This is Book Title This is Book Title This is Book Title

by

This is Book Author

Contributors

Amarjeet S. Bassi
The University of Western
Dept. of Chemical &
Biochemical Engineering
London ON CANADA

Farrokh Janabi-Sharifi
Ryerson University Mechanical,
Aerospace, & Industrial
Engineering Toronto ON
CANADA

Jideog Kim
Samsung Advanced Institute of
Technology Kiheung-eup
Yongin, Kyunggi

1

Approximation Algorithms for Connectivity

Ramakrishna Thurimella
University of Denver

1.1 Introduction

The edge connectivity of an undirected graph G is the minimum number of edges that must be removed to disconnect G. For example, the edge connectivity of a tree is 1, and the edge connectivity of a simple cycle is 2. Similarly, the vertex connectivity of a graph is the minimum number of vertices that must be removed to disconnect it.

It is not hard to see why these concepts are fundamental and have applications to network reliability and distributed computing. For instance, a graph could be used to model a network such as the Internet where vertices correspond to routers and edges represent the connections between them.

In distributed computing, one of the measures of complexity is the *message* complexity– the number of messages that need to be exchanged in order to compute something in a distributed manner. A key building block for these algorithms is the broadcast operation. To broadcast, each vertex after receiving, or generating in the case of the vertex that wants to broadcast, the message can send out copies of that message on every outgoing link, i.e. flood the network. This is expensive, costing m messages. To make this operation efficient, one can use a spanning tree and send the message out on every link of the tree. Since a spanning tree has only $n-1$ edges, the number of messages used is also $n-1$. For a dense graph, defined as $m = \Omega(n^2)$, the savings are substantial.

0-8493-8597-0/01/$0.00+$1.50

The problem with using a spanning tree is that it is too fragile: single vertex or edge failure causes the broadcast to fail. Hence, we seek a subgraph with high connectivity that has the fewest number of edges. Unfortunately, finding such a subgraph is NP-complete for all connectivity values greater than 1 [GJ79]. Therefore, we turn to approximation algorithms for this problem–the focus of this chapter.

Given the importance of the problem, it was extensively studied [Hoc97, NI02, Nag04]. It is not surprising that there exist many flavors of this basic problem and multitude of solutions. For example, each of the following assumptions and different combinations of them give rise to a number of variations: the presence or absence of directions/weights on edges, same connectivity between every pair of vertices–known as the *uniform* connectivity requirement–or possibly different connectivity values between vertex pairs, whether the graphs under consideration are simple or multigraphs, and finally if the special structure present for small constant values of connectivity, e.g. $k = 1$, 2, or 3, can be exploited to design efficient algorithms.

The algorithms given in this chapter are limited for the most part to simple, undirected, unweighted graphs with uniform connectivity requirements. Rest of the chapter is organized as follows. Section 1.2 defines some common graph-theoretic terms that are used in this chapter; definitions that are specific to algorithms of this chapter are deferred to the respective sections. As a warm up, we present a simple 2-approximation in Section 1.3. Section 1.4 presents a different algorithm that also achieves the same factor, but runs in linear-time. Next we show in Section 1.5 how to beat the factor of 2 for some specific values of connectivity. Section 1.6 presents the most technical result of this chapter, culminating in an algorithm that subsumes all the results from the previous sections. We end the chapter with some concluding remarks in Section 1.7.

1.2 Definitions and Notation

For the most part, we use standard graph theory notation. Refer to [Die06] for definitions not covered here. The vertex set and edge set of a graph G are denoted by V(G) and E(G), respectively. The number of vertices and the number of edges are denoted by n and m respectively. The degree of a vertex v in G is designated by $\delta_G(v)$.

We will only be dealing with undirected graphs. A *path* in a graph $G = (V, E)$ is a sequence of vertices $v_1, v_2, ..., v_p$ from V such that for all i, $1 \leq i < p$, $(v_i, v_{i+1}) \in E$. v_1 and v_p are called the *end vertices*. A *cycle* is a path that starts and ends at the same vertex. A *tree* is a connected graph that does not have any cycles. The *level* of a vertex v in a rooted tree with root r is the number of edges between the v and r. Thus, the root is at level 0, its children are at level 1 etc. A *forest* is a collection of trees. For a graph $G = (V, E)$, a *maximal spanning forest* $F = (V, E_F)$ is a subgraph such that E_F has no cycles. Furthermore, E_F is *maximal* in the sense that $E_F \cup e$ contains a cycle for all edges $e \in E - E_F$. The cycle created by the addition of a non-tree edge e to a maximal spanning forest is called the *fundamental cycle created by* e. Notice that when G is connected, F is a spanning tree and denoted by T depending on the context.

A *simple* path is a path in which if a vertex appears once, it cannot appear again. The *parent* of a vertex v in a rooted tree T with root r is the vertex that immediately follows v on the unique path from v to r in T. Two paths are *internally disjoint* if they share only the end vertices.

A *subgraph* $G_s = (V_s, E_s)$ of a graph $G = (V, E)$ is a graph whose vertex and edge sets are subsets of that of G, i.e. $V_s \subseteq V$ and $E_s \subseteq E$, and if $(x, y) \in E_s$, $x \in V_s$ and $y \in V_s$. A subgraph $G_s = (V_s, E_s)$ of $G = (V, E)$ is said to be a *spanning* subgraph if $V_s = V$ and

$E_s \subseteq E$, i.e. G_s has the same vertex set as G, but not necessarily the same edge set. If $G_i = (V, E_i)$ and $G_j = (V, E_j)$ are spanning subgraphs of $G = (V, E)$, then $G_i + G_j$ is shorthand for the spanning subgraph $(V, E_i \cup E_j)$. Similarly, if $G_i = (V, E_i)$ is a spanning subgraphs of $G = (V, E)$, then $G - G_i$ is refers to the spanning subgraph $(V, E - E_i)$.

A *connected component* of an undirected graph is a subgraph $C = (V_c, E_c)$ in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in $V - V_c$.

Given $k > 0$, a connected graph $G = (V, E)$ with at least $k + 1$ vertices is called *k-edge* (respectively, *k-vertex*) *connected* if the deletion of any $k - 1$ edges (respectively, vertices) leaves the graph connected. If a connected graph contains a single vertex whose removal disconnects the graph, then that vertex is called an *articulation point*. A graph is called *biconnected* if it has no articulation points.

By connectivity, we mean both vertex and edge connectivity. An edge $e \in E$ in a k-connected graph (V, E) is *critical*, if $(V, E - \{e\})$ is not k-connected. In a *minimally k* connected graph, every edge is critical.

For a subset of vertices $V' \subseteq V$, the subgraph G' induced by V' is (V', E') where $E' = \{(x, y) \in E \mid x \in V' \text{ and } y \in V'\}$. Also, the *neighborhood* of V', denoted $N(V')$, is the subset of vertices $\{y \mid \exists (x, y) \in E \text{ such that } x \in V' \text{ and } y \in V - V'\}$

For a connected graph $G = (V, E)$, a subset $Z \subset V$ is called a *vertex cut* if $G - Z$ has at least two connected components. The size of a vertex cut Z is defined by $|Z|$. We are generally interested in minimum size vertex cuts.

A *certificate* for the k-connectivity of G is a subgraph (V, E'), $E' \subseteq E$, that is k-connected if G is k-connected. A certificate for k-connectivity is *sparse* if it has $O(kn)$ edges. For example, a spanning tree is a sparse certificate for 1-connectivity of a connected graph. In our analysis of sparse certificate heuristics, we represent a k-connected subgraph of G that has the *optimum* number of edges by G^*.

1.3 A Simple 2-Approximation

In this section, we show how to find sparse certificates that have no more than twice the number of edges from the optimum. How do we know the optimum, given that the problem we are tackling is NP-complete? We don't. Instead, we bound the optimum from below and compute the ratio of upper and lower bounds. Obviously, closer the lower and upper bounds, the better the approximation ratio.

1.3.1 A Trivial Lower Bound

In order for a graph to be k-connected, notice that every vertex v must have degree at least k. Otherwise, i.e. if there exists a v such that $\delta(v) < k$, deleting the all edges incident on v, we can disconnect v from the rest of the graph. Similarly, $N(\{v\})$ constitutes a vertex cut whose size is less than k. From this observation, we can conclude that in order for a graph to be k-connected, each vertex must have degree at least k. In any graph, the sum of the degrees of all the vertices is twice the number of edges, as every edge is counted twice, once from each end. Therefore, in order for a graph to be k-connected, the sum of the degrees should be at least kn. In other words, every k-connected graph must have at least $\frac{kn}{2}$ edges. It is often referred to as the *degree lower bound*. This lower bound is sufficient to get a factor 2 approximation for the algorithms presented in the rest of this section.

1.3.2 Non-algorithmic Upper Bounds

Note that any minimally k-edge connected graph has at most $k(n-k)$ edges [Bol04, Mad71, Mad72]. This upper bound combined with the degree lower bound from Section 1.3.1 immediately yields a 2-approximation algorithm, as minimally k-edge connected graphs can be found in polynomial time.

Recent algorithmic work, the subject of the rest of this chapter, gives alternate, easy and efficient methods for finding a k-connected spanning subgraph whose size (i.e., number of edges) is at most kn.

1.3.3 Edge Connectivity

Consider Algorithm 1 for finding edge certificates that operates iteratively. In the ith iteration, it computes a sparse certificate for i connectivity G_i. The desired k-connected subgraph G_k is output at the end of the kth iteration.

Algorithm 1 Edge Certificate

Require: $G = (V, E)$ and an integer $k > 0$
Ensure: A sparse certificate $G_k = (V, E_k)$ for k edge connectivity
1: **procedure** EC(G,k)
2: $G_0 \leftarrow (V, \emptyset)$
3: **for** $i \leftarrow 1, k$ **do**
4: Find a maximal spanning forest F_i in $G - G_{i-1}$
5: $G_i \leftarrow G_{i-1} + F_i$
6: **end for**
7: **return** G_k
8: **end procedure**

The following theorem proves the correctness of Algorithm 1 Edge Certificate [Thu89].

THEOREM 1.1 *If G is k-edge connected, then G_k is k-edge connected.*

Proof: Denote the spanning forests found in the **for** loop of the algorithm by $F_1, F_2, \ldots, F_k$. Note that as $k > 0$ and G is connected, F_1 is a spanning tree. Hence G_k is connected. Assume for contradiction that G_k is not k-edge connected, but G is. Then, there must exist a set of edges K fewer than k in number whose removal disconnects G_k, but does not disconnect G. Let C_1 and C_2 be the two connected components of $G_k - K$. Since $|K| < k$, by the Pigeonhole Principle, there must exist an F_i, $1 \leq i \leq k$, such that F_i does not have any edges in common with K. Let e be an edge in G that has one end in C_1 and the other in C_2. Such an edge must exist as $E - K$ is connected. Now, adding e to F_i would not create a cycle, contradicting that each of the k spanning forests found line 4 of Algorithm 1 is maximal. □

1.3.4 Vertex Connectivity

While Algorithm Edge Certificate is sufficient to find edge certificates, as shown in Figure 1, it can fail for vertex certificates. This is because if a pair of vertices x and y, are connected

in maximal spanning forests F_i and F_j by paths P_i and P_j respectively, where F_i and F_j are some maximal spanning forests found in Algorithm 1, then all we can say is that P_i and P_j are edge disjoint. But for vertex connectivity, we need these paths to be vertex disjoint.

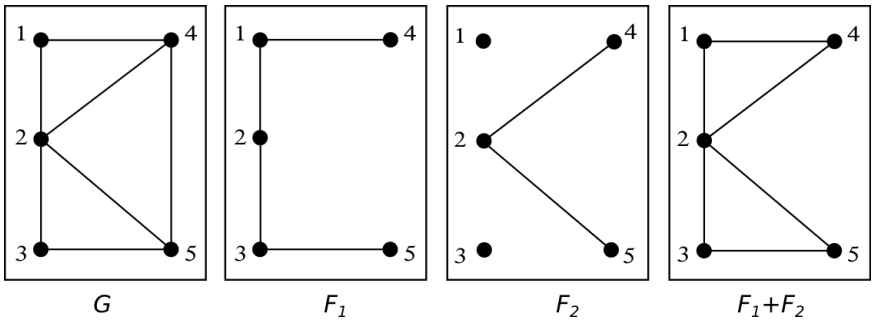

Figure 1 Algorithm Edge Certificate may not work for vertex certificates

However, by finding a specific kind of spanning forests in line 4 of Algorithm Edge Certificate, we can make the same algorithm work for vertex certificates. For example, using breadth-first spanning forests, one can show that the resulting sparse subgraph preserves vertex connectivity. We show in the rest of this section that the full power of breadth-first search is not even needed. A less restrictive form of search called *scan-first* search (SFS) is sufficient. The algorithm presented in this section is due to Cheriyan, Kao, and Thurimella [CKT93]. The earliest work that employed a similar notion is due to Doshi and Varman [DV87] though their method is limited to biconnected graphs.

Algorithm 2 Scan-First Search

Require: A connected graph $G = (V, E)$ and a root $r \in V$
Ensure: A spanning tree $T = (V, E_T)$ where $E_T \subseteq E$

```
1:  procedure SFS(G,r)
2:      Initialize all vertices of G as unmarked and unscanned
3:      E_T ← ∅
4:      mark r
5:      while there exist a marked, but unscanned vertex u do
6:          for every unmarked neighbor v of u do
7:              mark(v)
8:              add (u, v) to E_T
9:          end for                          ▷ u is now considered scanned
10:     end while
11:     return T = (V, E_T)
12: end procedure
```

Starting from a given root r or an arbitrary root, breadth-first search (BFS) explores the graph in systematic way satisfying two conditions:

1. when a vertex is being explored, all its unvisited neighbors are marked visited and added to a collection Q, and
2. the next vertex to be explored is taken from Q using the First-In-First-Out (FIFO) order.

SFS is less restrictive than BFS. In SFS, we require only the first condition. In other words, SFS chooses the next vertex to be scanned from Q, but not necessarily in the FIFO order.

The spanning tree implied by a search, whether breadth-first or scan-first, is the one that results from each vertex v, $v \neq r$, choosing the edge (v, u) where u is the vertex that was responsible for marking v as visited. Such a tree is called BFS or SFS spanning tree as the case may be. See Algorithm 2.

Therefore, every BFS spanning tree is an SFS spanning tree, but the converse need not be true. The following property of BFS and SFS is noteworthy. The nontree edges with respect to a BFS tree are *all* cross edges (i.e. no back edges) and if (x, y) is one such nontree edge then $|level(x) - level(y)| \leq 1$. The nontree edges with respect to an SFS tree are also *all* cross edges, but the level difference can be arbitrary. The reason for the absence of back edges is the first condition. If a non-tree edge (u, v), $level(u) < level(v)$, is present in a tree T, then while scanning u, its unvisited neighbor v was not added as a child in T. Hence T does not satisfy the first condition and it cannot be an SFS tree. It is also worth pointing out that not all spanning trees in which back edges are absent are not *necessarily* SFS trees, as the first condition still needs to be satisfied, i.e when a vertex is scanned *all* its unscanned neighbors must be added to the collection of vertices to be scanned. Equivalently, if a vertex u of a scan-first tree T has a non-tree edge (u, v) incident on it, then the tree edge (t, v) incident on v must be such that t appears before u in the scan order that was used to construct T.

Here are some more examples that illustrate the difference between BFS and SFS spanning trees. Consider a graph that is a simple cycle C on 7 vertices $v_0, v_2, \ldots, v_6$. Denote the edge between $v_i, v_{(i+1) \bmod 7}$ as e_i. Then, $C - \{e_3\}$ is the only BFS tree starting from vertex v_0 whereas $C - \{e_1\}$, $C - \{e_2\}$, $C - \{e_3\}$, $C - \{e_4\}$, and $C - \{e_5\}$ are all valid SFS spanning trees.

Since the only vertex marked initially is r and at the time scanning r all neighbors of r get marked, the edges incident on r get added to the SFS spanning tree. Thus

Propostion 1.2 *If r is the root of a scan-first search, the all edges incident on r are part of the spanning tree resulting from that search.*

We would like to prove the following theorem:

THEOREM 1.3 *Let G be a k-vertex connected graph and G_k be $F_1 + F_2 + \ldots + F_k$ where F_i is a* scan-first *spanning forest in $G - G_{i-1}$, $1 \leq i \leq k$, where $G_0 = (V, \emptyset)$. Then, G_k is k-vertex connected.*

Before we present the proof, we propose the following fact which follows from the previous proposition.

Propostion 1.4 *If a vertex v is used as a root in F_i, then all edges incident on it are selected by G_i.*

We prove Theorem 1.3 along the same lines as Theorem 1.1, i.e. by contradiction, while keeping in mind a key difference between edge and vertex connectivity: assuming the size

of the separator *minimal*, removal of an edge separator always results in two connected components. On the other hand, removal of a minimal vertex separator could result in more than two components. Such separators are called shredders [CT99]. This characteristic of vertex separators makes the proof considerably more technical. To keep the discussion simple, we will only prove the theorem for $k = 2$ and refer the reader to [CKT93] for the general case.

Proof of Theorem 1.3: Assume G is biconnected but G_2 is not. Then there exists an articulation point v in G_2. However, as G is biconnected, $G - \{v\}$ is connected. Therefore, there exists an edge $(x, y) \in E(G) - E(G_2)$, such that x and y are in different components in $G_2 - \{v\}$.

If v is the root of F_1, then by Proposition 1.2, all edges incident on v belong to F_1. But, since all paths between two vertices that belong to different components of $G_2 - \{v\}$ must go through v, as v is an articulation point, there is no path between x and y in $G - F_1$. But, F_2 is a maximal spanning forest in $G - F_1$ to which we can add (x, y) and not create a cycle, thus contradicting the maximality of F_2.

If on the other hand v is the not root of F_1, then let r be the root of F_1 and let C_1 be the component that contains r in $G_2 - \{v\}$. Pick any other component of $G_2 - \{v\}$ and call it C_2. Notice that $G - \{v\}$ must contain an edge (x, y), from a vertex of C_1 to a vertex C_2, as we assumed G to be biconnected. Without loss of generality, let $x \in V(C_1)$ and $y \in V(C_2)$. We now show that adding (x, y) to F_2 does not create a cycle contradicting that F_2 is maximal. Notice that the scan-first search corresponding to F_1 starts in the component C_1 at root r and scans v before it scans any vertex of C_2. If the search does not proceed this way, there would be a path from a vertex of C_1 to a vertex in C_2 that does not go through v, contradicting our assumption that v is an articulation point in G_2. When scanning v, the spanning tree F_1 selects all edges (v, w) such that $w \in V(C_2)$. In other words, there is no path between x and y in $G - F_1$, i.e. adding (x, y) to F_2 does not create a cycle. A contradiction. □

1.3.5 Time Complexity

Algorithm Edge Certificate is very easy to implement, as formalized in the theorem below:

THEOREM 1.5 *Given G and G_{i-1}, each maximal spanning forest F_i in $G - G_{i-1}$, $1 \leq i \leq k$, where $G_0 = (V, \emptyset)$ can be found in linear time sequentially. Therefore, a sparse certificate for k-edge connectivity G_k can be found in $O(k(m + n))$ time.*

Proof: Discard the edges of G_{i-1} from G and run any standard spanning forest algorithm [CLRS09]. Since finding a maximal spanning forest takes time linear in the size of the graph [CLRS09], the theorem follows. □

Algorithm Scan-First Search can be implemented to run in linear time, implying that a sparse certificate of k-connectivity can be found in $O(kn)$ time.

THEOREM 1.6 *Given G and G_{i-1}, each maximal* scan-first *spanning forest F_i in $G - G_{i-1}$, $1 \leq i \leq k$, where $G_0 = (V, \emptyset)$ can be found in linear time sequentially. Therefore, a sparse certificate for k-vertex connectivity G_k can be found in $O(k(m + n))$ time.*

Proof: Discard the edges of G_{i-1} from G and run any linear-time breadth-first algorithm [CLRS09]. Since every BFS spanning forest is an SFS spanning forest, the theorem follows. □

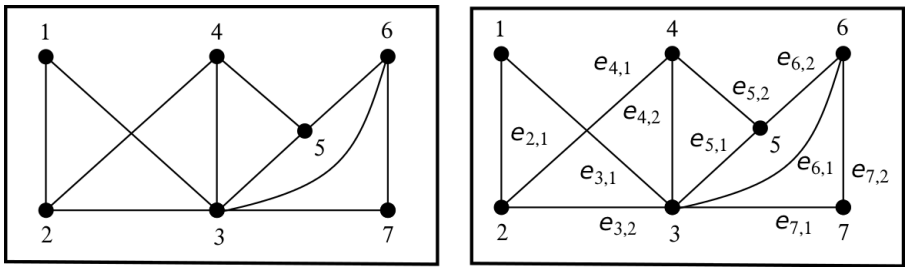

Figure 2 MA labeling example

Note that there are no efficient methods to execute BFS in parallel. Fortunately, scan-first search lends itself to an efficient parallel implementation [CKT93]:

THEOREM 1.7 *For graph G with m edges and n vertices, an SFS forest can be found in $O(\log n)$ time using using $C(n,m)$ processors on a CRCW PRAM, where $C(n,m)$ is the number of processors required to compute a spanning tree in each component in $O(\log n)$ time.*

Proof: One easy way to find an SFS tree T in parallel is to first find an arbitrary spanning tree T' rooted at some vertex r and rearrange the edges so that it becomes an SFS tree. First, label each vertex of T' with respect to preorder labeling. To obtain an SFS tree T from T', make each v, $v \neq r$, choose one edge (v,u) where u is a neighbor of v with the lowest preorder label. To see that T is indeed an SFS tree, treat the preorder labels as the order in which the vertices are scanned in some scan-first search. In order for an order to be a valid SFS order, when a vertex needs to be chosen to be scanned, it should be selected from the neighborhood of the vertices already scanned. Hence preorder is a valid scan-first order. With this interpretation, if a vertex v has a non-tree edge (v,w) incident on v in T, then the preorder label of the parent of w in T is less than that of v, consistent with the meaning of scanning a vertex in that w should be attached to the first neighbor of w marks it.

The complexity bound follows because the preorder numbers and the minimum labeled neighbor can be found in $O(\log n)$ time using $\frac{(n+m)}{\log n}$ processors [JÓ92]. The spanning tree computation dominates the resource bounds as the processor bound is $C(n,m) = \Omega(\frac{(n+m)}{\log n})$ for $O(\log n)$ parallel time. □

1.4 A Linear-Time 2-Approximation

In this section, we present an algorithm that computes a sparse k-connected spanning subgraph that is within a factor 2 from the optimal in a *single scan* of the graph. This elegant algorithm is due to Nagamochi and Ibaraki [NI92].

We need a few definitions first. Let V_i denote $\{v_1, v_2, \ldots, v_i\}$. An ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of vertices in G is called a maximum adjacency ordering (MA ordering, for short) if for all i, $2 \leq i < n$, v_{i+1} is one of the vertices from $N(V_i)$ that has the highest degree in the subgraph induced by $V_i \cup N(V_i)$.

An MA ordering can be found sequentially by starting with an arbitrary edge and des-

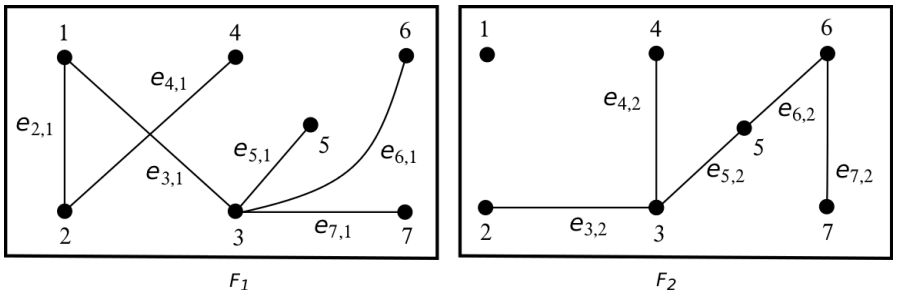

Figure 3 Edge Partitions

ignating its ends points as v_1 and v_2, thus defining V_2. Next, v_3 is selected from $N(V_2)$ based on the number connections to the vertices in V_2. In general, V_i is extended to V_{i+1} by adding v_{i+1} where v_{i+1} is any vertex from $N(V_i)$ that has the maximum number edges to the vertices in V_i. Figure 2 on the left illustrates MA labeling.

Next, we extend the MA labeling to edges. For each $2 \leq i \leq n$, assume there are j edges between $\{v_1, v_2, \ldots, v_{i-1}\}$ and v_i. Consider the neighbors of v_i in V_{i-1} in the increasing order of σ labeling and assign labels $e_{i,1}, e_{i,2}, \ldots, e_{i,j}$. Figure 2 on the right shows an example edge labeling. For instance in Figure 2, the vertex labeled 4 has two edges to $V_3 = \{v_1, v_2, v_3\}$, to v_2 and v_3. Edge labeling would label (v_4, v_2) and (v_4, v_3) as $e_{4,1}$ and $e_{4,2}$ respectively.

Edge labeling defines a partition on E. We could use the second component of the subscript of an edge label as the partition number. Specifically, define a partition of E to be $(F_1, F_2 \ldots, F_n)$ where $F_i = \{e_{2,i}, e_{3,i}, \ldots, e_{n,i}\}$ for $i = 1, 2, \ldots, n$
Note that some partitions F_i may be empty. See Figure 3 for an example. As before, define G_i to be $F_1 + F_2 + \cdots + F_i$ for $1 \leq i \leq n$.

THEOREM 1.8 *Each (V, F_i), $1 \leq i \leq n$, is a maximal scan-first spanning forest in $G - G_{i-1}$.*

Proof: For every vertex v, there can be at most one edge $e_{v,i}$ present in F_i. Furthermore, there must be a unique vertex in every connected component C of F_i that does not have such an edge–the vertex with the smallest σ number in that component. Assume that there are n_c vertices in C. As every vertex v, with the exception of one, has a unique $e_{v,i}$ incident on it, C has $n_c - 1$ edges. Since any connected graph with n_c vertices and $n_c - 1$ edges is necessarily a tree, F_i is a spanning forest.

It remains to show that F_i is a scan-first spanning forest. The proof is similar to that of Theorem 1.7. Notice that the σ order is a valid scan-first order. This is because in SFS, starting from an arbitrary vertex, the next vertex to be scanned is chosen from the neighborhood of the vertices that have already been scanned and all its unmarked neighbors are marked. Interpreting the σ labeling as the scan-first order in $G - G_{i-1}$, the tree implied in each component of $G - G_{i-1}$ is an SFS tree. Therefore, F_i is a maximal scan-first spanning forest in $G - G_{i-1}$. □

Combining Theorem 1.3 with the theorem above, we have

Corollary 1.9 *For any k, $F_1 + F_2 + \cdots + F_k$ is a sparse certificate for k-connectivity.*

1.4.1 Time Complexity

It turns out that MA labeling and the corresponding edge labeling can be computed in almost linear time, specifically in $O(m + n \log n)$ time [Nag04]. Note that this bound is independent of k.

THEOREM 1.10 *For graph G with m edges and n vertices, MA labeling can be computed in $O(m + n \log n)$ time.*

Proof: We will sketch an algorithm and allude to its implementation using Fibonacci heaps. (See Chapter 19 from [CLRS09] for a detailed description of Fibonacci heaps.) The algorithm, starting from an arbitrary vertex, scans one vertex at a time until all n vertices are processed. Assume inductively i vertices have been processed and an integer $d(v), 0 \leq d(v) \leq n-1$, is associated with each of the $n-i$ unprocessed vertices v. The integer $d(v)$ represents the number of v's processed neighbors. The next vertex to be processed, $(i+1)$th vertex, is picked from one of the unprocessed vertices that has a maximum d value. Say, x is one such vertex. Processing vertex x entails incrementing the d value for all unprocessed neighbors of x and designating x as processed. Increment operation is performed once for each edge. Moving a vertex from unprocessed to processed involves using the `delete max` operation. This is done at most $n-1$ times. Using Fibonacci heaps `increment key` can be performed in $O(1)$ amortized time and `delete max` takes $O(\log n)$ time. From these observations, the theorem follows. □

The above algorithm, while efficient, is inherently sequential as it processes one vertex at a time. However, Algorithm Scan-First Search lends itself to an efficient parallel implementation as proved in Theorem 1.7.

1.5 Beating 2 For the Edge Case

Khuller and Vishkin gave the first approximation algorithm for the special case of $k = 2$ with a performance ratio less than 2 [KV94]. Their algorithm is based on depth-first search and has a performance guarantee of $\frac{3}{2}$. A simple generalization of their algorithm has a performance guarantee of $2 - \frac{1}{k}$ for all k. Unfortunately, for higher values of k, this expression approaches 2.

In this section we provide a different algorithm due to Khuller and Raghavachari [KR96]. This is the first algorithm that achieves an approximation factor 1.85 for *all* values of k, for unweighted k-edge connectivity. For smaller values of k, their bounds are actually better: 1.66, 1.75, and 1.733 for $k = 3$, 4, and 5 respectively. The structure of this algorithm is similar to the one shown in the previous sections where the connectivity of the solution is increased incrementally in stages.

1.5.1 KR Algorithm

For the sake of clarity, we assume that k is even; if k is odd, first find a sparse $(k-1)$-connected spanning subgraph G_{k-1} by using the algorithm given in this section and add to it a maximal depth-first search (DFS) spanning forest F_i from $G - G_{k-1}$, thus obtaining a k-edge connected spanning subgraph G_k of G. Proofs from this section can be used for the odd case with minor modifications.

We use the following notation throughout this section:

Definition 1.5.1 *Let e be an edge in a DFS spanning forest F_{2i+1} with x and y as its ends*

Algorithm 3 KR Edge Certificate

Require: $G = (V, E)$ and an even integer $k > 0$
Ensure: A sparse certificate $G_k = (V, E_k)$ for k edge connectivity

```
1: procedure KR(G,k)
2:     G_0 ← (V, ∅)
3:     for i ← 0, k/2 − 1 do
4:         Find a maximal DFS forest F_{2i+1} in G − G_{2i} with arbitrary root(s)
5:         G_{2i+1} ← G_{2i} + F_{2i+1}
6:         Post-order label each edge of F_{2i+1}
7:         Build a forest F_{2i+2} as follows. Process each edge e = (x, y), level(x) < level(y),
8:           of F_{2i+1} in post-order and add b_e (see Definition 1.5.3) to F_{2i+2} if the number
9:           of edge-disjoint paths between x and y in G_{2i+1} + F_{2i+2} is less than 2i + 2.
10:          (Note that the edge (x, y) constitutes a path by itself.)
11:        G_{2i+2} ← G_{2i+1} + F_{2i+2}
12:    end for
13:    return G_k
14: end procedure
```

points. We will assume that x is the parent of y in F_{2i+1}.

Definition 1.5.2 *Let e be a critical edge in G_{2i+1}, i.e. G_{2i+1} is $2i+1$ edge connected but $G_{2i+1} - \{e\}$ is not, and let K be a cut of size $2i$ in $G_{2i+1} - \{e\}$. Since G is connected, G_{2i+1} is connected for all i as it contains F_1 which is a spanning tree of G. Denote the two connected components that result when $K \cup \{e\}$ is deleted from G_{2i+1} by C_1 and C_2 where $x \in C_1$ and $y \in C_2$.*

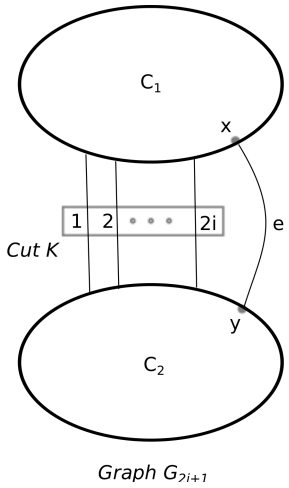

Figure 4 Illustration of Definition 1.5.2

See Figure 4 for an illustration of this definition.

Definition 1.5.3 *With every critical edge $e \in F_{2i+1}$, associate a back edge $b_e \in G - G_{2i+1}$ that satisfies two properties:*

- *the fundamental cycle C_e created by b_e in F_{2i+1} contains e, and*
- *of all fundamental cycles that contain e, C_e contains a vertex v with a smallest $level(v)$ value.*

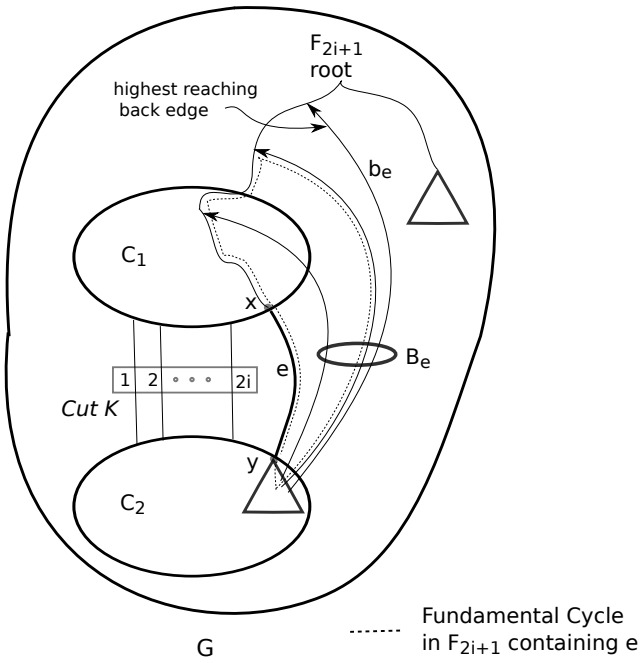

Figure 5: Illustration of Definitions 1.5.3 and 1.5.4

Definition 1.5.4 *Denote the set of edges that connect a vertex in C_1 to a vertex in C_2, excluding $K \cup \{e\}$, by B_e.*

In the following lemma, assume that G_{2i} is a $2i$ connected spanning subgraph of a k-connected graph G, for $k > (2i+1)$ and $i \geq 0$. Let $G_{2i+1} = G_{2i} + F_{2i+1}$ where F_{2i+1} is a maximal DFS spanning search forest in $G - G_{2i}$.

LEMMA 1.1 If G_{2i+1} has an edge cut K' of size $2i+1$, then there is exactly one edge e from F_{2i+1} in the cut, i.e. $K' \cap E(F_{2i+1}) = \{e\}$.

Proof: We first show that K' must contain at least one edge from F_{2i+1}. Consider the two connected components C_1 and C_2 that result when K' is deleted from G_{2k+1}. Since G is k connected for some $k > (2i+1)$, there exists an edge (a, b) in $G - G_{2i+1}$ that connects a vertex from C_1 to a vertex in C_2. Since F_{2i+1} is maximal, every edge in $G - G_{2i+1}$,

including (a, b), creates a cycle when added to F_{2i+1}. Let the path between a and b in F_{2i+1} be P. The path P cannot survive when the cut K' is deleted from G_{2i+1}, for otherwise C_1 would be connected to C_2 by P in $G_{2i+1} - K'$. Therefore, K' should contain at least one edge from F_{2i+1}. Denote K' as $K \cup \{e\}$ where $e \in F_{2i+1}$. To finish the proof, notice that K cannot contain any edges from F_{2i+1}, as it would mean that G_{2i} contains a cut smaller than $2i$. □

Corollary 1.11 *Let e, $e \in F_{2i+1}$, be a critical edge G_{2i+1} that belongs to a cut K of size $2i + 1$. Then, the edge b_e associated with each critical edge $e = (x, y)$ goes between a vertex from the subtree rooted at y, i.e. a descendant of y, and a vertex that is on the path from x to the root of F_{2i+1}, i.e. an ancestor of x.*

Proof: From the maximality of F_{2i+1}, we know that every edge from $G - G_{2i+1}$, including b_e, creates a cycle in F_{2i+1}. As F_{2i+1} is a DFS spanning forest, all such edges are back edges. From the previous lemma, we know that there is only one forest edge in K. Therefore, all edges from $G - G_{2i+1}$ that go between C_2 and C_1 are back edges that connect a descendant of y to an ancestor of x. □

THEOREM 1.12 *Let G be a graph which is at least k-edge connected for some $k > 0$. Then, Algorithm* KR Edge Certificate *with input G outputs a subgraph G_k of G that is k-edge connected.*

Proof: We prove this by induction on k. The theorem clearly holds for $k = 1$. Assume it holds for all values j, $j < k$. We will show that it holds for $j + 1$ as well. If j is even, then F_{j+1} is a maximal spanning forest in $G - G_j$. In this case, the fact that $G_j + F_{j+1}$ is $j + 1$ connected can be proved in a manner similar to the proof of Theorem 1.1.

Next consider the case when j is odd. Let $j = 2x + 1$ for some x. Then, F_{2x+2} is the set of back edges b_e for each critical edge e added in lines 7-9. We claim that every edge cut K in $G_j + F_{2x+2}$ is of size at least $2x + 2$. Note that the presence of a cut of size less than $2x + 1$ would violate the inductive hypothesis. If a cut K of size $2x + 1$ were to exist, one of the cut edges, say e, must come from F_{2x+1} by Lemma 1.1. This edge e would be a critical edge in G_j as it is part of a j cut. Therefore, the algorithm would add b_e to F_{2x+2} and make e non-critical. Hence no such cut K can exist in $G_j + F_{2x+2}$. □

1.5.2 Approximation Guarantee

Let b_e be a back edge added in lines 7-9 to F_{2i+2} corresponding to the critical edge e in G_{2i+1} and recall that B_e (see Definition 1.5.4) is the set of back edges whose fundamental cycles that contain e.

Remark: It is important to note that F_{2i+2} changes as back edges associated critical edges are added and it is this updated F_{2i+2} that is used when the criticality of a tree edge is checked in $G_{2i+1} + F_{2i+1}$ in lines 8-9.

Let d be another tree edge detected to be critical in the post-order corresponding to which b_d is added from the set of edges B_d whose fundamental cycles contains d. Then,

LEMMA 1.2 $B_d \cap B_e = \emptyset$.

Proof: Assume without loss of generality that e comes before d in the post-order traversal. Let $d = (s, t)$ and $e = (x, y)$. Then, by Corollary 1.11 the edges of B_d (resp. B_e) are all

back edges and have one end point in the subtree rooted at t (resp. y). Therefore, if d is not an ancestor of e, the lemma follows immediately. Let us now consider the case when d is an ancestor of e. Assume, for contradiction, that there is an edge f that is in both B_d and B_e. Note that adding f to G_{2i+1} makes both d and e non-critical. But, when e is processed in lines 7-9, either f is added or another edge whose span includes the span of f is added, rendering d non-critical, contradicting the assumption that d was critical at the time it was examined in the updated G_{2i+1}. □

Recall that G^* denotes a subgraph of G with the optimum number of edges. The following lower bound is crucial for our analysis:

LEMMA 1.3 For any phase i, $0 \leq i < \frac{k}{2}$, $|G^*| \geq (k-2i)|F_{2i+2}|$.

Proof: At the beginning of phase i, G_{2i} is $2i$ connected. By the end of the iteration, the algorithm increases the connectivity by 2 and outputs G_{2i+2}. Corresponding to every edge $b_e \in F_{2i+2}$, there exists a critical edge e in G_{2i+1} that is part of a $2i+1$ cut $K \cup \{e\}$. Since G^* is k-connected, every cut is of size at least k. As $B_e \cup K \cup \{e\}$ is a cut that disconnects C_1 from C_2, $|B_e \cup K \cup \{e\}| \geq k$ (see Definition 1.5.2 for the definition of C_1 and C_2). That is, $|B_e \cup \{e\}| \geq k - 2i$. Furthermore, corresponding to every edge b_e of F_{2i+2} there is an edge e, $e \in F_{2i+1}$, and a set of back edges B_e, $B_e \subseteq G - G_{2i+1}$. By Lemma 1.2, the sets corresponding two distinct edges from F_{2i+2} do not intersect with each other. Therefore G^* must contain at least $(k-2i)|F_{2i+2}|$ edges. □

Using the bounds from the previous lemma, we can finally establish

THEOREM 1.13 *The ratio of the number of edges of G_k output by Algorithm KR Edge Certificate to that of G^* is at most $\frac{(3+\ln 2)}{2}$.*

Proof: As in the algorithm, we restrict our attention to even k; minor adaptation of the proof is required for the odd case. Note that since F_{2i+1} is a forest, it has at most n edges. This holds F_{2i+2} as well. Also, from the degree lower bound given Section 1.3.1, we know that $|G^*| \geq \frac{kn}{2}$. Thus,

$$\sum_{i=0}^{k/2-1} \frac{(|F_{2i+1}| + |F_{2i+2}|)}{|G^*|} \leq \frac{\Sigma_{i=0}^{k/2-1}|F_{2i+1}| + \Sigma_{i=0}^{(k/4-1)}|F_{2i+2}| + \Sigma_{i=k/4}^{(k/2-1)}|F_{2i+2}|}{\max\{kn/2, \max_i\{(k-2i)|F_{2i+2}|\}\}}$$

$$\leq \frac{3kn/4 + \Sigma_{i=0}^{(k/4-1)}|F_{2i+2}|}{\max\{kn/2, \max_i\{(k-2i)|F_{2i+2}|\}\}}$$

$$\leq \frac{3kn/4}{kn/2ll} + \frac{\Sigma_{i=0}^{(k/4-1)}|F_{2i+2}|}{\max_i\{(k-2i)|F_{2i+2}|\}}$$

$$\leq \frac{3}{2} + \frac{1}{2}\sum_{i=0}^{(k/4-1)} \frac{1}{(k/2-i)} \tag{1.1}$$

Using variable substitution we can rewrite the second term as

$$= \frac{3}{2} + \frac{1}{2}\sum_{x=k/4+1}^{k/2} \frac{1}{x}$$

Since $\frac{1}{x}$ is a monotonically decreasing function, we can upper bound the sum using an integral (see Appendix A.2 from [CLRS09]) by reducing the bottom limit by 1:

$$\leq \frac{3}{2} + \frac{1}{2}\int_{x=k/4}^{k/2} \frac{1}{x} = \frac{3}{2} + \frac{1}{2}(\ln(\frac{x}{2}) - \ln(\frac{x}{4}))$$

$$= \frac{3}{2} + \frac{\ln 2}{2} < 1.85$$

□

As mentioned before this analysis assumes that k is even. By substituting $k = 4$ in (1.1), one can see that the bound is actually 1.75, which is somewhat better than 1.85. By a different analysis, it can be shown when k is 3 and 5, the approximation factors are 1.66 and 1.733 respectively, again better than 1.85.

1.5.3 Time Complexity

It turns out that analyzing the running time of KR Edge Certificate is rather easy. There are $\frac{k}{2}$ iterations of the *for* loop. In each loop, we find a DFS spanning forest which takes linear time [CLRS09]. Similarly, post-order labeling can be performed in linear time [CLRS09]. Rest of the loop involves checking whether a given edge e is critical and finding b_e if it is determined that e is indeed critical. Karzanov and Timofeev show that in $O(n^2 i)$ time we can find all mincuts of an i-connected graph [KT86]. Furthermore, they also show how to store all mincuts using a compact tree-like representation. Given this representation, it is easy to check whether there exists a cut of size less than $2i + 2$ between x and y in $G_{2i+1} + F_{2i+2}$ in constant time. This will reveal whether e is critical. If it is, we can perform a DFS of the tree T containing the edge e in linear time and see which back edge b_e from the subtree rooted at y comes closest to the root of T. Therefore, the dominant step is the construction of the cut representation of Karzanov-Timofeev which takes $O(n^2 i)$ time. Summing this over $\frac{k}{2}$ iterations, we get a time bound of $O(k^2 n^2)$. Note that as back edges b_e are added, the cut in which b_e participates definitely gets destroyed. Also some other (2i+1)-cuts might get destroyed by the addition of b_e. One assumption that is made in [KR96], though not explicitly stated, is that the Karzanov-Timofeev cut representation can be updated in linear time under edge additions.

1.6 Approximating Minimum-Size Spanning Subgraphs via Matching

We conclude this chapter with the best known algorithm for finding minimum-size k-connected spanning subgraphs. The algorithm presented in this section is extremely simple and for the vertex connectivity case, it meets or beats all other algorithms for all values of k. For undirected edge connectivity, it matches or improves the known bounds for values of $k \geq 3$. In particular, the algorithm presented in this section, due to Cheriyan and Thurimella [CT99], finds a k-vertex connected spanning subgraph of k-vertex connected graph, directed or otherwise, whose size is at most $1 + \frac{1}{k}$ times the optimal. For edge connectivity, the approximation factors are $1 + \frac{2}{(k+1)}$ and $1 + \frac{4}{\sqrt{k}}$ for undirected and directed graphs respectively.

For clarity of exposition, we limit our discussion to the undirected, vertex case. Before we present the algorithm, the following background is required on generalized *matching*. Given a graph $G = (V, E)$, a matching M, $M \subseteq E$, is set of pairwise *non-adjacent* edges, i.e. no

two edges share a common vertex. A *maximum matching* is a matching M that contains the largest number of edges. Notice that in any matching M, the degree of any vertex v is at most 1. The maximum matching can be generalized wherein the degree of each vertex is at most d. Additionally, the degree requirement can be made non-uniform, i.e. each vertex v can have its own degree constraint ranging from 1 to $\delta(v)$. This generalization is known as the *b-matching* problem.

A b-matching M of $G = (V, E)$ is really a subgraph with *most* number edges satisfying certain degree requirements. In particular, we are interested in a b-matching M where $\delta_M(v) \leq \delta_G(v) - (k-1)$. Denote $E - M$ by $\widetilde{M}$. Then $(V, \widetilde{M})$ is another subgraph of G with the *least* number edges wherein $\delta_{\widetilde{M}}(v) \geq (k-1)$ for each v. Denote this subgraph by $\widetilde{M}$.

Algorithm 4 CT Vertex Certificate

Require: An integer $k > 0$ and k-vertex connected graph $G = (V, E)$
Ensure: A sparse certificate $G_k = (V, E_k)$ for k vertex connectivity

```
1: procedure CT(G,k)
2:     Find M̃, a smallest subgraph of G where δ(v) ≥ (k − 1) for each v ∈ G
3:     F ← ∅
4:     for each edge e = (u, v) in G − M̃ do
5:         if (no. of vertex disjoint paths between u and v, including uv, in M̃ + F is ≤ k)
6:         then F ← F ∪ {e}
7:     end for
8:     return G_k = M̃ + F
9: end procedure
```

Let us now analyze the algorithm above to see how many edges G_k contains in relation to the best possible G^*. We first show that $|F| < (n-1)$, a fact that follows easily from a theorem of Mader [Mad71, Mad72, Theorem 1]. (For an English translation of the proof of Mader's theorem see Lemma I.4.4 and Theorem I.4.5 in [Bol04].)

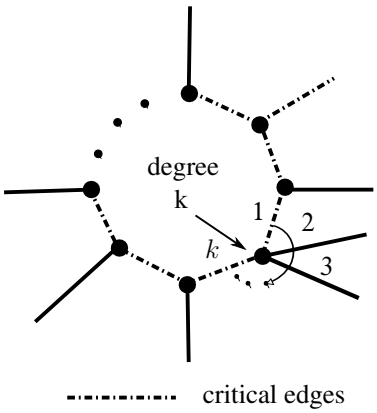

Figure 6: Illustration of Mader's Theorem

THEOREM 1.14 **(Mader [Mad72, Theorem 1])** *In a k-vertex connected graph, a cycle consisting of critical edges must be incident to at least one vertex of degree k.*

LEMMA 1.4 $|F| \leq (n-1)$

Proof: We claim that F cannot have any cycles. Assume otherwise. Since all edges of F are critical in $\widetilde{M} + F$, from Mader's theorem we know that this cycle must have a vertex v such that $\delta_{\widetilde{M}+F}(v) = k$. See Figure 6. Equivalently, $\delta_{\widetilde{M}}(v) < (k-1)$, contradicting that $\widetilde{M}$ is a subgraph of G in which every vertex has degree at least $(k-1)$. □

We now establish the performance guarantee.

THEOREM 1.15 $\frac{|\widetilde{M}+F|}{|G^*|} < 1 + \frac{2}{k}$

Proof: From the degree lower bound given in Section 1.3.1, we know that $|G^*| \geq \frac{kn}{2}$. Therefore

$$\frac{|\widetilde{M}+F|}{|G^*|} \leq \frac{|\widetilde{M}|}{|G^*|} + \frac{n-1}{(kn/2)} < \frac{|\widetilde{M}|}{|G^*|} + \frac{2}{k}$$

Since G^* is a subgraph in which every vertex v has $\delta(v) \geq k$, any subgraph of G that has the smallest number edges in which $\delta(v) \geq (k-1)$ will have no more than $|G^*|$ edges. Therefore, the first term is at most 1, thus establishing the theorem. □

CT Algorithm performs significantly better and can be shown to have to an approximation ratio of $1 + \frac{1}{k}$ as mentioned at the beginning of this section. To show this improved bound, notice that there is some slack in our analysis in the proof of our theorem where we argue that the first term is at most 1, i.e. when we compare the degrees of vertices in G^* to those in $\widetilde{M}$.

It turns out that $|\widetilde{M}| \leq |G^*| - \frac{n}{2}$. Using this fact in conjunction with Lemma 1.4, we see that $|\widetilde{M} + F| < |G^*| + \frac{n}{2}$ which implies an approximation ratio of $1 + \frac{1}{k}$.

The fact $|\widetilde{M}| \leq |G^*| - \frac{n}{2}$ follows quite easily when G^* has a perfect matching M, because in that case $G^* - M$ would be a subgraph in which every vertex v has $\delta(v) \geq k-1$. Proof of this fact is considerably technical when G^* does not have a perfect matching and is omitted here.

1.6.1 Time Complexity

Finding $\widetilde{M}$ is the same as finding a b-factor and removing the edges found, leaving a subgraph in which every vertex v satisfies $\delta(v) \geq k-1$. Gabow and Tarjan [GT91] give a $O(m^{1.5} \log^2 n)$ algorithm for finding a b-factor.

To find a set of critical edges F whose addition to $\widetilde{M}$ would yield a k-connected subgraph, examine each edge $e = (u, v)$ of $G - \widetilde{M}$ in an arbitrary order (line 4). To see whether the if condition is satisfied in line 5, attempt to find $k+1$ internally disjoint (vertex or edge as the case may be) paths between u and v. Each path can be found in linear time by an augmenting path algorithm [CLRS09]. Therefore each iteration of the for loop takes km time for a total of km^2 time. At termination, the subgraph $\widetilde{M} + F$ is k-vertex connected, and every edge $e \in F$ is critical.

The running time of finding F can be improved to $O(k^3 n^2)$ by first executing a linear-time preprocessing step to compute a sparse certificate of G for k-vertex connectivity and using that sparse certificate find k disjoint paths.

1.7 Conclusion

Approximation algorithms for graph connectivity are surprisingly elegant and have many useful applications. In this chapter, we have presented heuristics for finding k-connected spanning subgraphs of k-connected graphs. These polynomial-time heuristics are reasonably efficient and have *provable* performance guarantees. In some cases the approximation ratios come very close to the optimal. For example, the algorithm presented in the last section is provably within 5% from the optimal, i.e. $< 1.05 * \mathcal{OPT}$, if one is interested in finding a k-vertex connected subgraph of a k-vertex connected graph for $k = 20$.

References

1. B. Bollobás. *Extremal graph theory*. Dover Books on Mathematics. Dover Publications, 2004.
2. J. Cheriyan, M. Kao, and R. Thurimella. Scan-first search and sparse certificates: An improved parallel algorithms for k-vertex connectivity. *SIAM J. Comput.*, 22(1):157–174, 1993.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
4. J. Cheriyan and R. Thurimella. Fast algorithms for k-shredders and k-node connectivity augmentation. *J. Algorithms*, 33(1):15–50, 1999.
5. R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2006.
6. K.A. Doshi and P.J. Varman. Optimal graph algorithms on a fixed-size linear array. *IEEE Transactions on Computers*, C-36(4):460 –470, April 1987.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
8. H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38:815–853, 1991.
9. D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997.
10. J. JáJá. *An introduction to parallel algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1992.
11. S. Khuller and B. Raghavachari. Improved approximation algorithms for uniform connectivity problems. *J. Algorithms*, 21(2):434–450, 1996.
12. A. V. Karzanov and E. A. Timofeev. Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. *Cybernetics*, 22:156–162, 1986. Translated from Kibernetika, 2:8-12, 1986.
13. S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, March 1994.
14. W. Mader. Minimale n-fach kantenzusammenh ngende graphen. *Math. Ann.*, 191:21–28, 1971.
15. W. Mader. Ecken vom grad n in minimalen n-fach zusammenh ngenden graphen. *Arch. Math. (Basel)*, 23:219–224, 1972.
16. H. Nagamochi. Graph algorithms for network connectivity problems. *J. of the Op. Res.*, 47(4):199–223, 2004.
17. 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.
18. H. Nagamochi and T. Ibaraki. Graph connectivity and its augmentation: applications of ma orderings. *Discrete Applied Mathematics*, 123(1-3):447–472, 2002.

19. R. Thurimella. *Techniques for the Design of Parallel Graph Algorithms.* PhD thesis, The University of Texas at Austin, 1989.