The general outline of the algorithm is as follows: we use an augmented depth-first search to enumerate all the possible paths of length less than $k$ and find their connection strength with the source. Then, we include a list of connection strengths of each possible path to a particular node. After the augmented depth-first search is finished, we go back through all the nodes and find the maximum connection strength.

To begin, we will describe the augmented depth-first search. This search will be computing connection strengths of all the possible paths from the source node. First, we start at the source node $s$. We will be keeping track of the length of the path $l$, so we initialize $l = 0$. Then we loop through all nodes $v$ in $Adj[s]$, setting $l += 1$, and calculating the edge rank $ER(s,v)$ and setting the current strength to $s = ER(s,v)$. At each node $v$, we have a list $v.Str$ of the strengths of all the possible paths to that node, so we perform $v.Str.append(s)$. Now, we will visit all the nodes $v’$ in $Adj[v]$ which are not in the current path we have taken to $v$. This disallows cycles. We do this because cycles will never contribute to the maximum connection strength since edge ranks are in the range $(0,1)$, so that cycles will always have strictly lower connection strengths than the original pass over a set of nodes.

Then, we perform the same operations. We set $l += 1$, and calculate $ER(v, v’)$ so we can update the current strength $s = s \* ER(v,v’)$. Then, we update the list $v’.Str.append(s)$. We perform this procedure recursively, stopping when $l == k$.

This is much like depth-first search because we move through a path of length $k$, then after we finish performing the procedure at node $v\_k$ when $l == k$, we backtrack to the node $v\_{k – 1}$ which just preceded it, and continue.

This will update the $Str$ list at each node. After the augmented depth-first search completes, each node will contain a $Str$ list of the strength values that correspond to every possible path to a node, where the possible paths are of length $k$ or less. Now, we can go through each node, and find the maximum of each $Str$ list, and say that this is the maximum connection strength possible to that node. We know this is true because if there exists a path from $s$ to $v$, that path’s connection strength must be in $v.Str$. Thus, we are confident we are finding the maximum connection strength possible for a vagueness of $k$.

The runtime analysis will show that the algorithm runs in $\Theta(V + kE)$. First, we have to move through all the possible paths of length less than $k$. Since calculating $s \* ER(v,v’)$, $v.Str.append()$, and $l +=1$ all take $\Theta(1)$ time, we only need to know $P\_k$, the number of possible paths with lengths less than $k$. This will tell us the running time of the depth-first search, because $\Theta(P\_k) \* \Theta(1) = \Theta(P\_k)$, since we are only performing a constant number of operations for each path.

To find $P\_k$, we note that we do not have to consider paths with cycles. In the worst case, we will have a tree (since it has no cycles). Thus, each edge can be visited at most $k$ times. This is because an edge cannot occur more than once in a given path. Thus, there are a total of $k$ we have at most Thus, even with the trivial example of moving back and forth on an edge, we will obtain $k$ visits for that edge. Therefore, we have a total of $\Theta(kE)$ possible paths with length $k$ or less.