Dynamic Bottleneck Optimization for k-Edge and 2-Vertex Connectivity *

Orestis A. Telelis[1] Vassilis Zissimopoulos[2]

1. Dept. of Computer Science, University of Aarhus, Denmark
telelis@daimi.au.dk
2. Dept. of Informatics and Telecommunications, University of Athens, Greece
vassilis@di.uoa.gr

Abstract

We consider the problem of updating efficiently the minimum value b over a weighted graph, so that edges with a cost less than b induce a spanning subgraph satisfying a k-edge or 2-vertex connectivity constraint, when the cost of an edge of the graph is updated. Our results include update algorithms of almost linear time (up to poly-logarithmic factors) in the number of vertices for all considered connectivity constraints (for fixed k), and a worst case construction showing that these algorithms are almost optimal in their class.

1 Introduction

On a weighted graph $G(V, E)$ of n vertices and m edges we consider the problem of maintaining efficiently a minimum value b in between updates of edge costs, so that the spanning subgraph induced by edges with a cost at most b is k-edge (for fixed k) or 2-vertex connected. A graph is k-edge/vertex connected if removal of $k-1$ edge/vertices respectively does not disconnect it or, alternatively, if every pair of verices is connected via at least k edge- or vertex-disjoint paths respectively. It is assumed that G satisfies the same constraint. The value b is referred to as the *bottleneck* value for k-edge- or 2-vertex-connectivity of the underlying graph.

We design efficient *dynamic* algorithms that update b, when the cost of an edge of the underlying graph is updated. Our work falls in the field of *dynamic graph algorithms*, that maintain efficiently the solution to a graph problem over a dynamic graph, changing by well defined perturbations, most commonly edge insertions and deletions. A dynamic algorithm encodes the dynamic graph along with supplementary information related to the problem at hand into an appropriate data structure, and offers update operations for edge deletions and/or insertions. It also offers an operation for querying the solution to the problem. *Fully Dynamic* algorithms handle both kinds of operations, while incremental/decremental algorithms handle exclusively edge insertions or deletions (see [6] for a survey).

Bottleneck optimization for graph connectivity models minimum energy consumption minimization in wireless adhoc networks subject to ensuring connected fault-tolerant topologies [12], and generalizes the problem of deciding whether a dynamic graph changing by edge

*Research supported by the European Union (75%) and the Greek Government (25%), under project $E\Pi EAEKII$, "Archimedes", of T.E.I. - Chalkida, while the first author was at the Dept. of Informatics and Telecommunications, University of Athens, Greece.

insertions/deletions is k-connected (by usage of binary costs simulating absence/presence of an edge). Previous work on bottleneck optimization problems includes a linear $O(m+n)$ time algorithm for finding a bottleneck simply connected spanning subgraph [2] (see also [4]), algorithms for the *directed bottleneck spanning tree*, the *bottleneck maximum cardinality matching* [8], and $O(m+n)$ time algorithm for the bottleneck 2-vertex connected spanning subgraph [13]. To the best of our knowledge, no previous results concerned dynamic bottleneck optimization.

In this paper we design efficient dynamic algorithms for bottleneck k-edge connectivity, for arbitrary fixed k (section 3), and 2-vertex connectivity (section 4). We also give a justification of the intuition that our developments are of near-optimal complexity (section 5).

2 Preliminaries

We talk of k-connectivity when our statements apply to both kinds (edge and vertex) of connectivity. A vertex/edge whose removal increases the number of connected components of a graph is referred to as *cutvertex/bridge* respectively. End-vertices of a bridge are cutvertices. All graphs in this paper are denoted with their edge set and have n vertices. All set operations between graphs are with respect to their edge sets. Given a minimum spanning forest (MSF) F and two vertices u, v, $[u \cdots v]_F$ denotes the edge set of the unique path connecting u to v in F, if such a path exists, otherwise define $[u \cdots v]_F = \emptyset$. A *bottleneck k-connected (spanning) subgraph* of a weighted graph G is a k-connected spanning subgraph of G, the most expensive edge of which has a cost equal to the bottleneck value for k-connectivity of G. Notation $\widetilde{O}$ hides polylogarithmic factors from the complexity expression. The notion of sparse certificates has found extended application in designing dynamic graph algorithms [7]. We use the following definitions for certificates:

Definition 1 *A sparse certificate for bottleneck k-connectivity of a weighted graph G is a spanning edge-subgraph C of G with $O(n)$ edges, such that the bottleneck k-connected subgraph of C is a bottleneck k-connected subgraph of G.*

Definition 2 *A relaxed certificate for bottleneck k-connectivity of an arbitrary weighted graph G is a (not necessarily spanning) subgraph $C \subseteq G$, such that, for every edge $(u,v) \in G \setminus C$ there are k edge- or vertex-disjoint paths (depending on the kind of connectivity in discussion) that connect u and v in C and for each edge e on these paths $c(e) \leq c(u,v)$.*

Apart from relaxed certificates, all other subgraphs will be spanning, unless otherwise stated.
Incremental Algorithms Throughout the paper we use incremental algorithms for maintaining k-connected components of a graph changing by edge insertions. These algorithms provide operations for inserting an edge and for querying whether two vertices of the current graph belong in the same k-connected component (i.e. are k-connected to each other) or not. In particular, for edge connectivity we will use the algorithms developed in [16, 15, 5], for maintaining $2-$, $3-$, and 4-edge connected components of an initially 1-, 2-, and 3-edge connected graph respectively, spending amortized times of order $O(1)$, $O(\alpha(m,n))$, and $O(\alpha(m,n))$, per insertion or query, where m denotes the total number of inserted edges, and α is the inverse Ackermann's function. For 2-vertex connectivity we will use the incremental algorithm described in [16], which has amortized $O(\alpha(m,n))$ complexity per edge insertion or query when the initial graph is not known to be connected and $O(1)$ amortized time when it is connected.

Algorithm 1: Algorithm k-E-CONNECT.

Input: G, $\vec{c}$, k
Output: $E \subseteq G$
$E \leftarrow \emptyset$;
for $r = 1 \dots k$ **do**
  for $e = (u, v) \in G - E$ *in order of non-decreasing cost* $c(e)$ **do**
    if u *and* v *not* r*-edge connected* **then**
      insert (u, v) in E;
    end
  end
end
return E;

3 Edge Connectivity

For k-edge connectivity we use the technique of sparsification introduced in [7]: we maintain dynamically a sparse certificate for bottleneck k-edge connectivity per edge cost update, and execute a static algorithm over the (updated) certificate for obtaining a bottleneck k-edge connected subgraph. The bottleneck value for this subgraph will be the bottleneck value for the underlying graph. As a certificate for bottleneck k-edge connectivity we use a slight modification of the certificate proposed in [14, 3]. This certificate is the subgraph $\mathcal{F}_k = \cup_{i=1}^{k} F_i$, where F_i is a MSF of the subgraph $G - (\cup_{j=1}^{i-1} F_j)$. As shown in [14, 3], $\mathcal{F}_k$ is a sparse certificate of k-edge connectivity of G, that is $|\mathcal{F}_k| = O(n)$, for $k = O(1)$ and G is k-edge connected if and only if $\mathcal{F}_k$ is k-edge connected. This result is extended to our case as follows:

Lemma 1 *If the weighted graph G is k-edge connected, then $\mathcal{F}_k \subseteq G$ is a sparse certificate for bottleneck k-edge connectivity of G, for any fixed value of k.*

Proof. Suppose that G is k-edge connected. Let C be a bottleneck k-edge connected subgraph of G (that is, a certificate for bottleneck k-edge connectivity of G). We will transform C into a subset of $\mathcal{F}_k$, without increasing its bottleneck value. Replace every edge $(u, v) \in C$, $(u, v) \notin \mathcal{F}_k$, with the edge set $(\cup_{i=1}^{k} [u \cdots v]_{F_i})$. Then we observe that since $(u, v) \notin \mathcal{F}_k$, u and v must be connected in every F_i, $i = 1 \dots k$ through a unique path (otherwise (u, v) should belong in exactly one F_i). Furthermore, by optimality of MSF F_i, (u, v) must be the most expensive edge on the induced cycle $(u, v) \cup [u \cdots v]_{F_i}$ for $i = 1 \dots k$. Thus (u, v) is replaced by k edge-disjoint paths connecting u and v, while the bottleneck of C does not increase. □

As mentioned in [7], maintenance of $\mathcal{F}_k$ can be performed by using k copies of a fully dynamic MSF algorithm. For our purposes we use the dynamic algorithm proposed in [10] of $O(\log^4 n)$ amortized update time. Note that the algorithm of [10] handles edge insertions and deletions, while in our case we want to maintain $\mathcal{F}_k$ under edge cost modifications. Therefore, update of an edge's cost is handled as deletion of that edge from the underlying graph and re-insertion at the new cost. This is handled by the k copies of the dynamic MSF algorithm as $O(k)$ edge insertions and deletions.

The second ingredient of our dynamic algorithms for k-edge connectivity is a static algorithm for extracting a bottleneck k-edge connected subgraph from $\mathcal{F}_k$. Algorithm 1 is a general algorithmic scheme for finding a bottleneck k-edge connected subgraph of G.

We refer to algorithm 1 as k-E-CONNECT. It is easy to verify that k-E-CONNECT produces a bottleneck k-edge connected subgraph of its input graph: in the r-th iteration, right before insertion of the maximum cost edge, e_r, there are exactly two r-edge connected components in E. Clearly insertion of any edge e with $c(e) \leq c(e_r)$ could not merge the two components into one, thus e_r is optimum for the r-th iteration. Consequently the most expensive edge inserted in E in the k-th iteration has cost equal to the bottleneck value for k-edge connectivity of G.

Implentation of k-E-CONNECT for $k \leq 4$ can be carried out using the incremental algorithms discussed in section 2. The initialization time of each algorithm is $O(m+n)$, m being the number of edges of the graph on which the algorithm is initialized. The total time spent in initialization is $O(m+n)$ because $k = O(1)$.

For the general case of fixed $k > 4$ we use a different implementation of k-E-CONNECT, which makes use of an incremental algorithm proposed in [9], that answers queries regarding the minimum edge cut of a graph changing by edge insertions. k is the value of the minimum cut in this case. Edge insertions are supported in $O(k \log n)$ time, while queries regarding the minimum cut are answered in $O(1)$ time. The algorithm for extracting a bottleneck k-edge connected subgraph scans the edges of the input graph in order of non-decreasing weight and propagates an edge to the output subgraph if the minimum cut is less than k. For what follows we assume this scheme as a part of k-E-CONNECT (by adding a check for the value of k) for ease of presentation.

In order to execute k-E-CONNECT over the certificate $\mathcal{F}_k$ we need to maintain the edges of $\mathcal{F}_k$ sorted in order of non-decreasing cost. Therefore we also use a balanced binary tree, to encode $\mathcal{F}_k$. During update of $\mathcal{F}_k$ when dynamic MSF insertions/deletions take place, each deleted edge is also deleted from the balanced binary tree, while each inserted edge is also inserted to the tree. Using the tree we can always scan the edge set of $\mathcal{F}_k$ in order of ascending cost when executing k-E-CONNECT. We can now state the following result:

Theorem 1 *A bottleneck k-edge connected subgraph can be maintained in at most $O(n \log n)$ complexity per edge cost update and in $O(n \log^4 n)$ time per $O(n)$ edge cost updates, for any fixed value of k. In detail the complexity is:*

- *$O(n)$ per edge operation for $k = 2$.*
- *$O(n\alpha(n,n))$ per edge operation for $k = 3, 4$.*
- *$O(n \log n)$ per edge operation for fixed $k > 4$.*

Proof. Recall that we update $\mathcal{F}_k$ per edge operation and run k-E-CONNECT on the updated $\mathcal{F}_k$. A single edge cost update causes $O(k)$ edge deletions and insertions to happen during update of $\mathcal{F}_k$, each of them taking $O(\log^4 n)$ amortized time. Furthermore, it causes $O(k)$ removals and insertions of edges, each of $O(\log n)$ time, in the balanced binary tree storing edges of $\mathcal{F}_k$. Thus the total time for updating the certificate for a single edge cost update is $O(\log^4 n)$, given that k is a fixed constant. Thus, updating of $\mathcal{F}_k$ takes $O(\log^4 n)$ time for a single edge cost update and $O(n \log^4 n)$ time for $O(n)$ edge cost updates. Execution of k-E-CONNECT over the updated certificate $\mathcal{F}_k$ is of $\tilde{O}(n)$ time, with polylogarithmic factors given by running times of incremental algorithms [16, 15, 5, 9], and because $|\mathcal{F}_k| = O(n)$. □

Algorithm 2: Algorithm 2-V-CONNECT.

Input: G, $\vec{c}$
Output: $B \subseteq G$
Find a MSF $F \subseteq G$;
$B \leftarrow F$;
for $e = (u, v) \in G$ *in order of non-decreasing cost* $c(e)$ **do**
  if $e \notin B$ *and* u,v *not* 2*-vertex connected in* B **then**
    insert (u, v) in B;
  end
end
return B;

4 2-Vertex Connectivity

For 2-vertex connectivity we convert the static algorithm 2, referred to as 2-V-CONNECT, into a dynamic one, using sparsification. 2-V-CONNECT grows a bottleneck 2-vertex connected subgraph of G by first computing a MSF of G, which it augments with additional edges so as to 2-vertex connect it. Provided that edges are sorted in order of non-decreasing cost, a MSF is found by Kruskal's algorithm, while insertions and 2-vertex connectivity queries are implemented by the incremental algorithm of [16]. We prove two properties of 2-V-CONNECT:

Lemma 2 *Algorithm* 2*-V-CONNECT produces a sparse (*$O(n)$ *size) subgraph of its input graph.*

Proof. In 2-V-CONNECT B is initialized to be a MSF of the input G, containing at most $O(n)$ edges. An edge (u, v) is inserted during the loop of 2-V-CONNECT, if and only if the unique path connecting them in B prior to insertion of (u, v) contains at least one cutvertex and/or at least one bridge (because u,v are simply connected, but not 2-vertex connected). An edge inserted during the loop cannot become a bridge, because its removal does not increase the number of connected components of B. There are at most $O(n)$ cutvertices and at most $O(n)$ bridges (edges of F), thus at the end of the loop $|B| = O(n)$. □

Lemma 3 *Algorithm* 2*-V-CONNECT produces a relaxed certificate for bottleneck* 2*-vertex connectivity of its input graph (which is not required to be* 2*-vertex connected).*

Proof. Given a MSF T, for every edge $(u, v) \notin T$ it is known that there is a $u - v$ path in T with $c(e) \leq c(u, v)$ for every edge e on this path. Since u is 2-vertex connected with v in B, there must be a set of edges in $B \setminus T$, that 2-vertex connected u and v and were selected by 2-V-CONNECT before (u, v) was examined. Since edges are examined in order of non-decreasing cost, the result follows by definition 2. □

We prove a property of relaxed certificates next:

Lemma 4 *A relaxed certificate* $B \subseteq G$ *for bottleneck* 2*-vertex connectivity of* G *is a certificate of* G *for bottleneck* 2*-vertex connectivity, if* G *is* 2*-vertex connected.*

Proof. If G is 2-vertex connected, then B is also 2-vertex connected by definition 2. Suppose that G has a bottleneck value b for 2-vertex connectivity. Suppose that $B^+ \subseteq B$ contains all edges with cost $> b$ and assume that $B \setminus B^+$ is not 2-vertex connected. Then there is a set of edges $B^- \subseteq (G \setminus B)$, with cost $\leq b$, such that $(B \setminus B^+) \cup B^-$ is 2-vertex connected. Edges of B^- cost less than the ones of B^+, thus there must exist an edge $e \in B^-$ with cost $c(e)$ less than the cost of an edge on the one of the two vertex-disjoint paths connecting the endvertices of e in B, which is a contradiction to definition 2. □

4.1 Dynamic Algorithm

We elaborate on sparsification so as to convert 2-V-CONNECT into a dynamic algorithm. We encode the weighted graph G into a *Sparsification Tree* as follows. For each edge $e \in G$ the tree has a leaf-node x_e representing the graph $G(x_e)$ of n vertices and a single edge e. Every internal node x of the tree has two child-nodes y and z, and represents the graph $G(x) = G(y) \cup G(z)$. The root-node r of the tree represents the graph $G(r) = G$. At some level of the tree there may be an odd number of nodes, thus at most one of the internal nodes may have three children. We assume a fully binary tree, without loss of generality.

Let x be a node of the Sparsification Tree, and y, z its child-nodes. We store in x the subgraph $C(x) \subseteq G(x)$, that emerges as output of algorithm 2-V-CONNECT when executed on $C(y) \cup C(z)$. For leaf-nodes we set $C(x_e) = G(x_e)$. We show that $C(r)$ is a sparse bottleneck 2-vertex connected subgraph of $G(r) = G$:

Lemma 5 *(*Monotonicity of relaxed certificates*) Let $G = G_1 \cup G_2$ be an arbitrary edge partition of a (not necessarily 2-vertex connected) weighted graph G, and $C_1 \subseteq G_1$, $C_2 \subseteq G_2$ be relaxed certificates for bottleneck 2-vertex connectivity of G_1, G_2 respectively. Then $C_1 \cup C_2$ is a relaxed certificate for bottleneck 2-vertex connectivity of G.*

Proof. For every edge $e = (u, v) \in G \setminus (C_1 \cup C_2)$ there are two internally vertex-disjoint paths either in C_1 or in C_2, connecting u to v, and for every edge e' on these paths $c(e') \leq c(e)$, by definition. □

This leads inductively to the fact that $C(r)$ is a relaxed certificate for bottleneck 2-vertex connectivity of $G(r) = G$, thus a sparse bottleneck 2-vertex connected subgraph of G by lemma 3. When the cost of an edge e is updated, the dynamic algorithm follows a path from x_e towards the root r of the sparsification tree (where x_e is the leaf-node containing e), and computes $C(x)$ for all nodes on this path. Then $C(r)$ is an updated bottleneck 2-vertex connected subgraph of G.

Theorem 2 *The bottleneck value for 2-vertex connectivity can be maintained in $O(n\alpha(n)\log n)$ time per edge cost update of the underlying graph.*

Proof. The sparsification tree is of $O(\log n)$ height. Algorithm 2-V-CONNECT outputs selected edges sorted in order of non-decreasing cost. Therefore we can merge-sort edges of relaxed certificates that are stored in the child-nodes of a node in $O(n)$ time to the cardinality of their edges. UNION-FIND data structures can be used for computing a MSF, while augmentation to 2-vertex connectivity is done via the incremental algorithm of [16]. The result stems from the fact that at most $O(n)$ edges are input to every execution of 2-V-CONNECT. □

The Sparsification Tree has at most $O(n^2)$ nodes, and computation of a $C(x)$ for every node x has $O(n\alpha(n))$ complexity. A bottom-up construction of the tree incurs $O(n^3\alpha(n))$ complexity. Furthermore the dynamic algorithm requires $O(n^3)$ space, since every node stores at most $O(n)$ edges. We were unable to generalize this technique for 3-vertex connectivity: this constitutes an open question.

An improved randomized implementation We show how it is possible to spare the $\alpha(n)$ factor from the complexity of the dynamic algorithm and its initialization, by modifying the implementation of the sparsification tree and 2-V-CONNECT. We assume that the underlying graph G is complete: if it is not, then we can add missing edges at an infinite cost, without affecting the validity of our results. Then we can use the following:

Theorem 3 *(B. Bollóbas [1], Theorem 12) The edge set of a complete graph with n vertices can be partitionied in: $\frac{n}{2}$ Hamilton paths if n is even, or $\frac{n-1}{2}$ Hamilton cycles if n is odd.*

Assume that the graph consists of vertices $v_0, \ldots, v_{n-1}$. For the case where n is odd, it is easy to partition the edge set of G in $\frac{n-1}{2}$ Hamilton cycles: for $i = 1 \ldots \frac{n-1}{2}$ the i-th cycle is defined by the vertices permutation $v_{(j \cdot i) \bmod n}$, $j = 0 \ldots n-1$. For the case when n is even we can partition the edge set of G in $\frac{n}{2}$ Hamilton paths with a slightly different practice, that for $i = 1 \ldots \frac{n}{2}$ defines the i-th path as follows: it starts from vertex v_{i-1} and the next of vertex v_j is v_{j+i}, if $j + i < n$, or $v_{(j+i-1) \bmod n}$ if $j + i \geq n$.

Correctness analysis of the described partitioning methods can be easily carried out. We let every leaf-node of the sparsification tree represent a part of the partition of the complete graph G. We observe that every part has at most $O(n)$ edges and it also is connected. Internal nodes of the tree are defined as before (representing the union of edge sets represented by their child-nodes). For each leaf-node its edges are stored in a separate balanced binary tree, so that we can always traverse them in order of non-decreasing cost.

The second modification concerns implementation of 2-V-CONNECT for producing certificates. In the description of 2-V-CONNECT a MSF is found by the linear time algorithm described in [11]. This takes $O(n)$ time with probability $1 - exp(-\Omega(n))$. Subsequently the algorithm of [16] is assured to be initialized over a connected MSF (the leaf-nodes of the sparsification tree contain connected subgraphs, cycles or paths), thus $O(1)$ amortized time is spent per query/insert operation An additional $O(\log n)$ cost is incurred due to usage of balanced binary trees in the leaf-nodes. We conclude that production of a certificate is of $O(n)$ complexity with probability $1 - exp(-\Omega(n))$ in the worst case. Thus:

Theorem 4 *The bottleneck value for 2-vertex connectivity can be maintained in $O(n \log n)$ time with probability $1 - exp(-\Omega(n))$ per edge cost modification of the underlying complete weighted graph. The dynamic algorithm can be initialized in $O(n^3)$ time with high probability.*

5 A Worst-Case Example

We examine the difficulty of dynamic maintenance of the bottleneck value of k-connectivity, for the class of algorithms that encode and maintain explicitly a bottleneck k-connected subgraph. We show that:

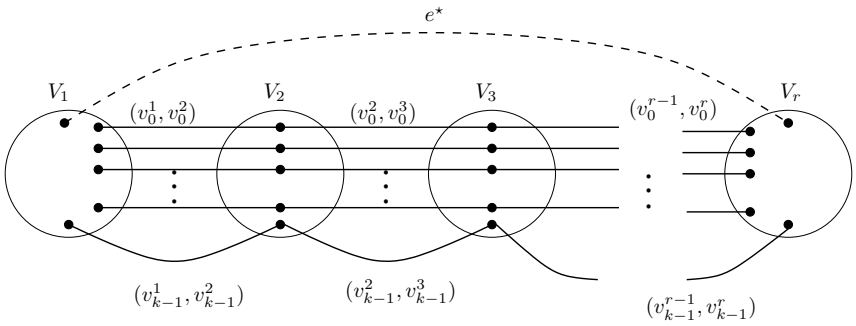

Figure 1: Worst case construction for proposition 1

Proposition 1 *There is a weighted graph G, and a sequence of edge cost updates, such that the underlying subgraph has always a unique sparse bottleneck k-connected subgraph, which differs by $\Theta(n)$ edges in between edge cost updates.*

Proof. Here is our construction of G. Partition the vertex set $V(G)$ in $r = \lfloor n/(k+1) \rfloor$ disjoint sets V_i, $i = 1 \dots r$. Place the remaining $O(k)$ vertices in any of these sets. Then $V(G) = \cup_{i=1}^{r} V_i$, where $|V_i| \geq k+1$, $|V_i| = O(k)$. Let E_i be the edge set of the clique induced by V_i. Set $w(e) = 0$ for every $e \in E_i$. Number the vertices in each subset V_i, so that v_j^i is the j-th vertex of V_i. Define $U_i = \{(v_j^i, v_j^{i+1}) | j = 0 \dots k-2\}$, $i = 1 \dots r-1$, and set $w(e) = 0$ for every $e \in U_i$. Let $C = \{(v_{k-1}^i, v_{k-1}^{i+1}) | i = 1 \dots r-1\}$ and set $w(e) = 1$ for every $e \in C$. Finally let $e^\star = (v_k^1, v_k^r)$ and $w(e^\star)$ will be defined appropriately below. The construction is depicted in fig. 1. The subgraph $H = (\cup_{i=1}^{r} E_i) \cup (\cup_{i=1}^{r-1} U_i)$ is $(k-1)$-vertex/edge connected: it consists of r cliques with at least $k+1$ vertices (thus they are k-connected), and every clique V_i is connected to the next V_{i+1} through $k-1$ edges. Thus deletion either of these $k-1$ edges, or of v_j^i, $j = 0 \dots k-2$ from any V_i disconnects H. Let $w(e^\star) = 0$. Then the subgraph:

- $H \cup \{e^\star\}$ is k-connected with bottleneck value 0.
- $H \cup C$ is k-connected with bottleneck value 1.

A sequence of updates of the cost of $e^\star$ from 0 to 2 and vice versa causes changes of $|C| = r - 1 = \Theta(n)$ edges to the bottleneck subgraph (considering $k = O(1)$). □

Every algorithm that maintains the edge set of a bottleneck k-connected subgraph has $\Omega(n)$ complexity of handling updates. The algorithm for handling $O(n)$ edge cost updates (theorem 1) for k-edge connectivity is indeed optimum up to a factor of $O(\log^4 n)$.

References

[1] B. Bollóbas. *Modern Graph Theory*. Springer-Verlag, New York Inc., 1998.

[2] P. M. Camerini. The Min-Max Spanning Tree Problem and Some Extensions. *Information Processing Letters*, 7(1):10–14, 1978.

[3] J. Cheriyan and R. Thurimella. Algorithms for Parallel k-Vertex Connectivity and Sparse Certificates (Extended Abstract). In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 391–401, 1991.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[5] Y. Dinitz and J. Westbrook. Maintaining the Classes of 4-Edge-Connectivity in a Graph On-Line. *Algorithmica*, 20(3):242–276, 1998.

[6] D. Eppstein, Z. Galil, and G. F. Italiano. *Algorithms and Theory of Computation Handbook*, chapter 8: Dynamic graph algorithms. CRC Presss, 1999.

[7] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997.

[8] H. N. Gabow and R. E. Tarjan. Algorithms for Two Bottleneck Optimization Problems. *Journal of Algorithms*, 9(3):411–417, 1988.

[9] M. R. Henzinger. A Static 2-Approximation Algorithm for Vertex Connectivity and Incremental Approximation Algorithms for Edge and Vertex Connectivity. *Journal of Algorithms*, 24(1):194–220, 1997.

[10] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48:723 – 760, July 2001.

[11] D. R. Karger, P. N. Klein, and R. E. Tarjan. A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees. *Journal of the ACM*, 42(2):321–328, 1995.

[12] E. L. Lloyd, R. Liu, M. V. Marathe, R. Ramanathan, and S. S. Ravi. Algorithmic Aspects of Topology Control Problems for Ad Hoc Networks. *Mobile Networks and Applications*, 10(1-2):19–34, 2005.

[13] G. S. Manku. A Linear Time Algorithm for the Bottleneck Biconnected Spanning Subgraph Problem. *Information Processing Letters*, 59(1):1–7, 1996.

[14] H. Nagamochi and T. Ibaraki. A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph. *Algorithmica*, 7(5&6):583–596, 1992.

[15] H. La Poutré. Maintenance of 2- and 3-edge-connected components of graphs II. *SIAM Journal on Computing*, 29(5):1521–1549, 2000.

[16] J. Westbrook and R. E. Tarjan. Maintaining Bridge-Connected and Biconnected Components On-Line. *Algorithmica*, 7(5&6):433–464, 1992.