Online Supplement for "The optimal design of low-latency virtual backbones"

Hamidreza Validi and Austin Buchanan

School of Industrial Engineering & Management
Oklahoma State University
{hamidreza.validi,buchanan}@okstate.edu

June 3, 2019

1 SETH-Based Hardness Results

Below, we consider fundamental algorithmic questions about latency-s CDS's. For example, how quickly can one verify that a vertex subset is indeed a latency-s CDS? How quickly can one perform local search moves for the minimum latency-s CDS problem? These problems admit relatively straightforward algorithms that run in time $O(mn)$ when distances are hop-based. A natural question is whether there exist faster algorithms. We show that, based on the strong exponential time hypothesis (SETH), this would be difficult.

SETH is an unproven complexity assumption that is stronger than P≠NP. While it is unproven and some doubt that it is true, it is nevertheless a benchmark for gauging how surprising or noteworthy a faster algorithm would be. The reader is referred to the survey of Lokshtanov et al. (2013) for more information about SETH. To prove our results, we use the following theorem of Roditty and Vassilevska Williams (2013).

Theorem 1. *If SETH holds, then for every $\varepsilon > 0$ there is no algorithm for verifying that a simple, connected graph $G = (V, E)$ has* $\operatorname{diam}(G) = 2$ *that runs in time $O(m^{2-\varepsilon})$.*

Below, we restate and prove Proposition 2 from the main body of the paper.

Proposition 1 (restatement of Proposition 2)**.** *If SETH holds, then for every $\varepsilon > 0$ there exists no algorithm for verifying that a subset D of vertices is a latency-s CDS that runs in time $O(m^{2-\varepsilon})$, even in the simplest nontrivial case of hop-based distances and $s = 2$.*

Proof. The proof follows by reduction from the problem in Theorem 1. Namely, bidirect the edges of the graph to get $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ and let $s = 2$, $D = V$, and $w_e = 1$ for each $e \in \overleftrightarrow{E}$. It can be observed that D is a latency-s CDS for $\overleftrightarrow{G}$ if and only if G has $\operatorname{diam}(G) = 2$, and this reduction runs in linear time, so the proposition follows. □

A similar negative result shows a difficulty that would be encountered when applying local search to the minimum latency-s CDS problem. In the following proposition, we refer to the local search move in which one is given a known-to-be-feasible solution $D \subseteq V$ along with a vertex $v \in D$, and the task is to determine whether one can move to $D \setminus \{v\}$ and maintain feasibility.

Proposition 2. *If SETH holds, then for every $\varepsilon > 0$ there exists no algorithm for the local search move defined above that runs in time $O(m^{2-\varepsilon})$, even in the simplest nontrivial case of hop-based distances and $s = 2$.*

Proof. The proof follows by reduction from the problem in Theorem 1, where we assume, without loss, that G is not complete and thus $\text{diam}(G) \geq 2$. Add a new node v to the graph $G = (V, E)$ and connect it to all other nodes. Call this new graph $G' = (V', E')$, where $V' = V \cup \{v\}$ and $E' = E \cup \{\{u, v\} \mid u \in V\}$. Bidirect all edges of G' to get $\overleftrightarrow{G}' = (V', \overleftrightarrow{E}')$. Let $s = 2$ and $D = V'$ and $w_e = 1$ for each $e \in \overleftrightarrow{E}'$. Since v is an in-neighbor and an out-neighbor of all other nodes in $\overleftrightarrow{G}'$ and since $v \in D$, it is clear that D is a latency-s CDS for $\overleftrightarrow{G}'$. It can be observed that $D \setminus \{v\}$ is a latency-s CDS for $\overleftrightarrow{G}'$ if and only if G has $\text{diam}(G) = 2$, and this reduction runs in linear time, so the proposition follows. □

2 Strength of CUT vs. POLY

Figure 1 shows that POLY cannot be stronger than CUT. We were unable to prove/disprove that CUT is stronger than POLY.

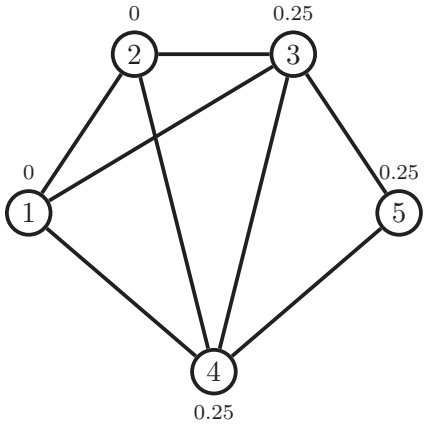

Figure 1: A graph with nodes numbered $\{1, 2, 3, 4, 5\}$ and values for x^* given by the nodes. When the edges shown are treated as bidirected and unit-weighted, this vector x^* is infeasible for formulation CUT (for every s), since the length-s vertex cut inequality for $C = \{3, 4\}$ is violated. However, there exist values y^* and z^* for which (x^*, y^*, z^*) satisfies formulation POLY when $s = 3$. Indeed, these values for x^* are *optimal* for POLY's LP relaxation. The solve logs, obtained via Gurobi, are below.

```
Optimize a model with 230 rows, 80 columns and 590 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 131 rows and 45 columns
Presolve time: 0.00s
Presolved: 99 rows, 35 columns, 241 nonzeros
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.000000e+00   0.000000e+00      0s
      20    5.0000000e-01   0.000000e+00   0.000000e+00      0s
Solved in 20 iterations and 0.00 seconds
Optimal objective  5.000000000e-01

X_1 = 0
X_2 = 0
X_3 = 0.25
X_4 = 0.25
X_5 = 0
Y_{1,2}^2 = 0.25
Y_{1,3}^2 = 0.25
Y_{1,4}^2 = 0.25
```

```
Y_{1,5}^2 = 0.5
Y_{2,1}^2 = 0.25
Y_{2,3}^2 = 0.25
Y_{2,4}^2 = 0.25
Y_{2,5}^2 = 0.5
Y_{3,1}^2 = 0.25
Y_{3,2}^2 = 0.25
Y_{3,4}^2 = 0
Y_{3,5}^2 = 0.25
Y_{4,1}^2 = 0.25
Y_{4,2}^2 = 0.25
Y_{4,3}^2 = 0
Y_{4,5}^2 = 0.25
Y_{5,1}^2 = 0.5
Y_{5,2}^2 = 0.5
Y_{5,3}^2 = 0.25
Y_{5,4}^2 = 0.25
Y_{1,2}^3 = 0
Y_{1,3}^3 = 0
Y_{1,4}^3 = 0
Y_{1,5}^3 = 0.5
Y_{2,1}^3 = 0
Y_{2,3}^3 = 0
Y_{2,4}^3 = 0
Y_{2,5}^3 = 0.5
Y_{3,1}^3 = 0
Y_{3,2}^3 = 0
Y_{3,4}^3 = 0
Y_{3,5}^3 = 0
Y_{4,1}^3 = 0
Y_{4,2}^3 = 0
Y_{4,3}^3 = 0
Y_{4,5}^3 = 0
Y_{5,1}^3 = 0.5
Y_{5,2}^3 = 0.5
Y_{5,3}^3 = 0
Y_{5,4}^3 = 0
Z_{1,2}^2 = 0
Z_{1,3}^2 = 0
Z_{1,4}^2 = 0
Z_{1,5}^2 = 0
Z_{2,1}^2 = 0
Z_{2,3}^2 = 0
Z_{2,4}^2 = 0
Z_{2,5}^2 = 0
Z_{3,1}^2 = 0.25
Z_{3,2}^2 = 0.25
Z_{3,4}^2 = 0
```

```
Z_{3,5}^2 = 0.25
Z_{4,1}^2 = 0.25
Z_{4,2}^2 = 0.25
Z_{4,3}^2 = 0
Z_{4,5}^2 = 0.25
Z_{5,1}^2 = 0
Z_{5,2}^2 = 0
Z_{5,3}^2 = 0
Z_{5,4}^2 = 0
```

3 The Importance of Strengthening the Inequalities

The following algorithm can be used to strengthen length-s vertex cut inequalities, i.e., to find *inclusion-minimal* length-s vertex cuts. For our purposes, it is best described in terms of a "bad" vertex subset $B \subseteq V$, i.e., one that is not a latency-s CDS for G. (If given a length-s vertex cut, initialize the algorithm using its complement.) The algorithm takes B as input and returns an inclusion-minimal length-s vertex cut $C \subseteq V \setminus B$ for G.

MinimalizeBasic(G, B, s):

1. $Q \leftarrow V \setminus B$;
2. for $q \in Q$ do
   - if $B \cup \{q\}$ is **not** a latency-s CDS for G then update $B \leftarrow B \cup \{q\}$;
3. return $C := V \setminus B$.

Proposition 3. *Algorithm* MinimalizeBasic *finds a minimal length-s vertex cut in time $O(n^4)$.*

Proof. The runtime is dominated by the $|Q|$ different calls to IsLatencyConstrainedCDS, which take time $O(|Q|n^3) = O(n^4)$. Since latency-s CDS's are closed under taking supersets (as a consequence of Proposition 3 of the paper), it can be argued that, in step 3, $B \cup \{v\}$ is a latency-s CDS for every $v \in V \setminus B$. Further, B is not. Thus, $|D \cap C| \geq 1$ for every latency-s CDS $D \subseteq V$, and no proper subset of C satisfies this property. So, by Lemma 1 of the paper, C is a minimal length-s vertex cut. □

The algorithm MinimalizeBasic for finding a minimal length-s vertex cut runs in time $O(n^4)$. Here, we provide an improved $O(n^3)$ implementation. It is based on maintaining the collection F of "far" pairs of vertices, i.e., pairs of vertices that cannot communicate quickly enough through $B \subseteq V$.

First we provide pseudocode for the subroutine EnumerateFarPairs, which finds all far pairs. It is similar to IsLatencyConstrainedCDS described earlier but requires more space, since it requests the set F (and not just a query as to whether F is empty). Recall the subgraph of G denoted $\overrightarrow{G}_v^B$ whose edge set $\overrightarrow{E}_v^B$ was defined in equation 7 of the paper. This is used to compute shortest paths from v to all other nodes, using nodes from B as intermediaries.

EnumerateFarPairs(G, B, s):

1. $F \leftarrow \emptyset$;
2. for each $v \in V$ do
   - compute shortest paths from v in digraph $\overrightarrow{G}_v^B$;
   - for $t \in V \setminus \{v\}$ do
     - if $\text{dist}_{\overrightarrow{G}_v^B}(v,t) > s$ then $F \leftarrow F \cup \{(v,t)\}$;
3. return F.

Now we give an improved implementation of MinimalizeBasic. In its pseudocode, we need notation for another subgraph of G, denoted $\overleftarrow{G}_v^B = (V, \overleftarrow{E}_v^B)$, that is used when computing the shortest paths from all other nodes to v, using only nodes from B as intermediaries. Its edge set $\overleftarrow{E}_v^B$ contains all edges that might be used in one of these paths.

$$\overleftarrow{E}_v^B := E(B) \cup \delta^-(B) \cup \left(\delta^+(B) \cap \delta^-(v)\right). \tag{1}$$

Minimalize(G, B, s):

1. $Q \leftarrow V \setminus B$ and $F \leftarrow$EnumerateFarPairs(G, B, s);
2. for $v \in Q$ do
   - compute shortest paths from v in graph $\overrightarrow{G}_v^B$;
   - compute shortest paths to v in graph $\overleftarrow{G}_v^B$;
   - $R \leftarrow \{(a,b) \in F \mid \text{dist}_{\overleftarrow{G}_v^B}(a,v) + \text{dist}_{\overrightarrow{G}_v^B}(v,b) \leq s\}$;
   - if $F \neq R$ then update $B \leftarrow B \cup \{v\}$ and $F \leftarrow F \setminus R$;
3. return $C := V \setminus B$.

Here, $R \subseteq F$ represents the subset of far pairs that would no longer be far if paths could cross node v. If the sets R and F are the same, then this means that $B \cup \{v\}$ a latency-s CDS (which we do not want), otherwise the algorithm adds v to B, thus moving to the larger infeasible set $B \cup \{v\}$—whose complement is a smaller length-s vertex cut.

Theorem 2. *Algorithm* Minimalize *finds a minimal length-s vertex cut in time $O(n^3)$ and space $O(n^2)$, or, more precisely, in:*

- *$O(nm + n^2 \log\log n + |Q||F_0|)$ time and $O(m + |F_0|)$ space under nonnegative weights;*
- *$O(nm + |Q||F_0|)$ time and $O(m + |F_0|)$ space in the hop-based case.*

Here, F_0 denotes the original set of far pairs, i.e., the set F from step 1.

In Table 1, we compare the performance of formulation CUT with three different settings: (1) without minimalizing the length-s vertex cuts, (2) applying algorithm MinimalizeBasic, and (3) applying algorithm Minimalize. In these tests, we set $s = \text{diam}(G)$ since this is the smallest feasible value of s. Moreover, we exclude the cases where $s = \text{diam}(G) = 2$, as there is nothing to minimalize. Note that these tests use the BestInHeuristic that is described in Section 5. CUT only solves 4 instances if we do not minimalize the length-s vertex cuts, in which case the formulation is practically useless. Using MinimalizeBasic, we can solve 20 of the 30 instances, but the callback time is large in many cases. For example, see the instance v200_d20 in which 37 minutes is spent in the callback. If Minimalize is used instead, the callback time reduces to 1 minute. Moreover, the time saved by Minimalize allows the solver to improve the lower bound in 5 of the 10 unsolved cases.

Table 1: An evaluation of the usefulness of MINIMALIZE as compared to no minimalizing and to MINIMALIZEBASIC. For all graphs G, we set $s = \text{diam}(G)$ and exclude instances where $s = 2$. We report the optimal objective (or the best lower/upper bounds $[L, U]$ after one hour) under the columns labeled obj. We also give the time spent in the callback (call), and the total solve time (total), where a dash indicates > 3600 seconds.

		No minimalizing			MINIMALIZEBASIC			MINIMALIZE		
graph	s	obj	call	total	obj	call	total	obj	call	total
v30_d10	8	15	10.29	280.24	15	0.00	0.01	15	0.00	0.02
v30_d20	5	8	0.01	0.05	8	0.02	0.06	8	0.00	0.03
v30_d30	3	[7,8]	229.27	-	8	0.11	0.21	8	0.02	0.10
v50_d5	14	[26,32]	49.88	-	32	0.02	0.07	32	0.02	0.07
v50_d10	5	[13,20]	31.27	-	18	0.44	0.58	18	0.08	0.18
v50_d20	3	[8,14]	51.24	-	14	0.16	0.25	14	0.03	0.11
v50_d30	3	[6,8]	45.67	-	8	1.69	2.70	8	0.16	1.12
v70_d5	8	[24,36]	14.41	-	32	1.13	1.36	32	0.32	0.53
v70_d10	4	[14,31]	19.93	-	29	3.14	5.77	29	0.44	2.98
v70_d20	3	[8,18]	19.60	-	17	47.27	335.22	17	3.40	305.96
v70_d30	3	[6,8]	66.95	-	7	9.68	12.93	7	0.50	3.33
v100_d5	5	[24,57]	9.71	-	56	16.38	33.14	56	2.10	17.91
v100_d10	4	[14,31]	10.82	-	[22,26]	322.14	-	[22,26]	18.79	-
v100_d20	3	[8,20]	15.26	-	[14,20]	507.29	-	[14,20]	24.16	-
v120_d5	6	[25,40]	8.80	-	31	464.72	1511.87	31	25.86	1087.17
v120_d10	3	[13,68]	7.92	-	63	27.31	35.60	63	2.21	10.31
v120_d20	3	[8,21]	13.89	-	[10,21]	973.37	-	[10,21]	42.57	-
v120_d30	3	[6,12]	19.98	-	[7,12]	737.91	-	[7,12]	21.65	-
v150_d5	5	[25,54]	7.19	-	[29,54]	1088.47	-	[30,54]	121.73	-
v150_d10	3	[13,65]	7.35	-	[41,63]	717.64	-	[43,61]	39.95	-
v150_d20	3	[8,22]	12.70	-	[9,22]	1915.12	-	[9,22]	58.24	-
v200_d5	4	[25,92]	6.10	-	[48,92]	1851.87	-	[49,92]	169.57	-
v200_d10	3	[13,64]	6.26	-	[23,64]	1734.04	-	[26,64]	132.19	-
v200_d20	3	[7,22]	7.58	-	[7,22]	2226.01	-	[8,22]	58.71	-
IEEE-14	5	5	0.00	0.00	5	0.00	0.00	5	0.00	0.01
IEEE-30	6	14	0.33	3.98	14	0.00	0.01	14	0.00	0.01
IEEE-57	12	[24,35]	23.24	-	35	0.02	0.05	35	0.01	0.04
RTS-96	13	[27,40]	13.53	-	37	0.09	0.17	37	0.07	0.14
IEEE-118	14	[39,48]	15.22	-	48	0.01	0.17	48	0.00	0.15
IEEE-300	24	[104,139]	16.49	-	135	14.35	18.00	135	8.65	11.98

4 Fractional Separation

We followed a suggestion of one of the reviewers to use fractional separation. Specifically, we implemented the max-flow/min-cut-based procedure for $s = 3$, as described in the paper, and used it as part of a branch-and-cut approach.

When experimenting with fractional separation, we felt it was important to try different implementations, as the number of cuts and the cut violation can greatly impact the performance. With this in mind, we tried nine different implementations.

The generic pseudocode that we use in the callback is as follows, where x^* is the branch-and-bound node's LP solution, $\epsilon \in \{0.01, 0.1, 0.5\}$ is the cut violation threshold, and $I \in \{1, 2, 3\}$ controls how many cuts are added per callback.

- for every vertex $a \in \{1, 2, \ldots, n\}$ do
  - for every vertex $b \in \{1, 2, \ldots, n\}$ do
    - compute a minimum-weight length-3 a,b-separator $C \subseteq V \setminus \{a, b\}$ in graph $G = (V, E)$, where each vertex i has weight x^*_i;
    - if the cut violation $1 - x^*(C)$ is more than ϵ,
      - add the cut $x(C) \geq 1$;
      - if $I = 1$, then return;
      - if $I = 2$, then break the for-loop over b (i.e., go to the next a);

As can be observed in the pseudocode, setting $I = 1$ adds ≤ 1 cut per callback, setting $I = 2$ adds $\leq n$ cuts per callback, and setting $I = 3$ adds $\leq n^2$ cuts per callback.

There are several ways to speed up separation in practice. In all of our implementations, we use the following speedups:

1. We can skip the cases $a = b$.
2. Our test instances are bidirected, so it suffices to consider $b \in \{a + 1, a + 2, \ldots, n\}$.
3. If $x^*(N^+(a) \cap N^-(b)) + \epsilon \geq 1$, then no sufficiently violated cut will exist, and one can skip to the next vertex b.

We tried each of the three settings $I \in \{1, 2, 3\}$ across three different values of $\epsilon \in \{0.01, 0.1, 0.5\}$, resulting in a total of nine implementations. The results are provided in Tables 2, 3, 4. In each table, we report the lower and upper bounds on the objective value after a one-hour time limit. For reference, we also report the bounds obtained using our integer separation routine that we have been using all along.

Among the nine fractional separation implementations, there is no clear winner. For the graph v150_d10, the best bounds are achieved with the parameter settings $(I, \epsilon) \in \{(1, 0.01), (3, 0.01), (3, 0.1)\}$. While, for the graph v200_d20, the best bounds are achieved with a different parameter setting $(I, \epsilon) = (1, 0.1)$. However, if we include our original implementation into the mix, it is the clear winner. This is despite a good-faith effort towards using fractional separation, including a polynomial-time separation routine for $s = 3$ and helpful speedups. We imagine that the results would be even worse if we had to perform exact separation for $s \geq 5$, where separation is NP-hard. This confirms our suspicions that we had obtained when studying another semi-related problem (Buchanan and Salemi, 2018).

One explanation for the poor performance of $s = 3$ fractional separation is the time to solve the separation problem. It requires the solution of up to $\Theta(n^2)$ different minimum cut problems,

making the runtime something like mn^3. In contrast, our integer separation routine requires $O(n^3)$ time. Note that for the $s = 3$ instance v200_d20, $n^3 = 8,000,000$, while $mn^3 = 31,840,000,000$. Any time spent on the time mn^3 procedure in the callback is time not spent on other things (like branching). Apparently, this tradeoff is not worth it.

As a final remark, see that several of our instances remain unsolved even when $s = 2$. Recall that we add all length-2 vertex cut inequalities upfront in the $s = 2$ case since there are only $O(n^2)$ of them, in which case there is no need to perform fractional separation. Thus, even when fractional separation takes "zero time", the instances cannot be solved. This lends credence to the idea that there is little room for fractional separation to help; these instances may just be challenging. We note that one of our $s = 2$ instances was submitted to MIPLIB 2017 and has been included in their Collection Set. At the time of writing, it is still considered an "open" instance, meaning that no MIP solver has been able to solve it (even with the latest software releases). The instance (and its "open" status) can be found on the MIPLIB website: `http://miplib.zib.de/instance_details_v150d30-2hopcds.html`

Table 2: Lower and upper bounds after one hour when cut violation threshold is $\epsilon = 0.01$

graph	s	Lazy	$I = 1$ ≤ 1	$I = 2$ $\leq n$	$I = 3$ All Pairs
v100_d20	3	[14,20]	[11,20]	[10,20]	[11,20]
v120_d20	3	[10,21]	[8,21]	[7,21]	[8,21]
v120_d30	3	[7,12]	[4,12]	[4,12]	[4,12]
v150_d10	3	[43,61]	[38,65]	[35,65]	[38,65]
v150_d20	3	[9,22]	[7,22]	[7,22]	[7,22]
v200_d10	3	[26,64]	[27,64]	[24,64]	[27,64]
v200_d20	3	[8,22]	[6,22]	[6,22]	[6,22]

Table 3: Lower and upper bounds after one hour when cut violation threshold is $\epsilon = 0.1$

graph	s	Lazy	$I = 1$ ≤ 1	$I = 2$ $\leq n$	$I = 3$ All Pairs
v100_d20	3	[14,20]	[10,20]	[10,20]	[11,20]
v120_d20	3	[10,21]	[9,21]	[7,21]	[8,21]
v120_d30	3	[7,12]	[6,12]	[4,12]	[4,12]
v150_d10	3	[43,61]	[28,65]	[37,65]	[38,65]
v150_d20	3	[9,22]	[8,22]	[7,22]	[7,22]
v200_d10	3	[26,64]	[19,64]	[24,64]	[26,64]
v200_d20	3	[8,22]	[7,22]	[6,22]	[6,22]

Table 4: Lower and upper bounds after one hour when cut violation threshold is $\epsilon = 0.5$

graph	s	Lazy	$I = 1$ ≤ 1	$I = 2$ $\leq n$	$I = 3$ All Pairs
v100_d20	3	[14,20]	[11,20]	[9,20]	[10,20]
v120_d20	3	[10,21]	[8,21]	[7,21]	[7,21]
v120_d30	3	[7,12]	[4,12]	[4,12]	[4,12]
v150_d10	3	[43,61]	[28,65]	[37,65]	[36,65]
v150_d20	3	[9,22]	[7,22]	[6,22]	[6,22]
v200_d10	3	[26,64]	[20,64]	[23,64]	[24,64]
v200_d20	3	[8,22]	[6,22]	[6,22]	[6,22]

5 The Importance of Providing a Heuristic Solution to the Solver

We provide a simple "best-in" heuristic for the minimum latency-s CDS problem. In the pseudocode, the vertex subset D represents a partial solution; the set F represents the set of "far pairs," i.e., those pairs of vertices that inhibit D from being a feasible solution; and the score of a vertex is how many far pairs it would "close" or eliminate. Note that the scores can increase during the heuristic, and they can all be zero while F is nonempty.

BestInHeuristic(G, s):

1. compute $\text{diam}(G)$;
2. if $\text{diam}(G) > s$, return "infeasible";
3. initialize $D \leftarrow \emptyset$ and $F \leftarrow \{(i,j) \in V \times V \mid (i,j) \notin E,\ i \neq j\}$;
4. while $F \neq \emptyset$ do
   (a) for $v \in V \setminus D$ do
   - compute shortest paths from v in graph $\overrightarrow{G}_v^D$;
   - compute shortest paths to v in graph $\overleftarrow{G}_v^D$;
   - compute $\text{score}(v) := \left|\left\{(a,b) \in F \;\middle|\; \text{dist}_{\overleftarrow{G}_v^D}(a,v) + \text{dist}_{\overrightarrow{G}_v^D}(v,b) \leq s\right\}\right|$;

   (b) let $v^* \in V \setminus D$ be a vertex of maximum score;
   (c) $R \leftarrow \left\{(a,b) \in F \;\middle|\; \text{dist}_{\overleftarrow{G}_{v^*}^D}(a,v^*) + \text{dist}_{\overrightarrow{G}_{v^*}^D}(v^*,b) \leq s\right\}$;
   (d) update $D \leftarrow D \cup \{v^*\}$ and $F \leftarrow F \setminus R$;
5. return D.

This heuristic returns a feasible solution (when the instance is feasible), and its runtime is $O(n^4)$, or, more precisely, $O(n^3 + |D|mn + |D||F_0|n)$, where D is the heuristic solution at the end, and F_0 is the initial set of far pairs from step 3. Given that it takes time $O(n^3)$ to verify feasibility, this heuristic is not too costly. The output of this heuristic is not necessarily inclusion-minimal, so we run a post-processing procedure to make it so.

We also experimented with a "worst-out" heuristic that starts with V as an initial solution and greedily deletes vertices v, in order of decreasing indegree $|N^-(v)|$ plus outdegree $|N^+(v)|$, as long as this is still feasible. This heuristic, when applied to the minimum CDS problem, is "provably best" in the sense of Kahruman-Anderoglu et al. (2016) meaning that, unless P=NP, no polynomial-time algorithm always finds a better solution than this heuristic (when a better solution exists). Similar heuristics work well in practice for the minimum CDS problem (Butenko et al., 2004). However, this particular worst-out heuristic performed poorly compared to the best-in heuristic in initial experiments, so it is excluded.

In Table 5, we evaluate the usefulness of the proposed best-in heuristic (while employing MINIMALIZE to strengthen the inequalities). In most cases, the impact of the heuristic is marginal (positively or negatively). Of the instances that were solved to optimality, its effect is most pronounced on v70_d20 and v120_d5 where it slowed down the solver by 173 seconds and sped up the solver by 409 seconds, respectively. It may seem strange that providing an MIP start could worsen the performance, but this is likely a natural consequence of solver variability, and we did not attempt to tame or take advantage of this behavior. Note, however, that Gurobi was unable to find a feasible solution on four instances if left unaided. Further, on the instances v150_d20 and v200_d20 the best-in heuristic found solutions (in under a second) that were 12 vertices and 13 vertices better, respectively, than what Gurobi found in an hour. Frankly, Gurobi's performance

on some of the $s = 2$ instances surprised us. For example, on the instance v120_d50, CUT has only 120 binary variables and 3570 covering constraints, and yet it did not solve within an hour.

Table 5: An evaluation of the usefulness of BestInHeuristic as compared to running Gurobi without an MIP start. We report the objective of the heuristic solution (hobj) and the time spent in the heuristic (htime). For the meanings of other reported quantities, refer to Table 1.

		No heuristic		with BestInHeuristic			
graph	s	obj	total	hobj	obj	htime	total
v30_d10	8	15	0.01	15	15	0.00	0.02
v30_d20	5	8	0.09	8	8	0.00	0.03
v30_d30	3	8	0.16	8	8	0.00	0.10
v30_d50	2	7	0.02	7	7	0.00	0.01
v30_d70	2	3	0.02	4	3	0.00	0.04
v50_d5	14	32	0.05	32	32	0.01	0.07
v50_d10	5	18	0.18	20	18	0.01	0.18
v50_d20	3	14	0.14	14	14	0.00	0.11
v50_d30	3	8	7.23	8	8	0.00	1.12
v50_d50	2	9	0.18	11	9	0.00	0.17
v50_d70	2	4	0.89	4	4	0.00	0.79
v70_d5	8	32	0.44	36	32	0.01	0.53
v70_d10	4	29	1.34	31	29	0.01	2.98
v70_d20	3	17	132.69	18	17	0.01	305.96
v70_d30	3	7	1.95	8	7	0.00	3.33
v70_d50	2	10	0.76	10	10	0.00	0.56
v70_d70	2	5	2.04	5	5	0.00	1.57
v100_d5	5	56	17.02	57	56	0.04	17.91
v100_d10	4	[22,26]	-	31	[22,26]	0.02	-
v100_d20	3	[14,18]	-	20	[14,20]	0.01	-
v100_d30	2	39	5.35	42	39	0.03	5.00
v100_d50	2	12	59.15	13	12	0.01	61.27
v100_d70	2	5	8.91	5	5	0.00	8.17
v120_d5	6	31	1496.49	40	31	0.05	1087.17
v120_d10	3	63	10.33	68	63	0.06	10.31
v120_d20	3	[10,26]	-	21	[10,21]	0.02	-
v120_d30	3	[7,12]	-	12	[7,12]	0.01	-
v120_d50	2	[11,12]	-	13	[11,12]	0.01	-
v120_d70	2	5	32.86	6	5	0.00	29.67
v150_d5	5	[30,∞]	-	54	[30,54]	0.10	-
v150_d10	3	[39,∞]	-	65	[43,61]	0.10	-
v150_d20	3	[9,34]	-	22	[9,22]	0.04	-
v150_d30	2	[35,42]	-	45	[35,41]	0.06	-
v150_d50	2	[9,13]	-	13	[9,13]	0.02	-
v150_d70	2	6	560.68	7	6	0.01	569.10
v200_d5	4	[46,∞]	-	92	[49,92]	0.34	-
v200_d10	3	[23,∞]	-	64	[26,64]	0.20	-
v200_d20	3	[7,35]	-	22	[8,22]	0.07	-
v200_d30	2	[27,43]	-	45	[27,44]	0.12	-
v200_d50	2	[8,15]	-	16	[8,15]	0.04	-
v200_d70	2	[4,6]	-	7	[4,7]	0.01	-
IEEE-14	5	5	0.00	5	5	0.00	0.01
IEEE-30	6	14	0.01	14	14	0.00	0.01
IEEE-57	12	35	0.02	35	35	0.01	0.04
RTS-96	13	37	0.15	40	37	0.02	0.14
IEEE-118	14	48	0.07	48	48	0.06	0.15
IEEE-300	24	135	8.91	139	135	2.31	11.98

6 Results for the Fault-Tolerant Variant

In Table 6, we provide results for the fault-tolerant variant, where $r = 2$, and s is set to the smallest feasible value[1] $\max\{\text{diam}(G - v) \mid v \in V\}$. The reason for including results for $r = 2$ (and not for larger values) is that network topologies that continue to function after the failure of a single entity are often considered sufficient for practical purposes (Monma and Shallcross, 1989; Grötschel et al., 1992).

The implementation needed to be modified to accommodate $r = 2$. For one, we needed to alter the heuristic. Our approach is to first find a feasible solution $D \subseteq V$ for $r = 1$ using the BestInHeuristic from before. Then we consider some $v \in D$ and check if $D \setminus \{v\}$ is a latency-s CDS. If not, then we augment it by adding high *score* vertices from $V \setminus D$ in the same way as in BestInHeuristic until the failure of $v \in D$ will not cause problems. We do this for each vertex $v \in D$ from the initial solution (for $r = 1$) until our heuristic solution is feasible for $r = 2$.

We also had to modify the callback routines. When the solver encounters a vertex subset D that satisfies the initial constraints, we check if it is a latency-s CDS (for $r = 1$). If not then $V \setminus D$ is a length-s vertex cut, and we strengthen it using Minimalize as before. Supposing D is a latency-s CDS, then we check if it is a solution for $r = 2$, i.e., if $D \setminus \{v\}$ is a latency-s CDS for each $v \in D$. If not, then $(V \setminus D) \cup \{v\}$ is a length-s vertex cut, and we use Minimalize on it.

References

Austin Buchanan and Hosseinali Salemi. Parsimonious formulations for low-diameter clusters, 2018. `http://www.optimization-online.org/DB_HTML/2017/09/6196.html`.

Sergiy Butenko, Xiuzhen Cheng, Carlos A Oliveira, and Panos M Pardalos. A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks. In *Recent developments in cooperative control and optimization*, pages 61–73. Springer, 2004.

Martin Grötschel, Clyde L Monma, and Mechthild Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40(2):309–330, 1992.

Sera Kahruman-Anderoglu, Austin Buchanan, Sergiy Butenko, and Oleg A. Prokopyev. On provably best construction heuristics for hard combinatorial optimization problems. *Networks*, 67(3): 238–245, 2016.

Daniel Lokshtanov, Dániel Marx, Saket Saurabh, et al. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 3(105), 2013.

Clyde L Monma and David F Shallcross. Methods for designing communications networks with certain two-connected survivability constraints. *Operations Research*, 37(4):531–541, 1989.

Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524. ACM, 2013.

[1] If the graph has a cut vertex, we exclude it from consideration since there is no feasible (finite) value for s.

Table 6: Results for the fault-tolerant variant, where $r = 2$ and $s = \max\{\mathrm{diam}(G - v) \mid v \in V\}$. Only biconnected graphs are considered. The number of branch-and-bound nodes is "BB node."

graph	diam	s	BB node	htime	hobj	obj	total
v30_d30	3	3	462	0.00	12	12	0.09
v30_d50	2	2	0	0.01	15	14	0.02
v30_d70	2	2	0	0.00	6	5	0.02
v50_d10	5	5	239	0.12	34	32	0.33
v50_d20	3	4	432	0.03	15	13	0.25
v50_d30	3	3	10625	0.01	13	12	2.28
v50_d50	2	2	9	0.02	18	15	0.19
v50_d70	2	2	0	0.00	7	6	0.33
v70_d5	8	10	201	0.27	53	51	0.61
v70_d10	4	5	7519	0.13	33	27	4.44
v70_d20	3	3	384357	0.12	31	26	106.28
v70_d30	3	3	29771	0.02	14	12	7.30
v70_d50	2	2	174	0.03	18	16	0.56
v70_d70	2	2	163	0.01	9	7	1.77
v100_d5	5	6	134933	0.71	63	56	102.40
v100_d10	4	4	943671	0.29	48	[38,41]	-
v100_d20	3	3	1260796	0.15	32	[24,27]	-
v100_d30	2	3	366610	0.07	18	[11,16]	-
v100_d50	2	2	21597	0.07	19	18	36.17
v100_d70	2	2	4146	0.02	10	8	14.25
v120_d5	6	7	134501	1.22	59	50	164.92
v120_d10	3	4	570958	0.56	42	[28,39]	-
v120_d20	3	3	357951	0.28	32	[18,30]	-
v120_d30	3	3	265467	0.08	19	[10,17]	-
v120_d50	2	2	1109463	0.13	22	[16,18]	-
v120_d70	2	2	6075	0.02	10	8	37.63
v150_d5	5	5	388468	2.44	80	[53,80]	-
v150_d10	3	4	283107	0.49	45	[25,45]	-
v150_d20	3	3	273843	0.51	36	[16,34]	-
v150_d30	2	3	244025	0.20	19	[9,19]	-
v150_d50	2	2	222265	0.18	22	[15,18]	-
v150_d70	2	2	628745	0.04	11	[8,9]	-
v200_d5	4	4	644105	12.81	128	[104,118]	-
v200_d10	3	3	426782	5.56	95	[55,95]	-
v200_d20	3	3	224602	0.79	32	[13,32]	-
v200_d30	2	2	77784	6.57	72	[49,63]	-
v200_d50	2	2	33061	0.64	25	[13,22]	-
v200_d70	2	2	119071	0.07	12	[7,9]	-