UDC 519.682.3

S. Antonakopoulos, I. Diakonikolas

# HEURISTIC IMPROVEMENTS TO A LINEAR-TIME APPROXIMATION ALGORITHM FOR THE 2-ECSS PROBLEM

Diakonikolas national technical university of Athens, Greece

## Introduction

Communication networks are often modeled by graphs, in which the vertices represent the nodes and the edges represent the feasible communication links. This model allows one to measure the properties and performance of the network by different graph theoretic parameters.

When designing reliable communication networks, the least that we must guarantee is that, after failure of some nodes or links, the surviving network still allows communication between all non-faulty nodes. This implies constraints on the connectivity of the corresponding graph. The $k$-vertex connectivity (respectively $k$-edge connectivity) is associated to the capability of a network to resist to the failure of any subset of $k-1$ nodes (respectively links). A general network design problem which requires the underlying network to be resilient to link failures is known as the Edge Connectivity Survivable Network Design Problem. In fact, 2-edge connectivity is a major feature in today's fast and reliable communication networks, as a single transmission failure could cause intolerable losses.

Most network optimization problems that require finding minimal subgraphs satisfying given connectivity constraints are NP-hard. As a result, it has become imperative to design approximation algorithms for such problems (e.g. [5], [2]). Unfortunately, the 2-edge-connected spanning subgraph problem is proven to be MAX SNP-hard [3], meaning that it cannot be approximated to less than a particular multiplicative constant, unless . On the other hand, several algorithms have been proposed that guarantee satisfactory approximation ratios, starting from 3/2 [6] and going down to 17/12 [1], 4/3 [8], $4/3-\epsilon$ [7] and finally 5/4 [4]. However, it is quite unlikely that a lower ratio will be achieved soon, because this would require finding new lower bounds on the optimal solution.

It is worth noting that only the first algorithm, due to Khuller and Vishkin, requires linear time with respect to the number of edges of the input graph. All the rest are significantly more time-consuming, which renders them impractical for use in virtual topologies over large networks, where speed is of the essence. Therefore, in this paper we focus on devising heuristic techniques to improve the aforementioned linear-time algorithm, without increasing its time complexity. We also implement several well-known lower bounds on the optimal solution. Thus, we are able compare the performance of the algorithm with and without heuristics, using random graphs as inputs, and draw useful conclusions.

## Heuristics

Our heuristic improvements to Khuller and Vishkin's algorithm [6] involve:

- The choice of the next vertex to be visited by the Depth-First Search routine, according to a primary and a secondary criterion, as implemented in the selectNextVertex() function.
- The removal of DFS tree edges which are no longer necessary because of the back edges included in $E_H$ by the algorithm. This optimization takes place after the Depth-First Search and is carried out by the PostProcessing() procedure.

The improved algorithm is shown below in pseudocode. In large part, we follow the notation used in [6]. In particular:

- Assume that all vertices are numbered by the order in which they were visited by the DFS, so that the operators < and > make sense when used on vertices.
- Every vertex is initially *unvisited*. When the DFS routine visits it for the first time, it becomes *discovered*. When we finally exit from the vertex, it is marked as *finished*.
- $H$ is the requested spanning subgraph and $E_H$ its edge set.

- low[v] is the vertex with the smallest DFS number that can be reached by following a single back edge originating anywhere in the DFS subtree rooted at v.
- $low_H[v]$ is defined to be the vertex with the smallest DFS number that can be reached by following a single back edge belonging to $E_H$ and originating anywhere in the DFS subtree rooted at v.
- savior[v] is the vertex in the DFS subtree rooted at v which is connected to low[v] by a back edge.

—— Algorithm for 2-ECSS ——
(* main routine *)
$E_H \leftarrow \square$;
DFSHeur($v_0$, nil);
PostProcessing($v_0$);
procedure DFSHeur(v, u); (* u is the parent of v in DFS tree *)
mark v discovered;
low[v] ← v;
$low_H[v] \leftarrow v$;
savior[v] ← v;
while v has unvisited neighbors do begin
w ← selectNextVertex(v);
$E_H \leftarrow E_H \cup \{(v, w)\}$;
DFSHeur(w, v);
low[v] ← min{low[v], low[w]};
if low[v] changes by the above then savior[v] ← savior[w];
$low_H[v] \leftarrow \min\{low_H[v], low_H[w]\}$;
end;
for each $w \in$ Adj[v] do
if w is discovered and $w \neq u$ then begin (* (v, w) is a back edge *)
low[v] ← min{low[v], w};
if low[v] changes by the above then savior[v] ← v;
end;
if $low_H[v] = v$ and $u \neq$ nil then begin
(* edge (u, v) is threatening to be a bridge; add the back edge (savior[v], low[v]) to cover it *)
$E_H \leftarrow E_H \cup \{(savior[v], low[v])\}$;
$low_H[v] \leftarrow low[v]$;
end;
mark v finished;
end DFSHeur;
function selectNextVertex(v)
for each unvisited $w \in$ Adj[v] do begin
$d_{G-H} \leftarrow$ number of unvisited neighbors of w;
x ← lowest discovered neighbor of w (nil if there is none)
(* primary criterion: choose vertex with the fewest unvisited neighbors *)
if $d_{G-H} <$ bestD then begin
bestD ← $d_{G-H}$; bestX ← x;
bestVertex ← w;
end
(* secondary criterion: choose vertex with the highest lowest discovered neighbor *)
else if $d_{G-H} =$ bestD and (x = nil or x > bestX) then begin
bestX ← x;
bestVertex ← w;
end;
end;
return bestVertex;
end selectNextVertex;
procedure PostProcessing(root) (* remove some redundant DFS tree edges *)
for each $v \neq$ root do
redundant ← false;
u ← parent of v in the DFS tree;

if there exists some w such that u = $low_H$[w] then begin
if (v, low[v]) □ $E_H$ then redundant ← true;
if there exist two children x, y of v in the DFS tree such that low[x] ≠ v and
low[y] ≠ v then redundant ← true;
end;
if redundant then $E_H$ ← $E_H$ \ {(u, v)}; (* edge (u, v) can safely be deleted *)
end;
end PostProcessing; ■

Let y be the vertex with the smallest DFS number that is a neighbor of v. If there exists a descendant w of v such that y = low[w], then savior[v] will not be equal to v. This fact is implied in the above code, yet it deserves to be stated explicitly because in some cases it renders the post-processing more effective.

## Lower Bounding

The performance of our algorithm is measured against a set of lower bounds on the optimal solution, since it would be overwhelmingly time-consuming to find the optimal solution itself. One obvious lower bound is the number of vertices n. Another is the size of a tree carving of the input graph G, as explained in [6]. This bound is directly related to the number of edges in the solution found by the algorithm, so practically no extra calculations are needed.

We also used a more sophisticated idea, due to Vempala and Vetta [8]. It consists of finding a minimum sized spanning subgraph in which every vertex has degree at least two. The aforementioned problem is called D2. It should be noted that the optimal solution to D2 may not be connected. Given that any two-connected graph must have minimum degree at least 2, it is straightforward that the optimal solution to D2 provides a lower bound on the size of the optimal solution to 2-ECSS.

D2 is solvable in polynomial time by a two-step procedure.

First, find a maximum sized subgraph of G in which every vertex has degree at most two. It is easy to see that the solution to this problem consists of disjoint cycles and paths, which means that the problem asks for a maximum cycle – path cover of the graph. This can be computed efficiently via a simple reduction to maximum bipartite matching, as described below.

Second, add one more edge to each of the end vertices of every path found.

Given a simple undirected graph G with vertex set { and edge set E, construct a bipartite graph H with vertex set {} and edge set } } n {. Find a maximum-cardinality matching M in H. This takes } O time by using a variant of the Hopcroft – Karp bipartite matching algorithm. It is crucial that we impose the following restriction on the computation of M: for every j, k, at most one of the edges and ( is allowed in M.

Let } □ ) ) . The edge set P defines a set of disjoint paths in G. This is true because M is a matching, which means that any vertex j has at most one “incoming” edge () (corresponding to the at most one edge () and at most “outgoing edge” () (corresponding to the at most one edge ). The paths defined by P are all simple and some of them may be closed, i.e. cycles. The time to construct P from M is O. Therefore, the time complexity of finding the cover is dominated by the computation of the restricted maximum matching. Under the above constraints, it can be proven that M is of maximum size if and only if P is a maximum cycle – path cover of G.

At this point, the necessity of the imposed restriction should be clear. Both edges ( and () in the bipartite graph correspond to the same edge (of the initial graph. Therefore, the co-existence of such edges in the matching M destroys the maximality of the corresponding cover P.

## Conclusions

In this paper, we presented a set of heuristic improvements to a well-known approximation algorithm for the 2-edge-connected spanning subgraph problem, while preserving its linear time complexity. Furthermore, we implemented several lower bounds on the optimal solution, in order to assess the algorithm’s performance. Early tests using random graph inputs of order n = 10 to 24 showed that the heuristics produced a solution which was, on average, within 1% of the optimal. On those same inputs, the corresponding error percentage of the original algorithm was ~10%. This improvement is impressive and, since it requires only little more time to be achieved, may be of great practical usefulness. A complete set of computational results will be included it the final version of the paper.

## References

1. J. Cheriyan, A. Sebo, Z. Szigeti, “Improving on the 1.5-approximation of a smallest 2-edge connected spanning subgraph”, SIAM Journal of Discrete Mathematics, 14:170–180, 2001

2. J. Cheriyan, S. Vempala, A. Vetta, “Approximation algorithms for minimum-cost k-vertex connected subgraphs”, Proceedings of the 34th ACM Symposium on the Theory of Computing (STOC), pp. 306–312, 2002

3. C. G. Fernandes, “A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem”, Journal of Algorithms, 28:105–124, 1998

4. R. Jothi, B. Raghavachari, S. Varadarajan, “A 5/4-approximation algorithm for minimum 2-edge-connectivity”, Proceedings of the 14th annual ACM-SIAM Symposium on Discrete Algorithms, pp. 725–734, 2003

5. S. Khuller, B. Raghavachari, “Improved approximation algorithms for uniform connectivity problems”, Journal of Algorithms, 21:433–450, 1996

6. S. Khuller, U. Vishkin, “Biconnectivity approximations and graph carvings”, Journal of the ACM, 41:214–235, 1994

7. P. Krysta, V. S. Anil Kumar, “Approximation algorithms for minimum size 2-connectivity problems”, Proceedings of the 18th International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 431–442, 2001

8. S. Vempala, A. Vetta, “Factor 4/3 approximations for minimum 2-connected subgraphs”, APPROX 2000, pp. 262–273, 2000

**Spyridon Antonakopoulos, Ilias Diakonikolas** National Technical University of Athens, School of Electrical & Computer Engineering, Department of Computer Science 9, Iroon Polytechniou Str., Zografou 15780, Greece, Tel.: +30 210 7721644, E-Mail: {el99612, el99091}@mail.ntua.gr