Sparse Certificates and Scan-First Search

1 Introduction

An algorithm for the parallel construction of $k$-connectivity certificates for graphs, Scan-First Search, was introduced by Joseph Cheriyan, Ming-Yang Kao, and Ramakrishna Thurimella in their paper *Scan-First Search and Sparse Certificates: An Improved Parallel Algorithm for $K$-Vertex Connectivity*.[2] The following is an attempt to clarify and simplify the concepts and proofs presented in that paper.

2 Sparse Certificates

If we are given a graph $G = (V, E)$, there exists some subset of edges $E' \subseteq E$ such that the subgraph $G' = (V, E')$ is $k$-vertex connected if and only if $G$ is $k$-vertex connected. This subset is considered a certificate for the $k$-connectivity of the graph $G$. For a graph with $n$ vertices, a $k$-connectivity certificate for it is considered sparse if it contains no more than $kn$ edges. Remember that a graph is $k$-vertex connected if it is possible to remove any set of $< k$ edges from the graph and still maintain connectivity. The graph $H_2$ from Figure 1 shows a sparse certificate for biconnectivity on the graph $G$.

Testing the $k$-vertex connectivity of a graph can be done in two stages. The first stage is determining a certificate for the $k$-connectivity of the graph. The second stage is to test the $k$-connectivity of our certificate. The Scan-First Search algorithm improves the running time of building a sparse certificate for $k$-connectivity using the parallel computation model. There are several known algorithms for testing the $k$-connectivity of a certificate once we have obtained one.[3][5]

The scan-first search algorithm is proposed to develop $k$-connectivity certificates in parallel in $O(k \log n)$ time, using $C(n, m) = \Omega((n + m)/\log n))$ processors. This is an improvement over both the distributed approach ($O(k \cdot n \cdot \log^3 n)$), and the sequential approach ($O(k(n + m))$).

3 Scan-First Search

We can find a sparse certificate for $k$-connectivity by iteratively running scan-first search $k$ times on subgraphs of our input graph. Our input is a graph $G = (V, E)$ and a root vertex $r$. For each iteration of scan-first search, we first compute a spanning tree $T$ of our input graph $G$, and assign a preorder numbering to all the vertices, which we will use as our scanning order. From our root $r$, we first scan $r$, which involves marking all its neighbouring vertices.

All previously unmarked vertices constitute the end-point of an edge from the currently scanned vertex, so if we start from some vertex $v$, and it has neighbours $w$ and $x$, then if both $w$ and $x$ are
unmarked, we create the edges \((v, w)\) and \((v, x)\) and add them to our output tree \(T'\). If either \(w\) or \(x\) was previously marked, we do not add the edge that includes that vertex to \(T'\). With these new edges in \(T'\), we move to the next vertex with the lowest preorder number to scan, which involves continuously marking previously unmarked vertices and adding the edges from the current vertex to these vertices to our output tree.

We use scan-first-search to generate certificates for \(k\)-connectivity by running it for \(k\) iterations. We will see shortly why this works. What is important to note moving forward is that for each edge added to some output tree \(T'\) in each iteration, we remove the edges from the original graph \(G\) so they may not be included in some spanning forest for the next iteration. However, we can view the markings on the vertices as reset, so no vertices are marked on the next iteration.

\[ G = G_0 \]
\[ G_1 = G_0 - F_1 \]
\[ H_1 = F_1 \]
\[ H_2 = F_1 \cup F_2 \]

Unmarked vertices
Marked and unscanned vertices
Scanned vertices

Figure 1: An example showing two iterations of scan-first search on the graph \(G\).

### 3.1 Performance in Different Computational Models

The most important running-time is that of the algorithm running in parallel, using the CRCW PRAM model in this case. Our first spanning tree \(T\) can be found in \(O(\log n)\) time using \(C(n, m)\) processors. Our preorder numbers and neighbours can also be calculated in \(O(\log n)\) time because parallel techniques[4] with \(O((n + m)/\log n)\) processors, our \(C(n, m)\) value. For this reason, we can generate a single \(T'\) corresponding to one iteration in \(O(\log n)\) time.

Using a distributed breadth-first search[1] approach, we can find our spanning forest in \(O(d \cdot \log^3 n)\) time on a graph with diameter \(d\) using \(O(m + n \log^3 n)\) messages. The sequential approach is quite simply the running time for breadth-first search, \(O(m + n)\).

### 3.2 The Main Certificate Theorem

Given an undirected graph \(G = (V, E)\) with \(n\) vertices, let \(k\) be some positive integer. For all \(i = 1, 2, \ldots, k\), let \(E_i\) be the set of edges generated by the \(i\)th iteration of scan-first search, corresponding to a graph \(G_{i-1} = (V, E - (E_1 \cup \ldots \cup E_{i-1}))\). So for each iteration of scan-first search, as stated
above, we will remove edges from the graph \( G \) to create some new graph \( G_i \) that results at the end of the \( i \)th iteration. For every iteration \( i \), our scan-first search forest is built from the graph \( G_{i-1} \), where \( G = G_0 \). The claim of the Main Certificate Theorem is that the union \( E_1 \cup \ldots \cup E_k \) is a certificate for the \( k \)-vertex connectivity of \( G \) and that it has at most \( k(n-1) \) edges.

### 3.3 Proving the Main Certificate Theorem

Let \( E_i \) denote the edge set of a spanning forest \( F_i \), generated by a scan-first search on \( G_{i-1} \), such that \( E_2 \) and \( F_2 \) would be generated from the graph \( G_1 \). Remember that \( G_1 \) is the graph after the first iteration of scan-first search, an iteration that was run on the graph \( G = G_0 \). We will let \( H_i \) denote the subgraph that results from the union of all the edge sets up to this iteration, so \( H_i = (V, E_1 \cup \ldots \cup E_i) \). Refer to Figure 1 for a look at what \( G_{i-1}, F_i, \) and \( H_i \) are at each \( i \)th iteration.

Our theorem so far states that after \( k \) iterations, we should have a certificate for the \( k \)-connectivity of \( G \). Since \( H_k \) is the accumulation of all the edges in our \( k \) iterations of the scan-first search, then \( G \) must be \( k \)-connected if \( H_k \) is. We can prove this through contradiction, by assuming that \( H_k \) is not \( k \)-vertex connected, but \( G \) is.

**Lemma 3.1** If \( H_k \) is disconnected but \( G \) is connected as stated above, then:

1. There is a subset \( S \subset V \) with \( |S| < k \) such that \( H_k - S \) is disconnected.
2. \( F_k \) contains a simple tree path \( P_k \) whose two end points are in different connected components of \( H_k - S \).

**Statement 1**

Menger’s theorem states that the some \( S \) of at most \( k-1 \) vertices does exist that will make \( H_k - S \) disconnected. Moving forward, we know that \( H_k \) is made disconnected by removing the set of edges in \( S \) from it, and since \( |S| \leq (k-1) \), we know that \( H_k \) cannot be \( k \)-vertex connected by the very definition of \( k \)-vertex connectivity. However, since \( G \) is \( k \)-vertex connected, we know that we can remove any set of \( k-1 \) edges from it and still maintain connectivity, and since \( |S| \leq (k-1) \) it follows that \( G - S \) is still \( k \)-vertex connected.

**Statement 2**

If \( H_k \) is disconnected and \( G \) is connected, since \( H_k \) is a subgraph of \( G \), there there should exist some edge in \( G \) that will bridge two separate connected components in \( H_k \). Since \( H_k \) is assumed to be disconnected, and \( H_k = E_1 \cup \ldots \cup E_k \), then some edge \( e \) that connects disconnected components in \( H_k \) cannot exist in \( E_1 \cup \ldots \cup E_{k-1} \). Since this edge must then be in \( G_{k-1} = (V, E - (E_1 \cup \ldots \cup E_{k-1})) \), and an edge must be part of one connected component that makes a spanning tree \( T \) in \( F_k \), there must exist some tree path \( P_k \subseteq T \) between the two end points of \( e \).

So far, we are working on the assumption that \( H_k \) is disconnected but \( G \) is connected, which we are trying to prove is not possible by contradiction. At this point, based on our assumption, we have discovered a path \( P_k \). However, to finish this contradiction, we will show that \( P_k \) cannot exist.
It is important at this point to remember our definition of $S$. $H_k = E_1 \cup \ldots \cup E_k$ after the $k$th scan-first search iteration. The set $S$ is a set such that $H_k - S$ is disconnected but the graph $G - S$ is not, and we are trying to show that this is not possible if $H_k$ is a certificate for $k$-connectivity for $G$.

Let $\omega = |S|$, and $s_1, \ldots, s_\omega \in S$ where $s_i$ is the first vertex in $S - \{s_1, \ldots, s_{i-1}\}$ scanned by scan-first search at the $i$th iteration. We know that $\omega < k$ because $\omega = |S| < k$ from Lemma 3.1.

Now let us discuss the notion of the home component of $s_i$. Any $s_i$ corresponds to some scan-first search forest $F_i$, with $r$ as the root of the tree in that forest that contains $s_i$. There are three different cases that define the home component as follows:

1. $r \notin S$, so the home component of $s_i$ is the connected component in $H_k - S$ that contains $r$.

   The first case seems clear enough. We scanned some $s_i \in S$ during our scan-first search, but the root of the tree in $F_i$ that contains $s_i$ is not also in $S$. That means there is some connected component in the graph $H_k - S$ with $r \notin S$ as its root, and $s_i$'s home component is this connected component.

2. $r \in S$ and $r \neq s_i$, so $s_i$ is in the home component of $r$.

   Here we need to focus on the fact that the $s_i$ at this point cannot be any of the set of \{s_1, \ldots, s_{i-1}\}, so if we scan a vertex $v \in S$ as the first in a new tree of forest $F_i$, then the only way we can avoid case 3 is if $v$ had already been scanned as the first $v \in S$ in a previous iteration, and is in our set \{s_1, \ldots, s_{i-1}\}. In this case, the home component becomes that of the root, and the root was some $s_j$ for $j < i$.

3. $r = s_i$, so $s_i$ has no home component.

   Our third case means that the first vertex $s_i \in S$ we scanned in our scan-first search at the $i$th iteration was the root of a new tree in the forest $F_i$, so $r = s_i$, and we can fairly clearly see how $s_i$ can have no home component in $H_k - S$ in this case.

**Lemma 3.2** For each $s_i \in S$, if $s_i$ satisfies either case 1 or 2 from the above definition, then the home component of $s_i$ is a connected component of $H_k - S$. If $s_i$ satisfies case 3, then $s_i$ has no home component.

The lemma provides a clear restatement of cases 1 and 3 from above. However, things are more difficult in case 2. We must prove that $s_i$'s home component is actually part of $H_k - S$, and we will do this inductively.

For all $j < i$, if $s_j$ satisfies case 2 then we assume that the home component of $s_j$ is a connected component in $H_k - S$. We are trying to prove that $s_i$ has a home component, and because $s_i$ satisfies case 2, we know that the root of the tree in $F_i$ containing $s_i$ is $s_h \in S$. So in this case, the root $r$ of the tree in our forest $F_i$ is in $S$ as stated in case 2 above is our $s_h \in S$. Now we will try to show that $s_h$ was scanned before $s_i$ with the claim that $h < i$.

It is fairly clear that we would have scanned the root $r \in S$ of the tree containing $s_i$ before $s_i$ itself, and it would seem that in this case $r$ would just be for some $h < i$. Let us show this by contradiction. Assume that $h > i$. This would mean that $s_h \in S - \{s_1, \ldots, s_{i-1}\}$. Since we know that $s_i$ is a descendant of $s_h$ in our tree, then our $s_h$ would be scanned before $s_i$, and therefore
Figure 2: Showing the 3 cases with $H_k$ divided by the set $S$. Case i) There is a clear path from the home component $A$ to $s_i \in S$. Case ii) Our $r$ was the $s_i$ discovered in Case i, so $r$ cannot be the current $s_i$. Case iii) Our $r$ is in $S$, but no previous $s_i$ had been discovered here, so there is no original tree containing this vertex rooted in a component, and thus no home component.

cannot come after $s_i$ in the ordering as it would in this case. Therefore we have a contradiction, and $h < i$.

We will try to show that $F_h$ cannot have $s_h$ as the root of any of its trees through contradiction. Let us assume that $s_h$ is the root of a tree in $F_h$. Adjacent edges to $s_h$ in $F_1, \ldots, F_{h-1}$ have already been removed from $G_{h-1}$, and all remaining edges adjacent to $s_h$ will be missing from $G_h$. If $s_h$ is the root of a tree, then $G_h$ will contain $s_h$ as an isolated vertex. Since $i > h$ as stated above, and $s_h$ is isolated in $F_i$, we can show that it our above claim that $s_i$ is a descendant of $s_h$ is contradictory because $s_h$ is isolated in this case, thus $s_h$ can not be the root of a tree in $F_h$.

Moving forward, we must define some new terms. For every vertex $s \in S$, let $hcc(s)$ denote the home component of $s$ given that $s$ has one. Otherwise, $hcc(s)$ denotes $s$. In addition, for all $v \in (V - S)$, we will let $hcc(v)$ denote which connected component in $H_k - S$ contains $v$. A jump of $F_i$ is a simple tree path $Q = v_1, \ldots, v_q$ in $F_i$ where $hcc(v_1) \neq hcc(v_q)$.

We divide edges incident with all $s \in S$ in $G$ into the following three categories:

- Back edges: Edges between $s$ and its home component.
- Side edges: Edges between $s$ and other vertices in $S$.
- Forward edges: Every other edge incident to $s$.

It is important to realize that $s$ may not have a home component, and so by the above definitions will not have any back edges.
Lemma 3.3 The following statements are true:

1. Every jump of $F_1$ contains at least one vertex of $S$.

2. The scan-first search forest $F_1$ contains all the forward edges of $s_1$.

Statement 1.

We will try to prove the first statement in this lemma by contradiction. We assume that $F_1$ does have a jump $Q = v_1, \ldots, v_q$ containing no vertices from $S$. We know then that $v_1, v_q \in (V - S)$. We also know that $hcc(v_1), hcc(v_q) \in (H_k - S)$, and $hcc(v_1) \neq hcc(v_q)$. $Q$ is a path that connects two pieces of $H_k - S$, and therefore $Q$ must contain some vertex in $S$. Remember that $H_k$ is a connected graph if $G$ is, and we are given a connected graph $G$, so if $H_k$ is disconnected by some $S$ and $Q$ reconnects disconnected components in $H_k - S$, it must pass through some vertex $s \in S$. Therefore, we have a contradiction, $Q$ does contain at least one vertex from $S$.

Statement 2.

Now we look at the second statement. We know that our first scan-first search forest $F_1$ will be a tree since the graph $G$ is connected, and just like breadth-first search, scan-first search will find a single connected component. If $r$ is the root of $F_1$, two things are possible: either $r \in S$ or $r \notin S$.

- $r \in S$: Since this is the first vertex we have encountered from $S$, we know that $s_1 = r$. Since our scan-first search just started, none of the vertices adjacent to $r$ have been marked, and thus all edges adjacent to $r$ are added to $F_1$. At this point, we have added all edges adjacent to $r$ at this stage, it logically follows that we must have included all forward edges as a subset of these.

- $r \notin S$: Once again, we will use contradiction to prove this case. We know that $hcc(s_1) = hcc(r)$. We assume that there exists some forward edge $(s_1, x) \notin F_1$. The only way the edge would not be added if $x$ is adjacent to $s_1$ is if $x$ had been marked prior to reaching $s_1$. By definition of a forward edge, $hcc(s_1) \neq hcc(x)$. It follows from this that the path $Q$ from $r \to x$ must be a jump of $F_1$, which from the statement above, says that $Q$ must include at least one vertex from $S$. We know that $s$ is an ancestor of $x$, and $s$ must be scanned before $x$ is marked. For $x$ to have been marked at this point, however, $s$ would have to be scanned before $s_1$ to mark $x$ such that the edge $(s_1, x)$ does not exist in $F_1$. That would mean that $s_1$ would have to have a higher index, and this is a contradiction, therefore all forward edges of $s_1$ are in $F_1$.

Lemma 3.4 For each $i \in \{1, 2, \cdots, \omega\}$, the following statements are true:

1. Every jump of $F_i$ contains at least one vertex in $S - \{s_1, \cdots, s_{i-1}\}$.

2. $F_1, \cdots, F_i$ contain the following edges of $G$:
   a) all forward edges of $s_i$.
   b) all side edges $\{s_i, s_j\}$ with $i > j$ and $hcc(s_i) \neq hcc(s_j)$.
We will attempt to prove this lemma by induction. We assume it holds for \( i = t \), and try to show that it holds for \( i = t + 1 \).

First, let us get some definitions out of the way:

- \( Q = v_1, \ldots, v_q \) : a jump of \( F_{t+1} \) with no vertices from \( S - \{ s_1, \ldots, s_t \} \).
- \( W \) : the set of all vertices in both \( Q \) and \( S \).
- \( U_0 \) : the set of edges in \( Q \) that have two end points in \( H_k - S \).
- \( U_1 \) : the set of edges in \( Q \) that have one end point in \( W \) and the other in \( H_k - S \).
- \( U_2 \) : the set of edges in \( Q \) that have two end points in \( W \).

**Statement 1.**

First we will show that the first statement holds for \( i = t + 1 \). To prove by contradiction, we make the assumption that a jump of \( F_{t+1} \) does not contain a vertex in \( S \). Remember that a jump of \( F_{t+1} \) is some path \( Q = v_1 \rightarrow v_q \) where \( hcc(v_1) \neq hcc(v_q) \).

Any edge \( (x, y) \in U_0 \) implies \( hcc(x) = hcc(y) \), which seems fairly clear since both end points are in the same connected component, and will then share the same home component. In the case of \( U_1 \), we have assumed that \( F_{t+1} \) does not contain a jump, and so our path \( Q \) can contain no forward edges, and so all edges \( (x, y) \in U_1 \) must satisfy \( hcc(x) = hcc(y) \). Since side edges are edges that have both vertices in \( S \), then edges in \( U_2 \) also satisfy \( hcc(x) = hcc(y) \) because these edges are along the path \( Q \) containing vertices that are descendants of the root \( r \) that have not crossed into a separate connected component yet.

We have shown that for every edge in the path \( Q \), if \( Q \) is a jump of \( F_{t+1} \) containing no vertices in \( S \), then for any edge \( (x, y) \in Q, hcc(x) = hcc(y) \) which violates the property of a jump where \( hcc(v_1) \neq hcc(v_q) \) and so we have a contradiction; any jump of \( F_{t+1} \) must contain a vertex in \( S \).

**Statement 2.a**

Now we will show that statement 2.a holds for \( i = t + 1 \). Let \( T \) denote a tree of \( F_{t+1} \) containing \( s_{t+1} \), with \( r \) as its root. For this, we need to remember that forward edges are all edges that do not connect two vertices in \( S \) and don’t connect some vertex \( s_i \) with its home component. We will again see that there are two cases:

1. \( r \neq s_{t+1} \) : We will once again try to prove this by contradiction. Since we claimed that all forward edges of \( s_{t+1} \) will be part of the forest \( F_{t+1} \), let us assume there is some edge \( e = (s_{t+1}, x) \notin F_{t+1} \).
   In a similar fashion to what we have seen before, for this edge not to be included in \( F_{t+1} \), \( x \) would have to be scanned before \( s_{t+1} \) at the \( (t+1) \)th iteration. Since we have assumed that an \( e \) exists, and it is within \( T \), then there exists a path \( Q \) from \( r \rightarrow x \). By our assumption, \( e = (s_{t+1}, x) \) is a forward edge, and so \( hcc(s_{t+1}) \neq hcc(x) \) and equivalently \( hcc(r) \neq hcc(x) \).

By definition, a jump of \( F_{t+1} \) is simply our path \( Q \) at this point, because the root of our tree \( T \) has a different home component than the end of the path, \( x \). We just claimed that path \( Q \) must contain some vertex in \( S - \{ s_1, \ldots, s_t \} \), but for this to occur, and \( x \) to have been scanned before \( s_{t+1} \), then on the \( (t + 1) \)th iteration some other edge \( s \) must have been discovered in \( S \) first.
This would mean that the index of \( s_{t+1} \) would be incorrect and we have a contradiction, thus statement 2.a holds for all \( i = t + 1 \).

2. \( r = s_{t+1} \) : In this case, we know that \( r \) is the first vertex scanned by the \((t + 1)\)th iteration of the scan-first search, and therefore all edges incident to \( s_{t+1} = r \) are unmarked. Since all edges then get included in \( F_{t+1} \), it must include all forward edges of \( s_{t+1} \).

**Statement 2.b**

Lastly, we must show that statement 2.b holds true for all \( i = t + 1 \). Recall that side edges are all edges with both end points in \( S \). We deal with the same two cases as above:

1. \( r \neq s_{t+1} \) : The proof for this is very similar to above. Let us assume that some edge \((s_{t+1}, s_j) \notin F_{t+1} \), and assume that \((t + 1) > j \). We can see immediately that for this to be true, \( s_j \) would have to have been scanned first, making the index \( s_{t+1} \) incorrect at this iteration, since \( s_j \) would be in its place otherwise because it would be the vertex in \( S \) scanned at the \((t + 1)\)th iteration.

   We know that \( hcc(s_{t+1}) = s(r) \) by definition in this case. We assume an edge \((s_{t+1}, s_j) \) where \( hcc(s_{t+1}) \neq hcc(s_j) \). For this edge to not be part of \( F_{t+1} \) and \( hcc(s_{t+1}) \neq hcc(s_j) \), \( s_j \) would have to have been scanned before \( s_{t+1} \) in the same iteration. If it were scanned after as part of the same \( T \), \( hcc(s_{t+1}) \) would have to be equal to \( hcc(s_j) \). This is because \( s_j \) would be a descendant of \( s_{t+1} \) and by that logic, it must have the same home component. Since we have assumed that edge \( e \) exists, \( s_j \in T \), it follows that there is some path \( Q = r \rightarrow s_j \). However, since \( hcc(s_{t+1}) \neq hcc(s_j) \), \( Q \) is a jump of \( F_{t+1} \) by our assumption.

   To recap, \( s_j \in W \), \( s_j \) is a descendant of all of \( W - \{s_j\} \), and \( s_{t+1} \in W \). \( W - \{s_j\} \) must logically be scanned before scanning \( s_j \), and since it is a descendant of \( s_{t+1} \) and \( s_{t+1} \) would be encountered first, then we must scan \( W - \{s_j\} \) before \( s_{t+1} \) as well. We know that before encountering \( s_{t+1} \), \( S = \{s_1, \ldots, s_t\} \). Notice how \( S = \{s_1, \ldots, s_t\} \) is \( W - \{s_j\} \) in this case. Since we assumed that \( s_j \) was scanned first, that would assume \( s_j \) is in \( S \) already, but this is a contradiction, because if it were added to \( S \) in this iteration then it would be \( s_{t+1} \) before we encounter the actual \( s_{t+1} \). Our contradiction shows that if such a side edge \((s_{t+1}, s_j) \) does exist in \( F_{t+1} \), then \( hcc(s_{t+1}) = hcc(s_j) \).

2. \( r = s_{t+1} \) : If \( r \) is \( s_{t+1} \) then none of the edges incident to \( s_{t+1} \) are unmarked and will now all be included in \( F_{t+1} \), so this naturally includes all side edges, and our above contradiction is unnecessary to show they will have the same home component.
At this point, we can wrap up the proof of the main certificate with one final lemma. We originally assumed that there is some set $S$ such that $H_k - S$ is disconnected but $G - S$ is connected. For this reason, there is some edge in $G$ but not in $S$ whose end points are in two connected components of our disconnected graph $H_k - S$. Recall that we were trying to prove from Lemma 3.1 that $F_k$ contains a simple tree path $P_k \subseteq T$ for some spanning tree $T \in F_k$.

**Lemma 3.5** The path $P_k$ of Lemma 3.1 cannot exist.

We will once again use contradiction to prove this true. We start by assuming that $P_k$ does exist, and we let $P_k = v_1, \ldots, v_q$. We stated earlier that $hcc(v)$ for all $v \notin S$ denotes the connected component $v$ is in, so if $P_k$ is a simple path that contains an edge between two components of $H_k - S$, then $hcc(v_1) \neq hcc(v_q)$. This makes $P_k$ a jump of $F_k$.

Let us define some sets before finishing our proof:

- $W$ : Vertices in both $P_k$ and $S$.
- $U_0$ : Edges in $P_k$ with both end points in $H_k - S$.
- $U_1$ : Edges in $P_k$ with an end point in $W$ and another in $H_k - S$.
- $U_2$ : Edges in $P_k$ with both end points in $W$.

Using Lemma 3.4 we are trying show that $P_k$ can not exist. Assuming that $P_k$ exists, it would be introduced to our subgraph $H_k$. Remember from Lemma 3.1 we have assumed that some $S$
disconnects $H_k$, but not $G$. For $S$ to exist while still including our edge in $F_k$ that connects two disconnected components in $H_k - S$, our path $P_k$ must avoid passing through $S$.

Looking at Lemma 3.4, we know that edges in $U_0$ have the same home component, so for all edges $(x, y) \in U_0, hcc(x) = hcc(y)$. Edges from $U_1$ can not be forward edges from Statement 2.a of Lemma 3.4, and so again for all edges $(x, y) \in U_1, hcc(x) = hcc(y)$. For $U_2$ we see Statement 2.b of Lemma 3.4 and realize we determined that for all edges $(x, y) \in U_2, hcc(x) = hcc(y)$.

We stated above that $P_k$ is a jump of $F_k$ since it connects two separate connected components in $H_k - S$. However, we just determined that for all edges $(x, y) \in P_k, hcc(x) = hcc(y)$ since we had to avoid an edge passing through $S$. For $P_k$ to be a jump, we must satisfy the condition $hcc(v_1) \neq hcc(v_q)$, and so we have a contradiction, $P_k$ can not exist. Therefore it is not possible to have some set $S$ such that $H_k - S$ is disconnected by $G - S$ is not, and so if $G$ is $k$-vertex connected then $H_k$ must be as well, and our proof that $H_k$ is a valid $k$-vertex connectivity certificate for $G$ is complete.
References


