c(\{u,v\})$, satisfying the capacity constraint.

For the other nodes, the flow is not modified, so $f_i(u,v) = f_{i-1}(u,v) \leq c(\{u,v\})$, satisfying the capacity constraint.

Skew simmetry property: In all cases in which the flow is updated, the flow in the reverse direction is also updated, and always with the opposite value. Since the augmenting path has no cycles, each pair is updated only once in each step, so the skew simmetry will always be mantained.

Flow conservation property: The augmenting path can be defined as $(a_0, a_1, a_2, \ldots a_n)$, with $a_0 = s$, $a_n = t$ and $a_j \in V - \{s,t\}$ for each $1 \leq j \leq n-1$. Consider Δ as the minimum capacity of this augmenting path. For each node of the path, except for the source and the sink, the

sum of all incident flows of node a_j is:

$$\begin{aligned}
\sum_{v \in V} f_i(a_j, v) &= f_i(a_j, a_{j+1}) + f_i(a_j, a_{j-1}) + \sum_{v \in V - \{a_{j+1}, a_{j-1}\}} f_i(a_j, v) \\
&= \big(f_{i-1}(a_j, a_{j+1}) + \Delta\big) + \big(f_{i-1}(a_j, a_{j-1}) - \Delta\big) + \sum_{v \in V - \{a_{j+1}, a_{j-1}\}} f_{i-1}(a_j, v) \\
&= f_{i-1}(a_j, a_{j+1}) + f_{i-1}(a_j, a_{j-1}) + \sum_{v \in V - \{a_{j+1}, a_{j-1}\}} f_{i-1}(a_j, v) \\
&= \sum_{v \in V} f_{i-1}(a_j, v) \\
&= 0
\end{aligned}$$

resulting that the flow conservation property is preserved.

4.2 The Flow is a Maximum Flow

The following proof is based on a similar proof presented in [3].

Definition: Given a connected graph $G = (V, E)$, directed or undirected, and a pair of nodes $s, t \in V$. A $(s - t)$ *cut* is a partition of V into S and $T = V - S$, such that $s \in S$ and $t \in T$. The *capacity* of the cut, represented as $c(S, T)$, is given by:

$$\sum_{u \in S} \sum_{v \in T} c(u, v)$$

Definition: A *minimum cut* is a cut whose capacity is minimal.

Consider a cut (S, T) in graph $G = (V, E)$. Any flow in this graph must be smaller than or equal to the capacity of this cut, since every flow unit must cross the cut. Since this predication is valid for all cuts and for all flows, we deduce that the value of the maximum flow is never greater than the capacity of the minimum cut.

In order to demonstrate that the maximum-flow algorithm has indeed produced a maximum flow, we need only produce a cut separating s from t whose capacity equals the value of the terminal flow produced by the algorithm. This is done as follows.

The algorithm terminates when no augmenting path can be found. After the final application of the algorithm, there is a set of nodes that is reachable from s in the resulting residual network. Let R be this set of nodes, and let R' be the set of the unreachable nodes. Clearly, $R' = V - R$, so pair (R, R') is a cut.

Upon termination of the algorithm, each edge formed by a given node $u \in R$ and a given node $v \in R'$ carries a flow equal to its capacity, in the direction $u \rightarrow v$; otherwise the residual capacity of the pair of nodes would be positive, and v would be reachable in the residual network. Clearly, every flow unit must traverse an arc of this cut at least once. Since all these edges carry the flow in the direction $u \rightarrow v$, there is no flow unit being carried from R' to R, and no flow unit can traverse this cut more than once. Thus, the total flow from s to t equals the capacity of this cut, because every edge from the source side to the sink side carries a flow equal to its capacity.

4.3 The Algorithm Terminates After a Finite Number of Steps

Let's initially accept that all edge capacities are integers. In this case, in each step of the algorithm, the flow is increased at least by one unit. Since the graph is a finite graph, the maximum flow is also finite, so the algorithm will take no more steps than the maximum flow. A similar idea can be used to prove that the algorithm is finite for rational numbers.

For irrational numbers, it's proved that, in some cases, the algorithm will not finish. Further documentation about this cases can be found in [4]. This work is restricted to cases where edge capacities are rational.

5 Implementation

The algorithm has been implemented, specifically for undirected graphs with no capacity (i.e., unitary capacity). The implementation is used specifically to find the minimum cut, that is a related concept, as presented in section 4.2. The implementation is developed in Java language [8], using the library JDigraph, available in [7]. The algorithm is used as a part of the implementation of a connectivity simulator, and the complete source code is available in [6].

6 Conclusion

In this paper we present an algorithm for computing the maximum flow of undirected graphs based on augmenting paths. We also present the proof of correctness in terms of that the algorithm construct a flow, this flow is a maximum flow and the algorithm terminates after a finite number of steps.

Future work will use the proposed algorithm for applications related to edge connectivity in computer networks, specifically problems related to network routing.

References

[1] J. COHEN, E. P. DUARTE JR., *Fault-Tolerant Routing Based on Network Connectivity,* IEEE DRCN, Budapest, 2001.

[2] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, *Introduction to Algorithms,* The Massachusetts Institute of Technology Press, 1990.

[3] J. EVANS, E. MINIEKA, *Optimization Algorithms for Networks and Graphs,* 2nd edition, Dekker, Monticello, 1992.

[4] L. R. FORD JR., D. R. FULKERSON, *Flows in Networks,* Princeton University Press, 1962.

[5] D. R. KARGER, M. S. LEVINE, *Finding Maximum Flows in Undirected Graphs Seems Easier than Bipartite Matching,* Proc. 30th Annual ACM Symp. on Theory of Computing, 69-78, 1998.

[6] J. SCHROEDER, P. R. TORRES JR., N. P. MOLINA JR., *Busca de desvios em redes baseada nos critérios de conectividade #C(v) e MCC(v),* Trabalho de

conclusão de curso, Bacharelado em Ciência da Computação, Universidade Federal do Paraná, 2003.

[7] D. WALEND, *JDigraph,* http://jdigraph.sourceforge.net, accessed in Feb'2004.

[8] *Java Technology,* http://java.sun.com, accessed in Feb'2004.