stackoverflow.com

algorithm - Single edge addition to minimize number of bridges in a graph

5-6 minutes

This is not a complete answer but some ideas to steer you towards it:

To avoid having to run the bridge-finding algorithm after trying each possible edge, it pays to ask: By how much can adding a single edge (u, v) change the number of bridges in a graph G?

1. If u and v are not already connected by any path in G, then certainly (u, v) will itself become a bridge. What about the "bridgeness" (bridgity? *bridgulence?*) of all other pairs of vertices? Does it change? (Most importantly: Can any edge go from being a bridge to being a non-bridge? If you can prove that this can never happen, then you can immediately discard all such vertex pairs (u, v) from consideration as they can only ever make the situation worse.)
2. If u and v are already connected in G, there are 2 possibilities:
1. Every path P that connects them shares some edge (x, y) (note that x and y are not necessarily distinct from u and v). Then (x, y) is a bridge in G, and adding (u, v) will cause (x, y) to stop being a bridge, because it will then become possible to get from x to y

"the long way", by going from x back to u, via the new edge (u, v) to v, and then back up to y. (This assumes that x is closer to u on P than y is, but clearly the argument still works if y is closer: just swap u and v.) There could be multiple such bridges (x, y): in that case, *all* of them will become non-bridges after (u, v) is added.

2. There are at least 2 edge-disjoint paths P and Q already connecting u and v. Obviously no edge (x, y) on P or Q can be a bridge, since if (x, y) on P were deleted, it's still possible to get from x to y "the long way" via Q. The question is, again: What about the bridgeness of all other vertex pairs? You should be able to prove that this property doesn't change, meaning that adding the edge (u, v) leaves the total number of bridges unchanged, and can therefore be disregarded as a useless move (unless there are no bridges at all to start with).

   We see that 2.1 above is the only case in which adding an edge (u, v) can be useful. Furthermore, it seems that the more bridges we can find in a single path in G, the more of them we can neutralise by choosing to connect the endpoints of that path.

   So it seems like "Find the path in G that contains the most bridges" might be the right criterion. But first we need to ask ourselves: Does the number of bridges in a path P accurately count the number of bridges eliminated by adding an edge from the start of P to the end? (We know that adding such an edge must eliminate *at least* those bridges, but perhaps some others are also eliminated as a "side effect" -- and if so, then we need to count them somehow to make sure that we add the edge that eliminates the most bridges overall.)

   Happily the answer is that no other bridges are eliminated. This

time I'll do the proof myself.

Suppose that there is a path P from u to v, and suppose to the contrary that adding the edge (u, v) would eliminate a bridge (x, y) that is not on P. Then it must be that the single edge (x, y) is the only path from x to y, and that adding (u, v) would create a second path Q from x, via the edge (u, v) in either direction, to y that avoids the edge (x, y). But for any such Q, we could replace the edge (u, v) in Q with the path P, which from our initial assumption avoids (x, y), and still get a path Q' from x to y that avoids the edge (x, y) -- this means that (x, y) must have already been connected by two edge-disjoint paths (namely the single edge (x, y) and Q'), so it could not have been a bridge in the first place. Since this is a contradiction, it follows that no such "removed as a side effect" bridge (x, y) can exist.

So "Find the path in G that contains the most bridges, and add an edge between its endpoints" definitely gives the right answer -- but there is still a problem: this sounds a lot like the Longest Path problem, which is NP-hard for general graphs, and therefore slow to solve.

However, there is a way out. (There must be: you already have an O(V^2*E) algorithm, so it can't be that your problem is NP-hard :-)) Think of the biconnected components in your input graph G as being vertices in another graph G'. What do the edges between these vertices (in G') correspond to in G? Do they have any particular structure? Final (big) hint: What is a critical path?