ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
CONSIGLIO NAZIONALE DELLE RICERCHE

A. Galluccio, G. Proietti

POLYNOMIAL TIME ALGORITHMS FOR EDGE-CONNECTIVITY AUGMENTATION PROBLEMS

R. 533 Ottobre 2000

Anna Galluccio - Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, 00185 Roma, Italy. Email:galluccio@iasi.rm.cnr.it.

Guido Proietti - Dipartimento di Matematica Pura e Applicata, Università di L'Aquila, via Vetoio, 67010 L'Aquila, Italy, and.Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, 00185 Roma, Italy. Email:proietti@univaq.it

Istituto di Analisi dei Sistemi ed Informatica, CNR
viale Manzoni 30
00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.rm.cnr.it
URL: http://www.iasi.rm.cnr.it

Abstract

Given a graph G of n vertices and m edges, and a spanning subgraph H of G, the problem of finding a minimum weight set of edges of G, denoted as $\textsc{Aug}_2(H, G)$, to be added to H to make it 2-edge connected, is known to be NP-hard. We present polynomial time efficient algorithms for solving special cases of this classic augmentation problem. More precisely, we show that for unweighted graphs, if H is a depth first search tree in G, then $\textsc{Aug}_2(H, G)$ can be computed in $O(m)$ time and space. Moreover, if H is a Hamiltonian path in G and G is non-negatively weighted, we show that $\textsc{Aug}_2(H, G)$ can be computed in $O(m + n \log n)$ time and $O(m)$ space. These results have an interesting application for solving a survivability problem on communication networks.

1. Introduction

Let $G = (V, E)$ be a connected, undirected graph, of n vertices and m edges, where an edge $e = (u, v) \in E$ represents a potential connection between vertices u and v. Let us assume that a non-negative weight is associated with each edge $e \in E$, expressing some cost for activating the edge, and consider the problem of building a network in G which allows all the sites to communicate. For this kind of networks, it is generally important to be both *economically attractive*, i.e., they should be as sparse as possible to reduce set-up costs, and *reliable*, i.e., they should remain operational even if individual network components fail.

As a consequence, the problem of designing networks which combine in the best possible way these two conflicting parameters, i.e., sparseness and reliability, is usually known as the *survivable network design problem* [8, 9]. This topic encompasses a large set of practical (e.g., communication and transportation networks design, VLSI layout [13], etc.), as well as theoretical problems (e.g., the Steiner tree [16], the traveling salesman, the minimum cost k-connected subgraph [10], etc.).

The cheapest solution to the problem of designing a communication network in G is that of designing a *minimum weight spanning tree* of G, namely a connected, spanning subgraph of G such that the sum over all the edge weights is minimum. Unfortunately, such a structure will not even survive a single link or site failure. For the case of link failures, which is of interest for this paper, one possibility to solve the problem is that of designing networks with higher edge-connectivity degree. In fact, edge-connectivity $k > 1$ implies the existence of $k - 1$ edge-disjoint paths between any pair of nodes. Thus, a k-edge connected network will survive to the disruption of $k - 1$ links. However, this approach has (at least) two drawbacks: First, computing a minimum weight k-edge connected spanning network of a given graph is NP-hard (although, approximable within a constant ratio [7, 12, 15]), and second, the communication protocol redundancy grows as k grows.

Fortunately, in practical applications, we can safely assume that a damaged network component can be restored quite quickly, and therefore the likelihood of having multiple overlapping failures is small. Hence, an alternative strategy to increase the reliability without oversizing the network may be to design it onto two levels: a *primary level* of active links (i.e., the backbone where communication is carried out in the absence of failures), and a *secondary level* of inactive links, a subset of which will switch to active as soon as the network undergoes to some link failure. Given their role, links on the secondary level are called *replacement* links.

From a theoretical point of view, the problem of finding a minimum weight set of replacement edges to restore the edge connectivity in case of an edge failure in the network is a classic *edge-connectivity augmentation problem*. In its more general formulation, this problem consists of finding a minimum weight set of edges of a graph G whose addition to a given spanning connected subgraph H of G increases its edge-connectivity to a prescribed value. Such a problem turns out to be NP-hard [4]. Nevertheless, Eswaran and Tarjan proved it can be solved in polynomial time if G is complete and all edges have weight 1, namely all potential links between sites may be activated at the same cost [4]. Concerning approximate solutions for general, undirected graphs, the best approximation algorithm known for the weighted case guarantees a performance ratio of 2 [6, 11], while for the unweighted case, Nagamochi and Ibaraki developed a $(51/26 + \epsilon)$-approximation algorithm, for any constant $\epsilon > 0$ [14]. Analogous versions of augmentation problems for vertex-connectivity and for directed graphs have been widely studied, and we refer the interested reader to the following comprehensive papers [5, 10].

The main purposes of the present paper are the following: we show that if G is unweighted

and H is a depth first search tree in G, then finding the minimum number of edges of G to be added to H to increase its edge connectivity to 2 can be solved in optimal $O(m)$ time and space. Moreover, if G has non-negative weights on its edges and H is a Hamiltonian path in G, then the weighted version of the above problem can be solved in $O(m + n \log n)$ time and $O(m)$ space.

Our algorithms find practical applications in those scenarios in which 2 overlapping link failures in communication networks has to be afforded, but a 3-edge connected primary level of the network itself is too costly to be maintained. In such a case, we can design a 2-edge connected primary level of the network, and as soon as a link fails, we can efficiently provide an optimal set of replacement links, both in the weighted and in the unweighted case, so that the *emergency network*, as obtained by removing from the original network the failed link and by adding the corresponding replacement links, is again 2-edge connected, while at the same time it keeps on to maintain all the old, still working, links.

The paper is organized as follows: in Section 2 we give some basic definitions that will be used throughout the paper; in Section 3 we show how to augment a depth first search tree in unweighted graphs, while in Section 4 we study the problem of augmenting a Hamiltonian path in weighted graphs; in Section 5, we apply these results to handle transient edge failures in 2-edge connected networks, and finally, in Section 6, we present conclusions and list some open problems.

2. Basic definitions

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. G is said to be *weighted* if there exists a real function $w : E \mapsto \mathbb{R}$, otherwise G is *unweighted*. In this paper, we will be concerned with non-negatively weighted graphs. If multiple edges between pairs of vertices are allowed, then the graph is said to be a *multigraph*. A graph $H = (V(H), E(H))$ is called a *subgraph* of G if $V(H) \subseteq V$ and $E(H) \subseteq E$. If $V(H) = V$ then H is called a *spanning subgraph* of G. The weight of H is defined as $w(H) = \sum_{e \in E(H)} w(e)$.

A *simple path* (or a *path* for short) in G is a subgraph H of G with $V(H) = \{v_0, \ldots, v_k | v_i \neq v_j$ for $i \neq j\}$ and $E(H) = \{(v_i, v_{i+1}) | 0 \leq i < k\}$, also denoted as $P(v_0, v_k) = \langle v_0, v_1, \ldots, v_k \rangle$. A *cycle* is a path whose end vertices v_0 and v_k coincide. A spanning path of G is called a *Hamiltonian path* of G. A graph G is *connected* if, for any $u, v \in V$, there exists a path $P(u, v)$ in G.

A connected acyclic graph is called a *tree*. A *rooted* tree T_r is a tree with a privileged vertex $r \in V(T)$ distinguished from the others. Let $P(r, x)$ denote the unique path in T_r between r and $x \in V(T)$. Any vertex y in $P(r, x)$ is called an *ancestor* of x in T_r. Symmetrically, x is called a *descendant* of any vertex y in $P(r, x)$, and x and y are called *relatives* in T_r. A *depth first search* tree of G, *DFS-tree* for short, is a rooted spanning tree T_r of G such that, for any edge $(u, v) \in E \setminus E(T)$, u and v are relatives in T_r. Edges in T_r are called *tree edges*, while the remaining edges of G are called *cycle edges*. A cycle edge (u, v) *covers* all the tree edges along the (unique) path from u to v in T_r.

A graph G is said to be *k-edge connected*, where k is a positive integer, if the removal of any $k - 1$ distinct edges from G leaves G connected. Given an h-edge connected spanning subgraph H of a k-edge connected graph G, and a positive integer $h < \lambda \leq k$, finding a *λ-augmentation* of H in G means to select a minimum weight set of edges in $E \setminus E(H)$, denoted as $\textsc{Aug}_\lambda(H, G)$, such that the spanning subgraph $H' = (V, E(H) \cup \textsc{Aug}_\lambda(H, G))$ of G is λ-edge connected.

3. Augmenting DFS-trees in unweighted graphs

Let us recall the notion of *tree-carving* $\Gamma(G)$ of a 2-edge connected and unweighted graph G [10]. Let $\{V_1, V_2, \ldots, V_k\}$ be a partition of the vertex set V of G; $\Gamma(G)$ is a tree with vertex set $\{\nu_1, \nu_2, \ldots, \nu_k\}$, where vertex ν_i is associated with vertex set V_i, such that for every vertex $v \in V_i$, all the neighbours of v in G belong either to V_i itself, or to V_j, where V_j is adjacent to V_i in the tree $\Gamma(G)$.

We first prove the following:

Theorem 3.1. *Let G be a 2-edge connected and unweighted graph with n vertices and m edges. Let T_r be a DFS-tree of G. Then, $\textsc{Aug}_2(T_r, G)$ can be computed in $O(m)$ time and space.*

Proof. Since T_r is a DFS spanning tree of G, the *DFS-tree partition* of the vertices of G induced by T_r yields a tree-carving of G and can be computed in $O(m)$ time [10].

Let us briefly recall how this partition works. First of all, a set of cycle edges is added to T_r, in the following way: perform a depth first visit of T_r, and before withdrawing from a vertex v for the last time, check whether the edge joining v and its parent in T_r is currently still not covered; if so, cover it by adding a cycle edge which leads to the ancestor of v in T_r closest to r. After this first phase, all the edges of T_r which caused the insertion of a cycle edge are removed, and the resulting connected components in T_r provides the DFS-tree partition. Let $k > 1$ be the number of vertices of the tree-carving $\Gamma(G)$ induced by this DFS-tree partition. The following holds:

Claim: $|\textsc{Aug}_2(T_r, G)| \geq k - 1$.
Proof of the Claim. We know that a lower bound on the number of edges of any 2-edge connected spanning subgraph H of G is $2(k-1)$ [10]. More precisely, each edge (ν_i, ν_j) in $\Gamma(G)$ implies that at least two edges between V_i and V_j are needed in H. Since T_r contains just one edge between V_i and V_j (otherwise we would have a cycle in T_r), it follows that any 2-edge connected spanning subgraph of G contains at least $k-1$ edges which do not belong to T_r. From this, the claim follows.

To complete the proof, we observe that when the $k-1$ cycle edges selected by the DFS-tree partition are added to T_r, its connectivity is augmented to 2. Moreover, $O(m)$ time and space are trivially enough to perform all the operations. ∎

4. Augmenting Hamiltonian paths in weighted graphs

Let $G = (V, E)$ be a 2-edge connected graph with a non-negative weight function w on the edges and let $\Pi = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ be a Hamiltonian path of G. We describe a polynomial time algorithm to solve the $\textsc{Aug}_2(\Pi, G)$ problem.

In the following, a cycle edge (v_i, v_j), $i < j$, will be considered as a *right* edge for v_i and as a *left* edge for v_j, and $\mathcal{L}_i$ and $\mathcal{R}_i$ will denote the set of left and right cycle edges of v_i, respectively. Let Π_k denote the restriction of Π to $\langle v_0, v_1, \ldots, v_k \rangle$, and let e_k denote the edge (v_{k-1}, v_k). A *covering* of Π_k is a set of edges in $E \setminus E(\Pi)$ which cover all the edges of Π_k. Figure 1 illustrates used notation.

In the next subsections, we first give a high-level description of the algorithm, and we then analyze its correctness and its time and space complexity.

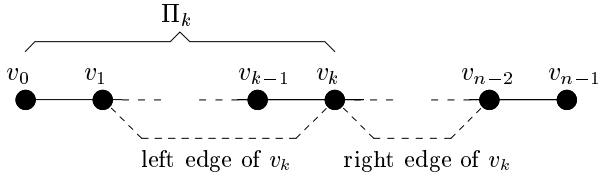

Figure 1: Subpath Π_k of Π, with a left and a right edge of v_k.

4.1. High-level description of the algorithm

The algorithm consists of $n-1$ iterations. At the k-th iteration, the algorithm computes a suitable edge $c(k) = (v_s, v_t)$ covering e_k, and such that the set of edges $\textsc{Sol}(k)$ defined recursively as follows

$$\textsc{Sol}(k) = \begin{cases} \emptyset & \text{if } k = 0, \\ c(k) \cup \textsc{Sol}(s) & \text{if } k > 0 \text{ and } c(k) = (v_s, v_t), \end{cases} \tag{1}$$

is a covering of Π_k. As we will prove later, such a covering satisfies the property of being a minimum weight set of edges covering Π_k. For space efficiency reasons, $\textsc{Sol}(k)$ cannot be stored explicitly throughout the execution of the algorithm. However, as we will see shortly, we only need to maintain its weight $w(\textsc{Sol}(k))$ to guarantee the algorithm correctness.

To select the edge $c(k)$ properly among all the edges covering e_k, the algorithm maintains a set of *active vertices* $\mathcal{V}_k = \{v_k, \ldots, v_{n-1}\}$. With each active vertex $v_j \in \mathcal{V}_k$, two labels are associated:

1. an *edge* $\sigma(v_j)$, belonging to $\mathcal{L}_j$;
2. a *key* $\kappa(v_j)$, containing the weight of a set of edges covering Π_k and using $\sigma(v_j)$.

Throughout the execution of the algorithm, the following invariant is maintained: at the beginning of iteration k, $\sigma(v_j)$ contains an edge (v_s, v_j) of $\mathcal{L}_j$, if any, such that

$$w(\textsc{Sol}(s)) + w((v_s, v_j)) = \min_{\substack{e = (v_i, v_j) \in \mathcal{L}_j \\ i = 0, \ldots, k-2}} \{w(e) + w(\textsc{Sol}(i))\}. \tag{2}$$

The algorithm starts by initializing $\mathcal{V}_0 := V$, and by letting $w(\textsc{Sol}(0)) := 0$. Moreover, it sets $\sigma(v_j) := \emptyset$ and $\kappa(v_j) := +\infty$, for each $v_j \in \mathcal{V}_0$.

At the first iteration, the algorithm aims at covering edge e_1. Hence, it first sets $\mathcal{V}_1 := \mathcal{V}_0 \setminus \{v_0\}$, and then it considers all the right edges of v_0. For each such edge, say $e = (v_0, v_j)$, it sets $\sigma(v_j) = e$ and $\kappa(v_j) = w(e)$. Then, it looks at the element in $\mathcal{V}_1$ having minimum key, say v_t, and sets $c(1) := \sigma(v_t)$ and $w(\textsc{Sol}(1)) := \kappa(v_t)$.

The generic k-th iteration may be described as follows:

`Step 1:` Remove vertex v_{k-1} from $\mathcal{V}_{k-1}$, creating $\mathcal{V}_k$.
(`Comment:` Edges in $\mathcal{L}_{k-1}$ cannot be used to cover e_k and subsequent edges; hence, they are removed from further considerations.)

`Step 2:` Consider all the edges in $\mathcal{R}_{k-1}$. For each such edge, say $e = (v_{k-1}, v_j)$, let

$$\kappa' = w(e) + w(\textsc{Sol}(k-1)). \tag{3}$$

If $\kappa' < \kappa(v_j)$, decrease the key of v_j to value κ', and set $\sigma(v_j) := e$.
(`Comment:` The labels of the active vertices in $\mathcal{V}_k$ are updated. More precisely, at the end of this step, for each v_j in $\mathcal{V}_k$, we have that

$$\kappa(v_j) = \min_{i=0,\ldots,k-1} \{w(e) + w(\textsc{Sol}(i)) \mid e \in \mathcal{R}_i \wedge e \in \mathcal{L}_j\}, \tag{4}$$

and $\sigma(v_j)$ is the left edge of v_j minimizing (4).)

`Step 3:` Find the minimum key in $\mathcal{V}_k$; let v_t be the corresponding vertex, and let $\sigma(v_t) = (v_s, v_t)$. Then set

$$c(k) := \sigma(v_t) \quad \text{and} \quad w(\textsc{Sol}(k)) := \kappa(v_t). \tag{5}$$

At the end of the $(n-1)$-th iteration, the algorithm computes the set $\textsc{Sol}(n-1)$, as defined in (1), by using the edges obtained in (5). In the next subsection, we shall prove that $\textsc{Sol}(n-1)$ contains a minimum weight set of edges covering Π in G.

4.2. Analysis of the algorithm

In order to prove that the algorithm finds a 2-augmentation of Π in G, we first show that $w(\textsc{Sol}(n-1)) = w(\textsc{Aug}_2(\Pi, G))$, namely that the weight of the solution found by the algorithm equals the weight of an optimal solution of the 2-augmentation problem.

We have the following:

Lemma 4.1. $w(\textsc{Sol}(n-1)) = w(\textsc{Aug}_2(\Pi, G))$.

Proof. Let $\textsc{Opt}(k)$ denote a minimum weight set of edges of G covering Π_k, with $\textsc{Opt}(0) := \emptyset$ and $w(\textsc{Opt}(0)) := 0$. Notice that $w(\textsc{Opt}(n-1)) = w(\textsc{Aug}_2(\Pi, G))$. We shall prove that $w(\textsc{Sol}(k)) = w(\textsc{Opt}(k))$, for $k = 0, \ldots, n-1$. The proof is by induction on k. For $k = 0$ and $k = 1$, the thesis follows trivially. Assume the thesis is true up to $k-1 < n-1$, i.e., $w(\textsc{Sol}(i)) = w(\textsc{Opt}(i))$ for $i = 0, \ldots, k-1$. We shall prove that $w(\textsc{Opt}(k)) = w(\textsc{Sol}(k))$. Clearly, $\textsc{Opt}(k)$ has to contain a cycle edge covering e_k. Therefore, an optimal solution for Π_k has weight

$$w(\textsc{Opt}(k)) = \min_{\substack{e=(v_i,v_j)\in E \\ i=0,\ldots,k-1 \\ j=k,\ldots,n-1}} \{w(e) + w(\textsc{Opt}(i))\} =$$

$$= \min_{j=k,\ldots,n-1} \left\{ \min_{i=0,\ldots,k-1} \{w(e) + w(\textsc{Opt}(i)) \mid e \in \mathcal{R}_i \wedge e \in \mathcal{L}_j\} \right\}. \tag{6}$$

On the other hand, from the algorithm, we have that

$$w(\textsc{Sol}(k)) = \min_{j=k,\ldots,n-1} \kappa(v_j) =$$

$$= \min_{j=k,\ldots,n-1} \left\{ \min_{i=0,\ldots,k-1} \{w(e) + w(\textsc{Sol}(i)) \mid e \in \mathcal{R}_i \wedge e \in \mathcal{L}_j\} \right\}, \quad (7)$$

and given that, by assumption, $w(\textsc{Sol}(i)) = w(\textsc{Opt}(i))$ for $i = 0, \ldots, k-1$, we have that (6) and (7) coincide, and the thesis follows. ∎

By making use of the above lemma, the following theorem can finally be proved:

Theorem 4.2. *Let G be a 2-edge connected graph with n vertices, m edges and with non-negative weights on the edges. Let Π be a Hamiltonian path of G. Then, $\textsc{Aug}_2(\Pi, G)$ can be computed in $O(m + n \log n)$ time and $O(m)$ space.*

Proof. To compute $\textsc{Aug}_2(\Pi, G)$, we make use of the algorithm presented in the previous section. The correctness of the algorithm derives from the fact that $\textsc{Sol}(k)$, for any $k = 0, \ldots, n-1$, contains a set of edges of G which, by construction, cover Π_k. Hence, $\textsc{Sol}(n-1)$ consists of a set of edges covering $\Pi_{n-1} = \Pi$, and from Lemma 4.1, its weight is minimum.

The time complexity follows from the maintenance of sets $\mathcal{V}_k, k = 0, \ldots, n-1$, by means of the efficient implementation of priority queues proposed in [2]. In fact, to create $\mathcal{V}_0$ we perform a *MakeQueue* operation, followed by n *Insert* operations (one for each vertex in G). Trivially, $\mathcal{V}_k$ can be obtained from $\mathcal{V}_{k-1}$ by simply removing v_{k-1}, and then we have a total of $n-1$ *Delete* operations. As far as key maintenance is concerned (`Step 2`), notice that $O(m)$ *DecreaseKey* operations take place (since a key may be decreased only when a new right edge is considered, and each right edge is considered at most once). Finally, a total of $n-1$ *FindMin* operations are needed to execute `Step 3` over all the algorithm. Therefore, we obtain a total of $O(m + n \log n)$ time to maintain sets $\mathcal{V}_k, k = 0, \ldots, n-1$, since we pay $O(\log n)$ worst-case time for a *Delete* operation, and $O(1)$ worst-case time for all the other operations [2].

The operations in (3) and (5) are clearly performed in $O(1)$ time for each cycle edge. Moreover, $\textsc{Sol}(n-1)$ can be computed in $O(n)$ time, by making use of (1). Finally, the time complexity for managing sets $\mathcal{R}_i$ and $\mathcal{L}_i, i = 0, \ldots, n-1$, is trivially $O(m)$. So, the overall time complexity of the algorithm is $O(m + n \log n)$.

It is easy to see that all the above operations can be performed by using $O(m)$ space, and then the claim follows. ∎

5. Maintaining biconnectivity through augmentation

The results of the previous sections have an interesting application for solving a survivability problem on networks, that is the problem of adding to a given 2-edge connected network undergoing to a transient edge failure, the minimum weight set of edges needed to reestablish the biconnectivity. In this way, extensive (in terms of both computational efforts and set-up costs) network restructuring is avoided.

Let H be a 2-edge connected spanning subgraph of a 3-edge connected graph G. Let $G - e$ denote the graph obtained from G by removing an edge $e \in E$. Given an edge $e \in E(H)$, if $H - e$ is not 2-edge connected, then we say that e is *vital* for H. In the sequel, an edge e removed from H will always be considered as vital for H.

Let $\textsc{Aug}_2(H - e, G - e)$ be a minimum weight set of edges in $E \setminus E(H - e)$ such that the spanning subgraph $H' = (V, E(H - e) \cup \textsc{Aug}_\lambda(H - e, G - e))$ of $G - e$ is 2-edge connected. Using the results of the previous sections, we prove that $\textsc{Aug}_2(H - e, G - e)$ can be computed efficiently when G has non-negative weights on the edges. More precisely:

Theorem 5.1. *Let G be a 3-edge connected graph with n vertices, m edges and with non-negative weights on the edges. Let H be a 2-edge connected spanning subgraph of G. Then, for any vital edge $e \in E(H)$, we have that $\textsc{Aug}_2(H-e, G-e)$ can be computed in $O(m + n \log n)$ time and $O(m)$ space. The running time can be lowered to $O(m)$ if all edge weights are equal.*

Proof. After the removal of e from H, every 2-edge connected component in $H - e$ can be contracted into a single vertex in $O(m)$ time and space [1]. Let V_i denote the vertex set of the i-th 2-edge connected component in $H - e$, and let $\{V_1, V_2, \ldots, V_k\}$ be the vertex partition of V induced by contracting all such components. Let $\Pi = \langle \nu_1, \nu_2, \ldots, \nu_k \rangle$ be the path resulting from the contraction, where vertex ν_i is associated with vertex set V_i. Let $\mathcal{G}$ be the multigraph with vertex set $V(\mathcal{G}) = V(\Pi)$ and edge set

$$E(\mathcal{G}) = E(\Pi) \cup \{(\nu_i, \nu_j) \mid \exists\ u \in V_i \wedge \exists\ v \in V_j \text{ such that } (u, v) \in E \setminus E(H - e)\}.$$

It is easy to realize that the algorithms presented in Section 3 and Section 4 can be extended to the case where parallel edges in G are allowed. Therefore, since Π is a Hamiltonian path in $\mathcal{G}$, we can apply both Theorem 3.1 and Theorem 4.2. It follows that, for any given edge $e \in E(H)$, there exist polynomial time algorithms to compute $\textsc{Aug}_2(H-e, G-e)$. Their complexity is the same as in Theorem 3.1 and Theorem 4.2, respectively. ∎

6. Conclusions

In this paper we presented time and space efficient algorithms for solving special cases of the classic problem of finding a minimum weight set of edges that has to be added to a spanning subgraph of a given (either unweighted or non-negatively weighted) graph to make it 2-edge connected. These techniques have been applied to solve efficiently an interesting survivability problem on 2-edge connected networks.

For the weighted case, our algorithm is efficient, but it is still open to establish whether its running time is optimal. Apart from that, many interesting problems remain open: (1) the extension of the result in Theorem 4.2 to DFS-trees; (2) the extension to vertex-connectivity augmentation problems, which are of interest for managing transient vertex failures in 2-vertex connected networks; (3) the extension of the results contained in Section 5 to the case in which all the possible edge failures in H are considered, aiming at providing a faster solution than that it would be obtained by applying our algorithms $O(|E(H)|)$ times, one for the failure of each vital edge in H.

We consider the last one as the highest-priority open problem, and we plan to attack it by means of ad-hoc amortization techniques. In fact, from a network management point of view, computing *a priori* the augmentation set associated with every edge in the network is essential to know how the network will react in any possible link failure scenario.

Acknowledgements – The authors are very grateful to Peter Widmayer for helpful discussions on the topic.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The design and analysis of computer algorithms*, Addison Wesley, (1974).

[2] G.S. Brodal, Worst-case efficient priority queues, *Proc. 7th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'96)*, ACM/IEEE Computer Society, 52–58.

[3] J. Cheriyan and R. Thurimella, Approximating minimum-size k-connected spanning subgraphs via matching, *Proc. 37th Ann. IEEE Symp. on Foundations of Computer Science (FOCS'96)*, IEEE Computer Society, 292–301.

[4] K.P. Eswaran and R.E. Tarjan, Augmentation problems, *SIAM Journal on Computing*, **5** (1976) 653–665.

[5] A. Frank, Augmenting graphs to meet edge-connectivity requirements, *SIAM Journal on Discrete Mathematics*, **5** (1992) 25-53.

[6] G.N. Frederickson and J. Jájá, On the relationship between the biconnectivity augmentation problems, *SIAM Journal on Computing*, **10** (1981) 270–283.

[7] A. Galluccio and G. Proietti, Towards a 4/3-approximation algorithm for biconnectivity, IASI-CNR Technical report R. 506, June 1999.

[8] M.X. Goemans and D.J. Bertsimas, Survivable networks, linear programming relaxations and the parsimonious property, *Mathematical Programming*, **60** (1993) 145–166.

[9] M. Grötschel, C.L. Monma and M. Stoer, Design of survivable networks, in *Handbooks in OR and MS, Vol. 7*, Elsevier (1995) 617–672.

[10] S. Khuller, Approximation algorithms for finding highly connected subgraphs, in *Approximation Algorithms for NP-Hard Problems*, Dorit S. Hochbaum Eds., PWS Publishing Company, Boston, MA, 1996.

[11] S. Khuller and R. Thurimella, Approximations algorithms for graph augmentation, *Proc. 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, Vol. 630 of Lecture Notes in Computer Science, Springer, 330–341.

[12] S. Khuller and U. Vishkin, Biconnectivity approximations and graph carvings, *Journal of the ACM*, **41**(2) (1994) 214–235.

[13] R.H. Möhring, F. Wagner and D. Wagner, VLSI network design, in *Handbooks in OR and MS, Vol. 8*, Elsevier (1995) 625–712.

[14] H. Nagamochi and T. Ibaraki, An approximation for finding a smallest 2-edge connected subgraph containing a specified spanning tree, *Proc. 5th Annual International Computing and Combinatorics Conference (COCOON'99)*, Vol. 1627 of Lecture Notes in Computer Science, Springer, 31–40.

[15] S. Vempala and A. Vetta, Factor 4/3 approximations for minimum 2-connected subgraphs, *Proc. 3rd Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2000)*, Vol. 1913 of Lecture Notes in Computer Science, Springer, 262–273.

[16] P. Winter, Steiner problem in networks: a survey, *Networks*, **17** (1987) 129–167.