Pesquisa Operacional (2017) 37(3): 437-453
© 2017 Brazilian Operations Research Society
Printed version ISSN 0101-7438 / Online version ISSN 1678-5142
www.scielo.br/pope
doi: 10.1590/0101-7438.2017.037.03.0437

COMPUTING MAX FLOWS THROUGH CUT NODES

João Paulo de Freitas Araujo[1*], Fernanda Maria Pereira Raupp[2], José Eugenio Leal[1] and Madiagne Diallo[1]

Received November 15, 2016 / Accepted October 23, 2017

ABSTRACT. In this work, the presence of cut nodes in a network is exploited to propose a competitive method for the multi-terminal maximum flow problem. The main idea of the method is based on the relation between cut-trees and cut nodes, which is observed in the context of sensitivity analysis on the variation of edges capacities. Computational experiments were conducted with the proposed algorithm, whose results were compared with the ones of Gusfield, for randomly generated and well-known instances of the literature. The numerical results demonstrate the potential of the method for some classes of instances. Moreover, the proposed method was adapted for the single maximum flow problem, but failed to improve existing running times for the very same classes of instances.

Keywords: maximum flow, cut-tree, cut nodes.

1 INTRODUCTION

Computing the maximum flow value between a source and a terminal node of a given network is a classical problem in the context of network flows. Its extension, namely the *multi-terminal maximum flow problem*, consists of finding the maximum flow values between all pairs of nodes of an undirected network. These problems have several applications, especially in the field of logistics, biology, telecommunications and energy, see for example, Cohen & Duarte Jr. (2001), Tuncbag et al. (2010) and Diallo (2011).

It is noteworthy that the multi-terminal maximum flow problem differs from the multi-commodity flow problem. While, in the latter problem, mixed flows between multiple pairs of origins and destinations share the network, in the former, although we compute the maximum flow values for all pairs of the network nodes, there is a single flow between a source node and a destination node in the network at a time.

*Corresponding author.
[1] Departamento de Engenharia Industrial, PUC-Rio, Rua Marquês de São Vicente, 225, Gávea, 22451-900 Rio de Janeiro, RJ, Brasil. E-mails: johnpa@terra.com.br; jel@puc-rio.br; madiagne.diallo@gmail.com
[2] Laboratório Nacional de Computação Científica, Av. Getúlio Vargas, 333, Quitandinha, 25651-075 Petrópolis, RJ, Brasil. E-mail: fernanda@lncc.br

Ford & Fulkerson (1973) popularized the maximum flow problem in the '50s. Through the demonstration of the connection between a maximum flow value and a minimum cut capacity, they sophisticatedly solved it. Since then, many improved algorithms have been published to compute the maximum flow values or the minimum cut capacities, including the preflow-push algorithm from Goldberg & Tarjan (1986), which we will use in this work.

Regarding the multi-terminal maximum flow problem of a given undirected network G with n nodes, one can solve it naively, by running $n(n-1)/2$ times a maximum flow algorithm between all unordered pairs of nodes of G. However, Gomory & Hu (1961) developed a method to compute the maximum flow values of G, by just running $(n - 1)$ times a maximum flow algorithm. Its output, called cut-tree, summarizes the maximum flow values and identifies a minimum cut between any pair of nodes. After that, Gusfield (1990) presented a simpler procedure to obtain the same cut-tree, but also using $(n - 1)$ times the maximum flow algorithm. Then, Goldberg & Tsioutsiouliklis (2001) conducted a study comparing computationally three variations of the Gomory and Hu algorithm and the Gusfield algorithm. For the unweighted case, when each edge has one-unit capacity, Bhalgat et al. (2007) showed a faster algorithm that does not use a maximum flow algorithm as internal procedure.

Elmaghraby (1964) introduced the sensitivity analysis on multi-terminal flows, studying the effects on the maximum flow of a network through the variation of a single (parametric) edge capacity. Later, Barth et al. (2006) extended that study to the case of more than one parametric edge, noting that a total of 2^k cut-trees is sufficient to compute all maximum flows for any parameter value, being k the number of parametric edges in the network.

In this work, given an undirected network G, by using the theory of sensitivity analysis on multi-terminal network flows under edge capacity variations, we introduce a theoretical property that relates cut nodes and cut-trees. Based on such a property, we propose a new approach for the computation of cut-trees for networks that contain cut nodes. Similarly, a single maximum flow can be computed in parts, if there are cut nodes in the path between the source and the terminal node.

Computational experiments are conducted to compare the running times of the proposed procedures with respect to the traditional ones. For these experiments, four instances families are used: PATH and TREE, from Goldberg & Tsioutsiouliklis (2001), CACTUS, from Husimi (1950), and PARTED that we specially developed for the experiments. When a given undirected network contains cut nodes, the numerical results show that the computation of cut-trees with the proposed method is effective. However, for the same test instances it seems not the case, when we apply an adaptation of the method to solve particularly the single maximum flow problem.

The paper is structured as follows. Section 2 presents the used notations and basic concepts related to cut nodes and to the multi-terminal maximum flow problem. Section 3 presents some theoretical properties of cut nodes in network flows. In Section 4, we present the proposed method based on the identification of cut nodes, together with an example to show its application. In addition, an adaptation of the proposed method is showed to solve the maximum flow problem. Computational experiments with the proposed algorithms are shown in Section 5, while concluding remarks and perspectives are resumed in Section 6.

2 NOTATION AND BASIC CONCEPTS

From now on, we assume that the reader has basic knowledge of graph and network flows theories and problems. Reference books include Ford & Fulkerson (1973), and Hu & Shing (2002).

Let $G = (V, E)$ be a connected undirected graph, consisting of a set V of n nodes v, and a set E of m edges e, where each edge is an unordered pair $[i, j]$ of nodes in V. A *network* is a graph G associated with a *capacity* function over the edges $c : E \rightarrow R_+$. A *flow* from a source node s to a terminal node t in G is a function $f : E \rightarrow R_+$ with the conservation property at each node v, except for s and t

$$\sum_{i \in V} f(i, v) = \sum_{j \in V} f(v, j) \quad \forall v \in V \backslash \{s, t\},$$

and the capacity constraint

$$\forall i, j \in V, f(i, j) \leq c(i, j).$$

Observe that edges can be represented by two arcs of opposite directions. Therefore, unlike an arc, an edge has no direction, that is, it has two opposite directions at once. Thus, an undirected network has the same structure as a directed symmetric network, where each arc has the same capacity of the original edge, allowing equal capacity to either direction. It is important to note that, when a flow passes through a capacitated edge, it uses only one arc, never both. Hereafter, we will deal only with undirected networks.

We denote by $(s$ - $t)$ the cut separating the nodes s and t, by $c(s$ - $t)$ the capacity of the cut (cut value), and by $(X, \overline{X})$ a cut separating the nodes of a graph into two complementary subsets X and $\overline{X}$. Among all possible cuts separating s and t, one with the smallest capacity is called a *minimum cut*, and its capacity is the maximum flow value between s and t, which will be represented by $f_{s,t}$.

After observing the existence of at most $n - 1$ distinct values of maximum flow in an undirected network with n nodes, Gomory & Hu (1961) developed a method that obtains the $n(n - 1)/2$ values of maximum flows using a node contraction scheme and running only $n - 1$ times the maximum flow algorithm. The result is expressed by a cut-tree defined as follows.

Definition 2.1. *A* ***cut-tree*** *of a network $G = (V, E)$ is a tree $CT = (V, E')$ obtained from G, with weighted edges and the same set of nodes V. A cut-tree CT has the following properties:*

1. *Equivalent flow tree: the value of the maximum flow between any s and t of G is equal to the value of the maximum flow in CT between s and t, that is, the smallest edge capacity on the unique path connecting s to t in CT. Thus, the maximum flows between all pairs of nodes in G are represented in CT;*

2. *Cut property: the removal of any edge e with capacity $c(e)$ from CT separates its nodes into two sets. This partition corresponds to a minimum cut in G with capacity $c(e)$, separating the two extremities of the edge e.*

Figure 1 illustrates an example of a cut-tree CT constructed from a network G. As we can see, by the properties mentioned above, the minimum cut between nodes 2 and 3, and nodes 1 and 2, are respectively reflected by the edges [2, 3] and [1, 2] of the cut-tree CT. Its maximum flow values are the capacities of these edges, in this case 3 and 4, respectively. The minimum cut between nodes 1 and 3 is reflected, in turn, by the edge [2, 3] in CT, with maximum flow value equal to 3.

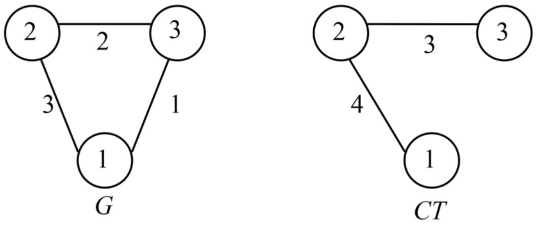

Figure 1 – Example of a cut-tree CT of a network G.

In general, there are several cut-trees for the same network. The cut-tree will be unique only if all the (s - t) minimum cuts of the network are unique.

Gusfield (1990) presented a very simple procedure that builds a cut-tree without using contraction of nodes. Its implementation is very easy: it takes only five additional lines of code to any algorithm that computes a minimum cut. Like the method of Gomory and Hu, Gusfield's method solves the multi-terminal maximum flow problem with $(n - 1)$ executions of a maximum flow algorithm.

Definition 2.2. *A node i of a connected graph G is called a* ***cut node*** *when $G\backslash\{i\}$ is not connected.*

As for illustration, nodes 3 and 6 are cut nodes of the graph showed in Figure 2(a), as we can verify respectively in Figures 2(b) and 2(c).

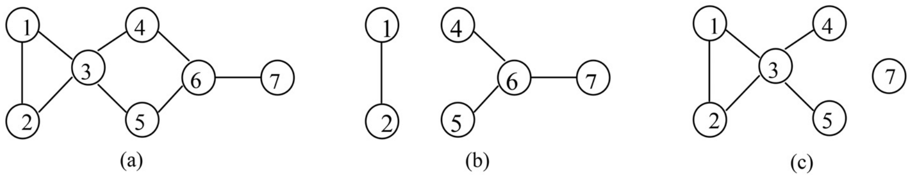

Figure 2 – Illustration of cut nodes.

Definition 2.3. *A* ***biconnected component*** *of a graph G is a maximal connected subgraph B of G containing no cut nodes, where the term "maximal" refers to the state that any inclusion of a node in B creates a cut node in B.*

Now, see an example of biconnected components of a given graph in Figure 3.

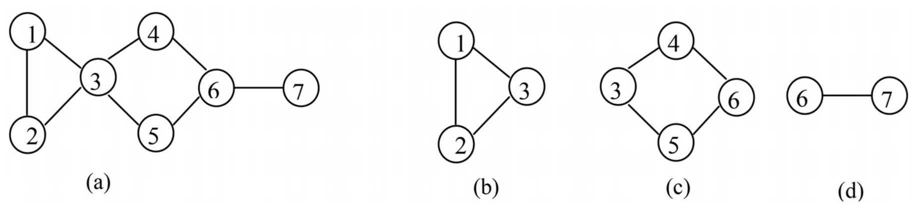

Figure 3 – The graph (a) and its three biconnected components (b), (c) and (d).

3 PROPERTIES OF CUT NODES

In this section, we introduce some theoretical results that allow an innovative computation of both the single and the multi-terminal maximum flow problems, by exploiting the presence of cut nodes in networks. For the case of the multi-terminal maximum flow problem, the property comes up from its parametric version, formulated in the light of the theory of sensitivity analysis on multi-terminal maximum flows. In general terms, this theory studies the behavior of the maximum flows values between the all pairs of nodes in a network under variations of edge capacities.

Barth et al. (2006) examined the parametric problem considering the increasing variation of the capacity of a single edge. The problem can be formulated as follows for a pair of nodes.

Let $G = (V, E)$ be a network with source node s and terminal node t. Consider an edge $e = [i, j] \in E$ with non-negative capacity $c(e) = \lambda$. The goal is to determine the maximum flow value between s and t with the increasing variation of λ.

The cited authors observed that, for a network that has only one edge with parametric capacity, the variation of this capacity may not influence the values of maximum flows (and minimum cuts) for various pairs of nodes. Denoting by $f^{\lambda}_{s,t}$ the maximum flow value between s and t when the capacity of the edge e is λ, they stated the following result.

Lemma 1 (Barth et al. (2006)). *Let $G = (V, E)$ be a network with n nodes, and $e = [i, j] \in E$ such that $c(e) = \lambda$. Let s and t be a pair of nodes of G and CT^{α} a cut-tree when $c(e) = \alpha$. If the path connecting s to t $P_{s,t}$ in CT^{α} has no edge in common with $P_{i,j}$, then $f^{\lambda}_{s,t} = f^{\alpha}_{s,t}$, $\forall \lambda > \alpha \geq 0$.*

Proof. Using the cut property of the cut-trees (Definition 2.1, item 2), one can show that there exists a minimum cut $C^{\alpha}_{s,t}$ separating s and t where both nodes i and j ($e = [i, j]$) are in the same side of the minimum cut. Therefore, the cut does not contain e for $\lambda > \alpha$, and it is insensible to the variation of λ. $\square$

Still, for the next result of Barth et al. (2006), the following definition is necessary.

Definition 3.1. *Let $G = (V, E)$ be a connected and acyclic network, i.e., a tree. Let i and j be a pair of nodes of G and $P_{i,j}$ the (unique) path connecting them. The* **(i, j)** ***forest decomposition*** *of G, denoted by $F_{i,j}$, is the set of trees that remains after the removal of $P_{i,j}$.*

For example, consider the tree G and the path i-3-j in G on the left of Figure 4. Then, the (i, j) forest decomposition of G is formed by the four subtrees on the right of Figure 4.

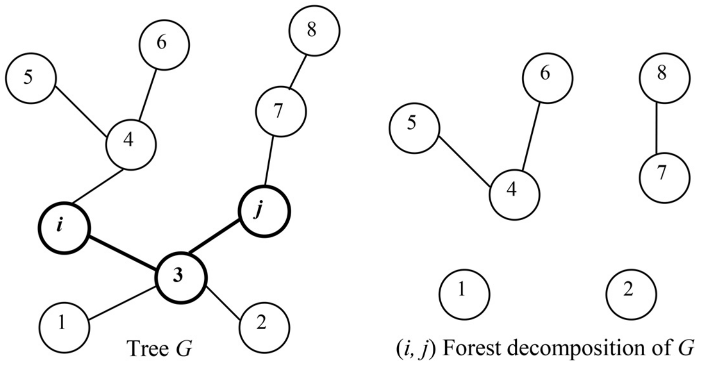

Figure 4 – Example of a (i, j) forest decomposition of a given tree G.

Lemma 2 (Barth et al. (2006)). *Let G be a network with an edge $e = [i, j]$ with parametric capacity. Let CT^α be the cut-tree when $c(e) = \alpha$. Let $F_{i,j}$ be the (i, j) forest decomposition of CT^α. For each tree $T \in F_{i,j}$, there exists a cut-tree of G with $c(e) = \lambda > \alpha$ that contains T as subtree.*

Proof. To see this proof, please, refer to the work of Barth et al. (2006). □

Now, we introduce the following results.

Lemma 3. *If two nodes belong to a biconnected component A, the maximum flow between them can be computed considering A as a graph itself.*

Proof. If s and t are nodes of G in A, there is no path between s and t that contains a node not in A. □

Before we introduce the next result, it is important to observe that:

- The affirmation that a given edge is not contained in a network is equivalent to say that this edge is contained in the network with null capacity. Therefore, adding an edge to a network can be understood as varying positively its capacity from zero.

Lemma 4. *The cut-tree of a graph is the union of the cut-trees of its biconnected components.*

Proof. Let G be a graph with a unique biconnected component A, which is the graph itself. Let CT and CT_A be the cut-trees of G and A, respectively. By adding edge after edge, and the nodes of its extremities, in G, we can create a biconnected component B that shares node z with A. In this process, let $e = [i, j]$ be the edge added at each step and $P_{i,j}$ the path between i

and j in CT. Since, at every step, $P_{i,j}$ doesn't have edge in common with CT_A, according to Lemmas 1 and 2, the cuts in CT_A are not influenced by the process and they can be part of the final CT. To conclude the proof, from Lemma 3, the nodes of A reside on the same side as z in all minimum cuts between nodes from B, which leads to the result that CT_A can be a subtree of the final CT. $\square$

Next we show an example of the proof of Lemma 4. Consider the network G and its cut-tree CT showed in Figure 5.

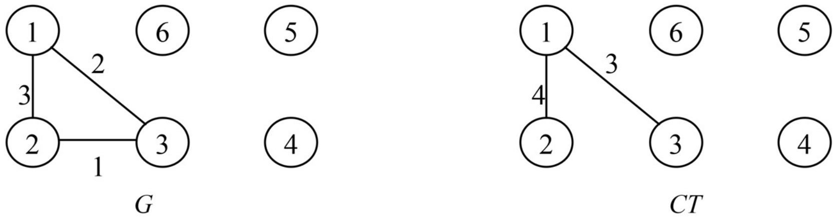

Figure 5 – Network G (left) and its cut-tree CT (right).

Based on Lemma 1, if we add to the network G edges [3, 6], [3, 5], [3, 4], [4, 5] and [5, 6], one by one, these additions would not influence the cuts in CT represented by the edges [1, 2] and [1, 3]. Furthermore, the new CT, according to Lemma 2, could contain these cuts. At last, from Lemma 3, node 3 will be adjacent to node 1 in the new CT. Figure 6 illustrates the new network and its corresponding CT.

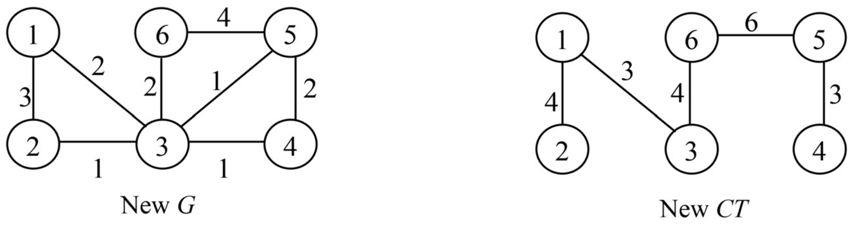

Figure 6 – Network New G (left) and its cut-tree New CT (right).

For the case of determining the single maximum flow between a source and a terminal node, we state that:

Lemma 5. *Let s and t be a pair of nodes of a network G. If a path P between s and t traverses x cut nodes of G, then the maximum flow between s and t is the minimum value among the maximum flows between $(s, z_1), (z_1, z_2), \dots, (z_x, t)$, where $z_1, z_2, \dots, z_x$ are the cut nodes of G in P in the order they are traversed from s to t.*

Proof. The proof is simple. If there are cut nodes of G in P, the removal of z_1 disconnects s from t, and so all the flow that leaves s must pass through z_1. The same result is true for $z_2, \dots, z_x$. $\square$

4 PROPOSED METHODS

Based on Lemma 4, if a network has cut nodes, one can solve the multi-terminal maximum flow problem by computing the cut-trees of each of its biconnected components and joining them at the end. Next, we describe the outline of the proposed method, namely CN.

For computing the maximum flow between all pairs of nodes in an undirected network $G = (V, E)$ with n nodes and capacities on the edges, CN identifies the biconnected components and performs a test. If no biconnected component has more than 80% of n nodes, then the method applies the Gusfield algorithm on each biconnected component. Finally, the cut-tree of G is achieved by joining all the cut-trees of the biconnected components. Otherwise, if there is a biconnected component with more than 80% of n nodes, it applies Gusfield algorithm to G.

Observe that the if-then condition is different from just having a cut node. It avoids the situation shown in Figure 7, where a biconnected component has almost the size of the network. In this situation, it becomes difficult to compensate the overhead of managing biconnected components with computations of maximum flow in networks considerably smaller than the original. The choice for the percentage of 80% will be discussed in Section 5. Regarding the if-else condition, note that a network without cut nodes has only one biconnected component, that is, the network itself. Figure 8 shows the pseudo-code of the method CN.

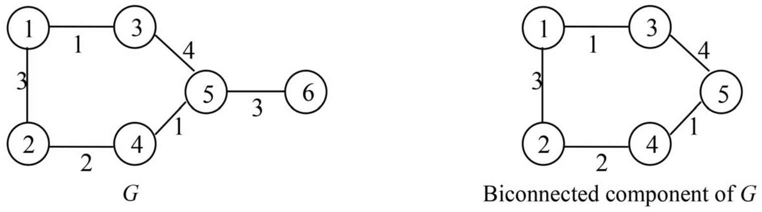

Figure 7 – Network G (left) and its biconnected component with more than 80% of the nodes (right).

```
Input: $G$
1  Identify all biconnected components in $G$;
2  if no biconnected component has more than 0.8n nodes then
3  |  Apply Gusfield algorithm to each biconnected component separately;
4  |  Join all the cut-trees of the biconnected components into a unique cut-tree $CT$;
5  else
6  |  Apply Gusfield algorithm to $G$ to get $CT$;
7  return $CT$;
```

Figure 8 – Pseudo-code of CN.

In our implementation of CN we used the Gusfield algorithm to construct the cut-trees duo to its simplicity, but one can also implement it using the Gomory and Hu algorithm. In addition, the procedure used to identify the biconnected components in the network was based on the algorithm of Hopcroft & Tarjan (1973). Moreover, the highest-label preflow-push algorithm of Goldberg & Tarjan (1986) was chosen to compute the maximum flows inside the Gusfield algorithm.

Given the network instance showed in Figure 9, in the following we illustrate the application of CN.

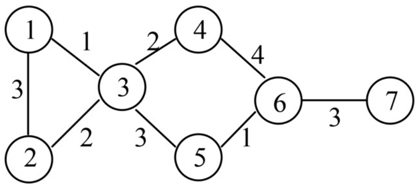

Figure 9 – Network *G*.

First, the algorithm finds nodes 3 and 6 as cut nodes. Then, it identifies three biconnected components in *G* (line 1), which are shown in Figure 10.

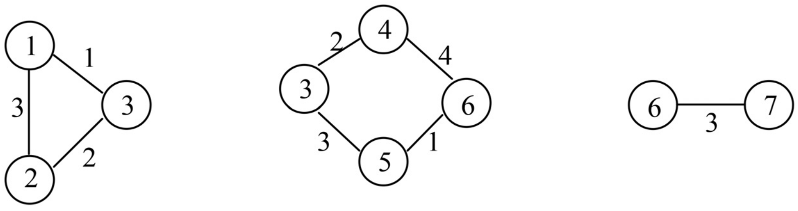

Figure 10 – Biconnected components of *G*.

As no biconnected component of *G* has more than five nodes (line 2), the algorithm computes, through Gusfield algorithm, the cut-tree of each biconnected component (line 3). The resulting cut-trees are illustrated in Figure 11.

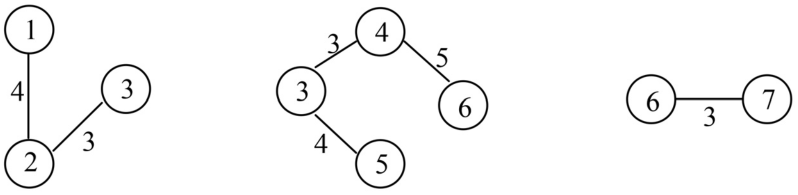

Figure 11 – Cut-trees of the biconnected components of *G*.

Finally, all the cut-trees of the biconnected components are joined to form the cut-tree of *G* (line 4), as in Figure 12.

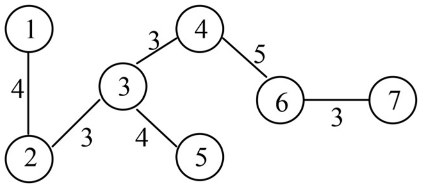

Figure 12 – Cut-tree of *G*.

Regarding the single maximum flow problem, we can implement a new approach based on Lemmas 3 and 5. After identifying the biconnected components of the network and a path between the source and the terminal nodes, the method computes, under the condition that there is no biconnected component with more than $0.8n$ nodes, all $(s, z_1), (z_1, z_2), \dots, (z_x, t)$ maximum flows values. Finally, it returns the minimum value among them. Otherwise, if there is a biconnected component with more than $0.8n$ nodes, a maximum flow algorithm is applied to G. Note that, if s and t are in a same biconnected component, there will be no z nodes, and the method will run just once the single maximum flow algorithm. The choice for the size of $0.8n$, in the condition of the algorithm, will be also discussed in Section 5. Figure 13 shows the pseudo-code of this method, namely MaxFlow_CN.

```
Input: $G, s, t$
1  Identify all biconnected components in $G$ and a path $P$ between $s$ and $t$;
2  if no biconnected component has more than 0.8n nodes then
3  | Let $z$ be the current node while traversing $P$ from $s$ to $t$;
4  | Set $z$ as the node adjacent to $s$;
5  | while $z \neq t$ do
6  | | if $z$ is a cut node of $G$ then
7  | | | Compute $f_{s,z}$ in the biconnected component that contains both $s$ and $z$;
8  | | | $s \leftarrow z$;
9  | | $z \leftarrow$ next node in $P$;
10 | Compute $f_{s,z}$ in the biconnected component that contains both $s$ and $z$;
11 | maxflow $\leftarrow$ minimum value among all $f_{s,z}$;
12 else
13 | maxflow $\leftarrow$ Compute $f_{s,t}$ in $G$;
14 return maxflow;
```

Figure 13 – Pseudo-code of MaxFlow_CN.

Let us exemplify the application of MaxFlow_CN algorithm in the network shown in Figure 14, taking $s = 1$ and $t = 6$.

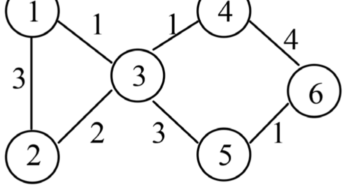

Figure 14 – Network G.

First, the algorithm finds node 3 as cut node in G. Then, it identifies two biconnected components in G, shown in Figure 15, and the path 1-2-3-5-6 between s and t (line 1).

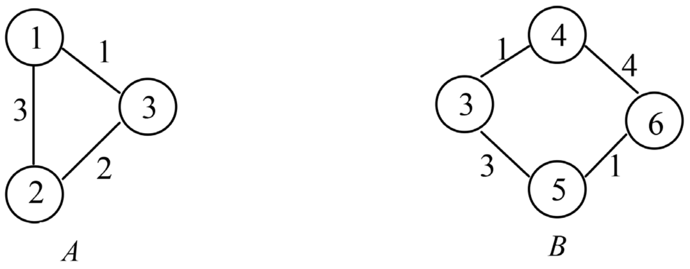

Figure 15 – Biconnected components A and B of G.

As no biconnected component of G has more than four nodes (line 2), the algorithm sets z as node 2 (lines 3 and 4). Then, since $2 \neq t$, the command **while** is executed (line 5). As node 2 is not a cut node (line 6), z is now updated as node 3 (line 9), and **while** is executed again, since $3 \neq t$. Now, as z is a cut node (the **if** condition is true), $f_{1,3}$ is computed in the biconnected component A and s is updated to node 3 (lines 6, 7 and 8). In this case, $f_{1,3} = 3$ with minimum cut composed by edges $[1, 3]$ and $[2, 3]$. In the third iteration of the loop **while**, with z set as node 5, the **if** condition is not true. Then, z is updated to node $6 = t$, causing the stopping of **while**. A maximum flow algorithm is run in B, resulting in $f_{3,6} = 2$, with minimum cut composed by $[3, 4]$ and $[5, 6]$ (line 10). Finally, the maximum flow value between nodes 1 and 6 is the minimum value between $f_{1,3}$ and $f_{3,6}$ (line 11), which is 2.

Since the method seeks the minimum $f_{s,z}$ value, we compute each $f_{s,z}$ faster by limiting it to an upper bound variable u. Initially $u = \mathrm{M}$, where M is a big number, and when $f_{s,z} < u$, then $u = f_{s,z}$. The maximum flow algorithm will perform lesser operations as no flow can exceed u. In the example above, the variable u is updated twice, from M to 3 and from 3 to 2, after computing respectively $f_{1,3}$ and $f_{3,6}$.

As in CN, the algorithms of Hopcroft & Tarjan (1973) and the highest-label preflow-push of Goldberg & Tarjan (1986) were used in MaxFlow_CN to identify the biconnected components and to compute the maximum flows, respectively.

5 COMPUTATIONAL EXPERIMENTS

In this section, we report computational experiments with CN and MaxFlow_CN, proposed here, in comparison with the algorithms of Gusfield (GUS) and Goldberg and Tarjan (MaxFlow), which were implemented by Skorobohatyj (2011). Moreover, we show some numerical tests that empirically defined the condition in line 2 in CN and MaxFlow_CN. In the four algorithms, the maximum flow value is computed with an optimized version of the highest-label preflow-push algorithm, where the resulting flow is not computed for all edges.

Both CN and MaxFlow_CN were implemented in C language with input file format and data structure from Skorobohatyj (2011). The algorithms were compiled with Mingw through the software Dev-C ++ 5.10 and the tests were run on a computer with 64-bit Processor Intel (R) Core (TM) i3 of frequency 3.50 GHz, 4 GB of RAM under the Windows 8 operational system.

For the experiments, the instances were created by the generators PATHGEN and TREEGEN, from Goldberg & Tsioutsiouliklis (2001), and PARTEDGEN, CACTUSGEN, and TESTGEN, specially developed here for this purpose. The desired characteristics of the test instances are: the presence of cut nodes or the high chance of having cut nodes in the generated graphs.

Let us denote the parameters used by the five generators: n the number of nodes in the graph, d the density of the graph given in terms of a percentage of arcs (that indirectly defines the number of arcs m), P the edge capacity factor, and S the seed of the generator. As follows, we introduce them briefly.

Given the path length (parameter k), the PATHGEN builds a path of $k-1$ edges and connects the remaining $n-k$ nodes to the path nodes at random. Then, it adds edges at random to achieve the desired number of arcs and to make the minimum cut problems more difficult. For the PATHGEN, the k value determines the path shape. For example, if $k=n$, then we get one path through all the nodes; if $k=1$, then we have all nodes sharing an edge with node 1.

Given the tree shape (parameter k), the TREEGEN generator builds a tree by connecting node i, $i=2,\dots,n$, to a randomly chosen node in $\{1, \min\{i-1, k\}\}$. Then, it adds edges at random to achieve the desired number of arcs and to make the minimum cut problems more difficult. The value of k determines the shape of the tree. For example, if $k=1$, then the tree is a star. If $k=n-1$, then the tree is obtained by connecting each node, except the first one, to a randomly chosen preceding node.

Given the number of biconnected components (parameter k), the PARTEDGEN builds a graph with $k-1$ cut nodes linking biconnected components of equal size. After building a path through all the nodes in the first step, it adds strategic edges to create k-edge disjoint cycles of size approximately n/k. Finally, it adds edges at random in each biconnected component to achieve the desired number of arcs.

To explain CACTUSGEN, the following definition is necessary.

Definition 5.1. *A connected graph in which every two cycles have at most one node in common is a* ***cactus graph****.*

Given the number of cycles (parameter k), the CACTUSGEN generator builds a cactus graph with k cycles. We created two types of the generator: CACTUS_PATHGEN and CACTUS_-STARGEN. In the first type, a path through all the nodes is built and then k edges are added to form k-edge disjoint cycles, and, in the second one, all k cycles have the same size and one node in common. In both, the density parameter d is not considered, since in these cases k defines the number of edges m.

Given the size of a biconnected component (parameter k), the TESTGEN builds a graph where one of its biconnected components has size approximately nk, with k being a percentage. After building a path through all the nodes in the first step, it adds a strategic edge to create a cycle of size approximately nk. Finally, it marks the nodes of the cycle with color 1 and the remaining nodes with color 2, and then it adds edges at random, by connecting nodes with the same color, to achieve the desired number of arcs.

In the computational experiments, we consider up to three distinct seeds ($S = 1, 2, 3$), except for the generation of PARTED instances. The edge capacities are chosen uniformly at random from the interval $[1, \ldots, 100P]$. We set the factor $P = 1$ when generating all the instances, as well as $n = 1000$. We observe that this value of n is relative large in relation to the ones used in Goldberg & Tsioutsiouliklis (2001). The inputs of algorithms MaxFlow and MaxFlow_CN were $s = 1$ and $t = 1000$, so that s and t were in different biconnected components in the generated instances.

To estimate the maximum size of a biconnected component, needed for the condition in line 2 of the algorithms CN and MaxFlow_CN, numerical tests were performed for instances of the family TEST, with the following variants of CN and MaxFlow_CN:

- CN_0 and MaxFlow_CN_0: CN and MaxFlow_CN implemented with the size $0.0{*}n$ in line 2 condition, i.e., for these variants, the condition is never satisfied;
- CN_1 and MaxFlow_CN_1: CN and MaxFlow_CN implemented with the size $1.0{*}n$ in line 2 condition, i.e., for these variants, the condition is always satisfied.

Hereafter, for each class of the test instances we show the comparison of the running times obtained by the proposed algorithms. The running times obtained by CN_0, CN_1, MaxFlow_CN_0 and MaxFlow_CN_1 are reported in Table 1. The running times obtained by the algorithms CN, GUS, MaxFlow_CN and MaxFlow for the PARTED and PATH generated instances are summarized in Tables 2 and 3, respectively. The results obtained for the TREE generated instances are shown in Table 4. Tables 5 and 6 show the running times for the CACTUS_PATH and CACTUS_STAR instances, respectively. For each test instance, the running time refers to the median of five runs of the algorithm given in microseconds (μs). The symbol $*$ that may appear next the seed value means that a biconnected component of the graph instance has more than 80% of the nodes of the graph instance.

Table 1 – Running time for the TEST instances with $n = 1000$, $m = 3497$, $k = 99, 95, 90, 85, 80, 75$ and $S = 1, 2$.

TEST ($n = 1000, m = 3497$)								
	$k = 99$	$k = 95$	$k = 90$	$k = 85$	$k = 80$		$k = 75$	
					$S = 1$	$S = 2$	$S = 1$	$S = 2$
CN_0	123314	123423	121045	115754	123865	120208	115726	116518
CN_1	180503	184355	158692	146726	133504	128467	119713	115259
MaxFlow_CN_0	348	329	328	309	361	352	357	459
MaxFlow_CN_1	561	485	521	481	536	553	492	518

Analyzing the numerical results, we point out that:

1. Regarding the TEST instances, we observe that CN_1 and CN_0 have closer results when $k = 75$. For this reason, the condition in line 2 of CN was set to the 80% percentage;

Table 2 – Running time for the PARTED instances with $n = 1000$, $m = 3497$ and $k = 2, 4, 8, 16$.

PARTED ($n = 1000, m = 3497$)				
	$k = 2$	$k = 4$	$k = 8$	$k = 16$
CN	71204	29685	15059	8394
GUS	97514	95450	94860	94339
MaxFlow_CN	433	401	390	424
MaxFlow	260	179	353	121

Table 3 – Running time for the PATH instances with $n = 1000$, $m = 1399$, $k = 250, 500, 750$ and $S = 1, 2, 3$.

PATH ($n = 1000, m = 1399$)									
	$k = 250$			$k = 500$			$k = 750$		
	$S = 1$	$S = 2$	$S = 3$	$S = 1$	$S = 2$	$S = 3$	$S = 1*$	$S = 2*$	$S = 3*$
CN	38200	35789	35667	49642	49253	50138	60114	61432	59639
GUS	55046	54261	55628	58469	58095	58162	57825	58748	57192
MaxFlow_CN	184	185	191	208	203	194	164	157	156
MaxFlow	180	65	64	71	65	58	76	68	68

Table 4 – Running time for the TREE instances with $n = 1000$, $m = 1549$, $k = 250, 500, 750$ and $S = 1, 2, 3$.

TREE ($n = 1000, m = 1549$)									
	$k = 250$			$k = 500$			$k = 750$		
	$S = 1$	$S = 2$	$S = 3$	$S = 1$	$S = 2*$	$S = 3$	$S = 1*$	$S = 2*$	$S = 3*$
CN	47976	45278	47024	50595	60095	55410	60904	62053	60074
GUS	58001	57821	56908	58532	57835	59569	58825	59864	58519
MaxFlow_CN	320	240	395	246	377	289	241	300	221
MaxFlow	184	89	223	136	256	135	141	195	120

2. Still, for the TEST instances, since MaxFlow_CN_1 running time does not get better when the parameter k decreases, the condition in line 2 of MaxFlow_CN was also set to the 80% percentage;

3. For the PARTED instances, when the parameter k increases, CN performance gets much better than GUS performance;

4. In both PATH and TREE instances, when the parameter k increases, the performance of CN decreases;

Table 5 – Running time for the CACTUS_PATH instances with $n = 1000$, $k = 10, 20$ and $S = 1, 2, 3$.

CACTUS_PATH ($n = 1000$)						
	$k = 10$ ($m = 1009$)			$k = 20$ ($m = 1019$)		
	S = 1	S = 2	S = 3	S = 1	S = 2	S = 3
CN	8901	9047	8905	4812	4867	4777
GUS	68897	67814	70287	59783	57929	61114
MaxFlow_CN	116	117	131	114	130	128
MaxFlow	115	161	121	122	80	52

Table 6 – Running time for the CACTUS_STAR instances with $n = 1000$, $k = 10, 20$ and $S = 1, 2, 3$.

CACTUS_STAR ($n = 1000$)						
	$k = 10$ ($m = 1009$)			$k = 20$ ($m = 1019$)		
	S = 1	S = 2	S = 3	S = 1	S = 2	S = 3
CN	9079	9090	9261	5313	5246	5320
GUS	69456	65624	65816	57762	55744	56923
MaxFlow_CN	63	65	61	52	51	52
MaxFlow	113	77	70	80	69	34

5. CN running times for PATH instances were up to 35% ($k = 250$, $S = 3$) lower than GUS. For the TREE instances, CN running times were up to 21% ($k = 250$, $S = 2$) lower than GUS;

6. For the generated instances with a biconnected component with more than $0.8n$, CN performs very close to GUS algorithm, while MaxFlow_CN performs worse than MaxFlow;

7. For CACTUS_PATH and CACTUS_STAR instances, CN outperforms GUS, even better when k increases;

8. For almost all instances, MaxFlow obtains better running times than MaxFlow_CN.

One possible reason to explain why the performance of CN improves when the parameter k increases in PARTED, CACTUS_PATH and CACTUS_STAR instances is that the biconnected components of the graphs become smaller. For the PATH and TREE instances, the opposite may occur, that is, when k increases, the biconnected components become larger.

Through point 6 above, the worse performance of MaxFlow_CN is explained by the fact that the identification of biconnected components is an expensive task for the single maximum flow case. Besides the identification of biconnected components, the execution of the maximum flow algorithm more than once also seems to have a significant impact in the running times of MaxFlow_CN.

Regarding CN's good performance, we observe that it executes $n - 1$ maximum flow algorithms in subgraphs of the original graph, whereas GUS applies $n - 1$ maximum flow algorithms in the original graph.

6 CONCLUSION

This work studied the relation between the maximum flows and cut nodes and proposed new approaches to solve the single and the multi-terminal maximum flow problem in graphs with cut nodes, aiming to reduce the running time, i.e., the computational complexity, when compared to the results obtained by classical algorithms.

The computational experiments conducted with the proposed methods used instances generated by PATHGEN, TREEGEN, PARTEDGEN and CACTUSGEN, where the last two were especially developed here. The numerical results pointed out that CN algorithm has better performance in comparison to Gusfield algorithm, whereas the MaxFlow_CN algorithm could not overcome the MaxFlow algorithm.

Variants of the proposed methods can still be developed and tested. For instance, a comparison study can be done with CN being implemented with Gomory and Hu's method as subroutine instead of Gusfield's. Since the maximum flow algorithm implemented by Skorobohatyj, that we used in both MaxFlow and MaxFlow_CN, is an optimized version of the Goldberg & Tarjan (1986) algorithm, tests with the full version are recommended.

REFERENCES

[1] BARTH D, BERTHOMÉ P, DIALLO M & FERREIRA A. 2006. Revisiting parametric multi-terminal problems: Maximum flows, minimum cuts and cut-tree computations. *Discrete Optimization*, **3**(3): 195–205.

[2] BHALGAT A, HARIHARAN R, KAVITHA T & PANIGRAHI D. 2007. An *$\tilde{O}(mn)$* Gomory-Hu Tree Construction Algorithm for Unweighted Graphs. STOC'07 Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, 605–614, June.

[3] COHEN J & DUARTE JR EP. 2001. Fault-Tolerant Routing of Network Management Messages in the Internet. The 2nd IEEE Latin American Network Operations and Management Symposium (LANOMS'2001), 87-98, Belo Horizonte, MG.

[4] DIALLO M. 2011. Méthodes d'Optimisation Appliquées aux Réseaux de Flots et Télécoms. Editions universitaires europeennes.

[5] ELMAGHRABY SE. 1964. Sensitivity analysis of multi-terminal flow networks. *Operations Research*, **12**(5): 680–688.

[6] FORD LR & FULKERSON DR. 1973. Flows in Networks. Princeton University Press, Princeton, NJ.

[7] GOLDBERG AV & TARJAN RE. 1986. A new approach to the maximum flow problem. *ACM Symposium On Theory of Computing*, 136–146, Berkeley, California, May.

[8] GOLDBERG AV & TSIOUTSIOULIKLIS K. 2001. Cut-tree Algorithms: An experimental study. *Journal of Algorithms*, **38**(1): 51–83.

[9] GOMORY RE & HU TC. 1961. Multi-terminal network flows. *SIAM Journal of Computing*, **9**(4): 551–570, December.

[10] GUSFIELD D. 1990. Very simple methods for all pairs network flow analysis. *SIAM Journal of Computing*, **19**(1): 143–155.

[11] HOPCROFT J & TARJAN R. 1973. Efficient algorithms for graph manipulation. *Communications of the ACM*, **16**(6): 372–378.

[12] HU TC & SHING MT. 2002. Combinatorial Algorithms, Enlarged 2nd Ed. Dover Publications, INC, Mineola, New York.

[13] HUSIMI K. 1950. Note on Mayers' theory of cluster integrals. *Journal of Chemical Physics*, **18**(5): 682–684.

[14] SKOROBOHATYJ G. 2011. Solver for the "all-pairs" minimum cut problem in undirected graphs. Zuse Institute Berlin. `http://ftp.zib.de/pub/Packages/mathprog/mincut/`. Last visited on January 7, 2011.

[15] TUNCBAG N, SALMAN FS, KESKIN O & GURSOY A. 2010. Analysis and network representation of hotspots in protein interfaces using minimum cut trees. *Proteins: Structure, Function, and Bioinformatics*, **78**(10): 2283–2294.