Tarjan's improved algorithm for optimal branchings:

Modify the shrinking step as follows:
In the zero-graph G_0 = (V, Z), run BFS, with r as the source, using only the zero-edges. If all nodes can be reached, then the BFS tree is a 0-weight spanning tree, rooted at r, and so it is an MST.

Shrinking step: Otherwise, let G_0 have k strongly connected components. Let r be in scc number 1. Since r has no incoming edges in G, that scc will not have other nodes in it. Shrink each scc into a single node. The new graph has k nodes, $C_1 \cdots C_k$. The weight of edge (C_i, C_j) is equal to the minimum weight of an edge connecting some $u \in C_i$ to some $v \in C_j$:

$$(C_i,\ C_j).weight\ =\ \min_{u \in C_i,\, v \in C_j} (u,\ v).weight$$

In the new graph H, C_1 (the node containing r) is the root node. For each edge of H, record its image, which is the edge of G to which it corresponds (i.e., a minimum-weight edge that is argmin in the above equation).

Theorem: Weight of MST of G rooted at r = Weight of MST of H rooted at C_1.

Expansion step: After finding an MST of H, rooted at C_1, we can expand each scc and find an MST of G, rooted at r as follows. C_1 contains only the root vertex, r. MST(H) rooted at C_1 has exactly one edge into each C_i, $i = 2 \cdots k$. Let the edge into C_i correspond to edge (u,v) of G, where $v \in C_i$. Find a spanning tree within C_i, rooted at v, using only 0-weight edges. The MST of G is the union of the k-1 spanning trees within C_i, $i = 2 \cdots k$, and the edges of G that are images of the edges of MST(H).

Implementation notes

To be able to solve large instances, it is not feasible to create a new graph in each phase of the algorithm. In many iterations, most strongly connected components may have just one node each, and it is possible that only one scc actually shrinks. Therefore, it is necessary to be able to add new vertices and edges as the algorithm progresses. Existing edges and vertices that are entirely contained within the same component have to be disabled. We can extend the graph, vertex and edge classes to facilitate these changes. Design the classes and their iterators carefully. If designed properly, it should be possible to call standard implementations of SCC, BFS, or DFS on this extended graph, and when it iterates over the outgoing edges of a node, the algorithm uses only edges of zero weight. When iterating over the vertices of a graph, it should automatically skip the disabled vertices.