Relative Survivable Network Design

Michael Dinitz
Johns Hopkins University, Baltimore, MD, USA

Ama Koranteng
Johns Hopkins University, Baltimore, MD, USA

Guy Kortsarz
Rutgers University, Camden, NJ, USA

Abstract

One of the most important and well-studied settings for network design is edge-connectivity requirements. This encompasses uniform demands such as the Minimum k-Edge-Connected Spanning Subgraph problem (k-ECSS), as well as nonuniform demands such as the Survivable Network Design problem. A weakness of these formulations, though, is that we are not able to ask for fault-tolerance larger than the connectivity. Taking inspiration from recent definitions and progress in graph spanners, we introduce and study new variants of these problems under a notion of *relative* fault-tolerance. Informally, we require not that two nodes are connected if there are a bounded number of faults (as in the classical setting), but that two nodes are connected if there are a bounded number of faults *and the two nodes are connected in the underlying graph post-faults.* That is, the subgraph we build must "behave" identically to the underlying graph with respect to connectivity after bounded faults.

We define and introduce these problems, and provide the first approximation algorithms: a $(1 + 4/k)$-approximation for the unweighted relative version of k-ECSS, a 2-approximation for the weighted relative version of k-ECSS, and a 27/4-approximation for the special case of Relative Survivable Network Design with only a single demand with a connectivity requirement of 3. To obtain these results, we introduce a number of technical ideas that may of independent interest. First, we give a generalization of Jain's iterative rounding analysis that works even when the cut-requirement function is not weakly supermodular, but instead satisfies a weaker definition we introduce and term *local* weak supermodularity. Second, we prove a structure theorem and design an approximation algorithm utilizing a new decomposition based on *important separators*, which are structures commonly used in fixed-parameter algorithms that have not commonly been used in approximation algorithms.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Fault Tolerance, Network Design

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2022.41

Category APPROX

Related Version *Full Version*: https://arxiv.org/abs/2206.12245 [13]

Funding *Michael Dinitz*: Supported in part by NSF award CCF-1909111.
Ama Koranteng: Supported in part by NSF award CCF-1909111.

1 Introduction

Fault-tolerance has been a central object of study in approximation algorithms, particularly for network design problems where the graphs that we study represent some physical objects which might fail (communication links, transportation links, etc.). In these settings it is natural to ask for whatever object we build to be fault-tolerant. The precise definition of "fault-tolerance" is different in different settings, but a common formulation is edge fault-tolerance, which typically takes the form of edge connectivity. Informally, these look like guarantees of the form "if up to k edges fail, then the nodes I want to be connected are still connected." For example, consider the following two classical problems.

- The Minimum k-Edge Connected Subgraph problem (k-ECSS), where we are given a graph G and a value k and are asked to find the k-edge connected subgraph of G of minimum size (or weight). In other words, if fewer than k edges fail, the graph should still be connected.
- The more general Survivable Network Design problem (SND, sometimes referred to as Generalized Steiner Network), where we are given a graph G and demands $\{(s_i, t_i, k_i)\}_{i\in[\ell]}$, and are supposed to find the minimum-weight subgraph H of G so that there are at least k_i edge-disjoint paths between s_i and t_i for every $i \in [\ell]$. In other words, for every $i \in [\ell]$, if fewer than k_i edges fail then s_i and t_i will still be connected in H even after failures.

Both of these problems have been studied extensively (for a small sample, see [24, 19, 10, 17]), and are paradigmatic examples of network design problems. But there is a different notion of fault-tolerance which is stronger, and in some ways more natural: *relative* fault-tolerance. Relative fault-tolerance makes guarantees that rather than being absolute ("if at most k edges fail the network still functions") are relative to an underlying graph or system ("if at most k edges fail, the subgraph functions just as well as the original graph post-failures"). This allows us to generalize the traditional definition: if the underlying graph has strong enough connectivity properties then the two definitions are the same, but the relative version allows us to make interesting and nontrivial guarantees even when the underlying graph does not have strong connectivity properties.

For example, the definition of Survivable Network Design has an important limitation: if G itself can only support a small number of edge disjoint $s_i - t_i$ paths (e.g., 3), then of course we cannot ask for a subgraph with more edge-disjoint paths. There simply would be no feasible solution. But this is somewhat unsatisfactory. For example, while we cannot guarantee that s_i and t_i would be connected after *any* set of 5 faults (since those faults may include an (s_i, t_i) cut of size 3), clearly there could be *some* set of 5 faults which do not in fact disconnect s_i from t_i in G. And if these faults occur, it is natural to want s_i and t_i to still be connected in (what remains) of H. In other words: just because there exists a small cut, why should we give up on being tolerant to a larger number of faults which do not contain that cut?

1.1 Our Results and Techniques

In this paper we initiate the study of relative fault-tolerance in network design, by defining relative versions of Survivable Network Design and k-ECSS.

▶ **Definition 1.** *In the Relative Survivable Network Design problem (RSND), we are given a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{\geq 0}$ and demands $\{(s_i, t_i, k_i)\}_{i\in[\ell]}$. A feasible solution is a subgraph H of G where for all $i \in [\ell]$ and $F \subseteq E$ with $|F| < k_i$, if there is a path in $G \setminus F$ from s_i to t_i then there is also a path in $H \setminus F$ from s_i to t_i. Our goal is to find the minimum weight feasible solution.*

▶ **Definition 2.** *The k-Edge Fault-Tolerant Subgraph problem (k-EFTS) is the special case of RSND where there is a demand between all pairs and every k_i is equal to k. In other words, we are given a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{\geq 0}$. A feasible solution is a subgraph H of G where for all $F \subseteq E$ with $|F| < k$, any two nodes which have a path between them in $G \setminus F$ also have a path between them in $H \setminus F$ (the connected components of $H \setminus F$ are identical to the connected components of $G \setminus F$). Our goal is to find the minimum weight feasible solution.*

For both of these problems, we say that they are *unweighted* if all edges have the same weight (or equivalently $w(e) = 1$ for all $e \in E$). Note that if s_i and t_i are k_i-connected in G for every $i \in [\ell]$, then RSND is exactly the same as SND, and if G is k-connected then k-EFTS is exactly the same as k-ECSS. Hence we have generalized these classical problems.

We note that the fault-tolerance we achieve is really "one less" than the given number (there are strict inequalities in the definitions). This is "off-by-one" from the related relative fault-tolerance literature [9, 5, 6], but makes the connection to SND and k-ECSS cleaner.

Difficulties. Before discussing our results or techniques, we briefly discuss what makes these problems difficult. The non-relative versions are classical and have been studied extensively: why can't we just re-use the ideas and techniques developed for them? Particularly since there is only a difference in the setting when there are small cuts in the graph, in which case we already know that the edges of those cuts must be included in any feasible solution?

Unfortunately, it turns out that this seemingly minor change has a dramatic impact on the structure of the problem. Most importantly, the *cut requirement function* has dramatically different properties. In k-ECSS, Menger's theorem implies that H is a valid solution if and only if for all $S \subset V$ with $S \neq \emptyset$, there are at least k edges between S and $\bar{S}$. Hence we can rephrase k-ECSS as the problem of finding a minimum cost subgraph such that that there are at least $f(S)$ edges across the cut $(S, \bar{S})$ for all $S \subset V$ with $S \neq \emptyset$, where $f(S) = k$. Similarly, we can rephrase SND as the same problem but where $f(S) = \max_{i \in [\ell]: s_i \in S, t_i \notin S} k_i$ (as was shown in [24]). Thus both problems can be thought of as choosing a minimum cost subgraph subject to satisfying some cut-requirement covering function $f : 2^V \to \mathbb{R}$. So a natural starting point for any approximation algorithm is to write the natural covering LP relaxation which has a covering constraint of $f(S)$ for every cut S. And indeed, the covering LP using the cut-requirement function was the starting point for both the primal-dual $O(\max_{i \in \ell} k_i)$-approximation for SND of [24] and the seminal 2-approximation for SND using iterative rounding due to Jain [19]. It has also been used for k-ECSS [17], although (unlike SND) there are also purely combinatorial approximations [10].

Hence the natural starting point for us to study RSND and k-EFTS would be to formulate them in terms of cut-requirement functions and try the same approaches as were used in SND and k-ECSS. But this is easier said than done. The functions are a little more complicated, but it is not too hard to construct a cut requirement function that characterizes feasible solutions. However, in order to use the iterative rounding technique of Jain [19] (or any of the weaker techniques which it superceded), the cut requirement function needs to have a structural property known as *weak* (or *skew*) *supermodularity* [19]. This turns out to be crucial, and there are still (to the best of our knowledge) no successful uses of iterative rounding in settings without weak supermodularity. And unfortunately, it turns out that our cut requirement functions *are not* weakly supermodular. So while we can phrase our problems as satisfying a cut requirement function, we cannot actually use iterative rounding, uncrossing, or any other part of the extensive toolkit that has grown around [19].

Our approaches. We get around this difficulty in two ways. For k-EFTS, we define a new property of cut requirement functions which we call *local* weak supermodularity, and prove that our cut requirement function has this property and that it is sufficient for iterative rounding. This is, to the best of our knowledge, *the first* use of iterative rounding without weak supermodularity. For RSND with a single demand, we use an entirely different combinatorial approach based on decomposing the graph into a chain of connected components using *important separators* [22], an important tool from fixed-parameter tractability that, to the best of our knowledge, has not been used before in approximation algorithms.

1.1.1 k-Edge Fault-Tolerant Subgraph

We begin in Section 2 with k-EFTS, where we prove the following two theorems.

▶ **Theorem 3.** *There is a polynomial-time* 2*-approximation for the* k*-EFTS problem.*

▶ **Theorem 4.** *There is a polynomial-time* $(1 + 4/k)$*-approximation for the unweighted* k*-EFTS problem.*

Both of these theorems are consequences of a structural property we prove about the cut-requirement function for k-EFTS: while it is not weakly supermodular, it does have a weaker property which we term *local* weak supermodularity. We define this property formally in Section 2.1.2, but at a high level it boils down to proving that while the inequalities required for weak supermodularity do not hold everywhere (as would be required for weak supermodularity), they hold for *particular* sets (i.e., they hold *locally*) which are the sets where the inequalities are actually applied by Jain's analysis. In other words, we prove that the places in the function where weak supermodularity are violated are precisely the places where we do not care if weak supermodularity holds. After overcoming a few more technical complications (we actually need local weak supermodularity even in the "residual" problem to use iterative rounding), this means that we can apply Jain's algorithm to prove Theorem 3.

To prove Theorem 4, it was observed for unweighted k-ECSS by [17] (with later improvements by [16]) that one of the main pieces of Jain's approach, the fact that the tight constraints can be "uncrossed" to get a laminar family with the same span, implies a $(1 + 4/k)$-approximation via a trivial threshold rounding. They pointed out that the fact that the linearly independent tight constraints form a laminar family implies that there are only $2n$ linearly independent tight constraints, while there are m variables, and hence at any basic feasible solution the remaining $m - 2n$ tight constraints defining the point must be the bounding constraints. These bounding constraints being tight means that the associated variables are in $\{0, 1\}$, and hence there are only $2n$ fractional variables in any basic feasible solution. Rounding all of these variables to 1 increases the cost by $2n$, but since $OPT \geq kn/2$ (since the input graph G must be k-connected) this results in a $(1 + 4/k)$-approximation.

Thanks to our local weak supermodularity characterization the laminar family result is still true even for k-EFTS, so it is still true that there are at most $2n$ nonzero variables at any extreme point. But since we are not guaranteed that G is k-connected we are not guaranteed that $OPT \geq kn/2$, and so this does not imply the desired approximation. Instead, we prove that the number of fractional variables at any basic feasible solution is at most $2n_h$, where n_h is the number of "high-degree" nodes. It is then easy to argue that $OPT \geq n_h k/2$, which gives Theorem 4.

1.1.2 Relative Survivable Network Design With a Single Demand

For k-EFTS, we strongly used the property that all pairs have the same demand. This is not true for RSND, which makes the problem vastly more difficult. We still do not know whether there exists a cut requirement function which characterizes the problem and is locally weakly supermodular. In this paper, we study the simplest case where not all demands are the same: when there is a single nonzero demand (s, t, k), and k is either 2 or 3 (the case of $k = 1$ is simply the shortest-path problem). It turns out to be relatively straightforward to prove a 2-approximation for $k = 2$ even when there are many demands (see Section 3), but the $k = 3$ case is surprisingly difficult. We prove the following theorem in Section 4.

▶ **Theorem 5.** *In any RSND instance with a single demand* $(s, t, 3)$*, there is a polynomial-time* $7 - \frac{1}{4} = \frac{27}{4}$*-approximation.*

To prove this, we start with the observation that if the minimum $s - t$ cut is at least 3 then this is actually just the traditional SND problem (and in fact, the even simpler problem of finding 3 edge-disjoint paths of minimum weight between s and t, which can be solved efficiently via min-cost flow). So the only difficulty is when there are cuts of size 1 or 2. Cuts of size 1 can be dealt with easily (see Section 3), but cuts of size 2 are more difficult. To get rid of them, we construct a "chain" of 2-separators (cuts of size 2 that are also *important separators* [22]). Inside each component of the chain there are no 2-cuts between the incoming separator and the outgoing separator, which allows us to characterize the connectivity requirement of any feasible solution restricted to that component. These connectivity requirements turn out to be quite complex even though we started with only a single demand, as fault sets with different structure can force complicated connectivity requirements in intermediate components. The vast majority of the technical work is proving a structure lemma which characterizes them. With this lemma in hand, though, we can simply approximate the optimal solution in each component.

Interestingly, to the best of our knowledge this is the first use of important separators in approximation algorithms, despite their usefulness in fixed-parameter algorithms [22].

1.2 Related Work

The most directly related work is the 2-approximation of Jain for Survivable Network Design [19], which introduced iterative rounding (see [20] for a detailed treatment of iterative rounding in combinatorial optimization). This built off of an earlier line of work on survivable network design beginning over 50 years ago with [23]. Since the success of Jain's approach for SND, there has been a significant amount of work on vertex-connectivity versions rather than edge-connectivity, which is a significantly more difficult setting. This has culminated in the state of the art approximation of [11]. There is also a long line of work on k-ECSS, most notably including [10, 17].

While not technically related, the basic problems in this paper are heavily inspired by recent work on relative notions of fault-tolerance in graph spanners and other non-optimization network design settings. A relative definition of fault-tolerance for graph spanners which is very similar to ours (but which takes distances into account due to the spanner setting) was introduced by [9], who gave bounds on the size of f-fault-tolerant t-spanners for both edge and vertex notions of fault-tolerance. This spawned a line of work which improved these bounds for both vertex and edge fault-tolerance [14, 4, 7, 15, 5, 6], culminating in [5] for vertex faults and [6] for edge faults. The basic spanner definition also inspired work on relative fault-tolerant versions of related problems, including emulators [3], distance sensitivity oracles for multiple faults [8], and single-source reachability subgraphs [2, 21]. What all of these results shared, though, was that they were not doing optimization: they were looking for existential bounds (and algorithms to achieve them) for these objects. In this paper, by contrast, we take the point of view of optimization and approximation algorithms and compare to the instance-specific optimal solution.

2 k-Edge Fault-Tolerant Subgraph

Both Theorems 3 and 4 depend on the same LP relaxation, which is based on a modification of the "obvious" cut-requirement function. So we begin by discussing this relaxation, and then use it to prove the two main theorems.

2.1 LP Relaxation

2.1.1 Basics

The natural place to start is the LP used by Jain [19], but with a cut requirement function $f(S) = \min(|\delta_G(S)|, k)$. Unfortunately, while this results in a valid LP relaxation, it is not weakly supermodular (see Section 2.1.2 for the definition, and Appendix A for a counterexample). So instead we modify this cut requirement function by removing edges which are "forced". For every subset S of V, let $\delta_G(S)$ be the set of edges with exactly one endpoint in S. Let $F = \{e \in E \mid \exists S \text{ where } e \in \delta_G(S) \text{ and } |\delta_G(S)| \leq k\}$. In other words, F is the set of all edges that are in some cut of size at most k. Clearly we can compute F in polynomial time by simply checking for every edge (u, v) whether the minimum $u - v$ cut in G has size at most k. For every set $S \subset V$ with $S \neq \emptyset$, we define the cut requirement function $f_F(S) = \min(k, |\delta_G(S)|) - |\delta_G(S) \cap F|$. Note that every edge in F must be in any feasible solution, since if any edge is missing then a fault set consisting of the rest of the cut (at most $k-1$ edges) would disconnect the endpoints of the missing edge in the solution but not in G, giving a contradiction. Then $f_F(S)$ is essentially the "remaining requirement" after F has been removed.

Since iterative rounding will add other edges and remove them from the residual problem, we will want to define a similar cut requirement function for supersets: formally, for any $F' \supseteq F$, let $f_{F'}(S) = \min(k, |\delta_G(S)|) - |\delta_G(S) \cap F'|$. For any $F' \supseteq F$, consider the following linear program which we call LP(F'), which has a variable x_e for every edge $e \in E \setminus F'$:

$$\boxed{\begin{array}{lll} \min & \displaystyle\sum_{e \in E \setminus F'} w(e) x_e & \\ \text{s.t.} & \displaystyle\sum_{e \in \delta_G(S) \setminus F'} x_e \geq f_{F'}(S) & \forall S \subseteq V \\ & 0 \leq x_e \leq 1 & \forall e \in E \setminus F' \end{array}} \tag{LP(F')}$$

It is not hard to see that this is a valid LP relaxation (when combined with F'), but we prove this for completeness.

▶ **Lemma 6.** *Let H be a valid k-EFTS and let $F' \supseteq F$. For every edge $e \in E \setminus F'$, let $x_e = 1$ if $e \in H$, and let $x_e = 0$ otherwise. Then x is a feasible integral solution to LP(F').*

Proof. Clearly $0 \leq x_e \leq 1$ for all $e \in E \setminus F'$. Consider some $S \subseteq V$. Since H is a valid k-EFTS, the number of edges in $H \cap \delta_G(S)$ is at least $\min(k, |\delta_G(S)|)$ (or else the edges in $H \cap \delta_G(S)$ would be a fault set of size less than k such that the connected components of H post-faults are different from the connected components of G post-faults). Hence

$$\begin{aligned} \sum_{e \in \delta_G(S) \setminus F'} x_e &= |(H \cap \delta_G(S)) \setminus F'| = |H \cap \delta_G(S)| - |H \cap \delta_G(S) \cap F'| \\ &\geq |H \cap \delta_G(S)| - |\delta_G(S) \cap F'| \geq \min(k, |\delta_G(S)|) - |\delta_G(S) \cap F'| = f_{F'}(S), \end{aligned}$$

as required. ◀

▶ **Lemma 7.** *Let $F' \supseteq F$ and let x be an integral solution to LP(F'). Let $E' = \{e : x_e = 1\}$. Then $H = E' \cup F'$ is a valid k-EFTS.*

Proof. Suppose for contradiction that H is not a valid k-EFTS. Then there are two nodes $u, v \in V$ and a minimal set $A \subseteq E$ with $|A| < k$ so that u, v are not connected in $H \setminus A$ but are connected in $G \setminus A$. Let S be the nodes reachable from u in $G \setminus A$, and so by minimality of A we know that $A = H \cap \delta_G(S)$.

Note that $|\delta_G(S)| > k$, or else all edges of $\delta_G(S)$ would be in F, implying that $E \cap \delta_G(S) = H \cap \delta_G(S) = A$ and so u and v would not be connected in $G \setminus A$. Thus

$$\sum_{e \in \delta_G(S) \setminus F'} x_e = |H \cap \delta_G(S)| - |F' \cap \delta_G(S)| = |A| - |\delta_G(S) \cap F'|$$
$$< \min(k, |\delta_G(S)|) - |\delta_G(S) \cap F'| = f_{F'}(S),$$

which contradicts x being a feasible solution to LP(F'). ◀

These lemmas (together with the fact that every edge in F must be in any valid solution) imply that if we can solve and round this LP while losing some factor α, then we can add F to the rounded solution to get an α-approximation. Hence we are interested in solving and rounding this LP.

We first argue that we can solve the LP using the Ellipsoid algorithm with a separation oracle. Note that unlike k-ECSS, here a violated constraint does not just correspond to a cut with LP values less than k, since our cut-requirement function is more complicated. Indeed, if we compute a global minimum cut (with respect to the LP values) then we may end up with a small cut which is not violated even though there are violated constraints. So we need to argue more carefully that we can find a violated cut when one exists.

▶ **Lemma 8.** *For every $F' \supseteq F$, LP(F') can be solved in polynomial time.*

Proof. We give a separation oracle, which when combined with the Ellipsoid algorithm implies the lemma [18]. Consider some vector x indexed by edges of $E \setminus F'$. Suppose that x is not a feasible LP solution, so we need to find a violated constraint. Obviously if there is some $x_e \notin [0,1]$ then we can find this in linear time. So without loss of generality, we may assume that there is some $S \subseteq V$ such that $\sum_{e \in \delta_G(S) \setminus F'} x_e < f_{F'}(S)$. This implies that $f_{F'}(S) > 0$ and that there is some edge $e^* \in \delta_G(S) \setminus F'$ with $x_{e^*} < 1$ (since otherwise the LP would not be satisfiable, contradicting Lemma 6 and the fact that G itself is a valid k-EFTS). Let $e^* = \{u, v\}$. Since $e^* \notin F'$, and $F \subseteq F'$, we know that e^* cannot be part of any cuts in G of size at most k, and thus the minimum $u - v$ cut in G has more than k edges.

On the other hand, if we extend x to F' by setting $x_e = 1$ for all $e \in F'$, then since S is a violated constraint we have that

$$\sum_{e \in \delta_G(S)} x_e = \sum_{e \in \delta_G(S) \setminus F'} x_e + |F' \cap \delta_G(S)| < f_{F'}(S) + |F \cap \delta_G(S)|$$
$$= \min(k, |\delta_G(S)|) - |\delta_G(S) \cap F'| + |\delta_G(S) \cap F'|$$
$$= k.$$

Thus if we interpret x as edge weights (with $x_e = 1$ for all $e \in F$), if we compute the minimum $s - t$ cut we will find a cut S' with more than k edges (since all $u - v$ cuts have more than k edges) with total edge weight strictly less than k. Let S' be this cut. Thus $\sum_{e \in \delta_G(S') \setminus F'} x_e < k - |\delta_G(S') \setminus F'| = f_{F'}(S')$, so S' is also a violated constraint.

Hence for our separation oracle we simply compute a minimum $s - t$ cut using x as edge weights for all $s, t \in V$, and if any cut we finds corresponds to a violated constraint then we return it. By the above discussion, if there is some violated constraint then this procedure will find some violated constraint. Thus this is a valid separation oracle. ◀

After solving this LP, we apply an obvious transformation used also in [19]: we delete every edge e with $x_e = 0$. This allows us to assume without loss of generality that every edge has LP value $x_e > 0$ in our LP solution.

2.1.2 Local Weak Supermodularity

As discussed in Section 1.1.1, it would be nice if this LP were *weakly supermodular*, as this would immediately let us apply Jain's iterative rounding algorithm to obtain a 2-approximation. Recall the definition of weak supermodularity from [19].

▶ **Definition 9.** *Let $f : 2^V \to \mathbb{Z}$. Then f is* weakly supermodular *if for every $A, B \subseteq V$, either $f(A) + f(B) \le f(A \setminus B) + f(B \setminus A)$, or $f(A) + f(B) \le f(A \cap B) + f(A \cup B)$.*

Unfortunately, our cut requirement function is not weakly supermodular; see Appendix A for a counterexample. But we can make a simple observation that, to the best of our knowledge, has not previously been noticed or utilized in iterative rounding: Jain's iterative rounding algorithm does not actually need the weak supermodularity conditions to hold for *all* pairs of sets A, B. It only needs weak supermodularity to "uncross" the tight sets of an LP solution into a laminar family of tight sets with the same span. Recall that a set is tight in a given LP solution if its corresponding cut constraint is tight, i.e., is satisfied with equality. Moreover, note that in our setting, depending on our choice of F' some cuts might be entirely included in F'. These cuts would not have any edges remaining, resulting in an "empty" constraint in LP(F'). Such a constraint cannot be tight by definition, and also is not linearly independent with any other set of constraints.

Hence in order to use Jain's iterative rounding, we simply need our cut-requirement function $f_{F'}$ to satisfy the weak supermodularity requirements for A, B where there is actually a nontrivial constraint for A, B and where $F' \supseteq F$ (here F' will consist of F together with edges that Jain's iterative rounding algorithm has already set to 1). We formalize this as follows. Given $F' \supseteq F$, we say that S is an *empty cut* if $\delta_G(S) \cap F' = \delta_G(S)$, and otherwise it is *nonempty.*

▶ **Definition 10.** *Given a graph $G = (V, E)$, a set $F' \subseteq E$, and a function $g : 2^V \to \mathbb{Z}$, we say that g is* locally weakly supermodular *with respect to F' if for every $A, B \subseteq V$ with both A and B nonempty cuts, at least one of the following conditions holds:*
- *$g(A) + g(B) \le g(A \setminus B) + g(B \setminus A)$, or*
- *$g(A) + g(B) \le g(A \cap B) + g(A \cup B)$.*

We will now prove that for any $F' \supseteq F$, the function $f_{F'}$ is locally weakly supermodular with respect to any F'. This is the key technical idea enabling Theorems 3 and 4.

We say that S is *large* if $|\delta_G(S)| > k$, and otherwise S is *small.* Note that since $F' \supseteq F$, any small cut is also an empty cut. We first prove a useful lemma.

▶ **Lemma 11.** *Let $F' \supseteq F$. If A and B are nonempty cuts for $f_{F'}$, then either $A \setminus B$ and $B \setminus A$ are nonempty cuts, or $A \cap B$ and $A \cup B$ are nonempty cuts.*

Proof. Let

$$S_1 = \delta_G(A \setminus B, V \setminus (A \cup B)), \quad S_2 = \delta_G(A \setminus B, B \setminus A), \quad S_3 = \delta_G(A \setminus B, A \cap B),$$
$$S_4 = \delta_G(B \setminus A, V \setminus (A \cup B)), \quad S_5 = \delta_G(B \setminus A, A \cap B), \quad S_6 = \delta_G(A \cap B, V \setminus (A \cup B)).$$

Suppose that $A \setminus B$ and $A \cap B$ are both empty cuts. Each edge in $\delta_G(A)$ is in S_1, S_2, S_5, or S_6. Additionally, S_1 and S_2 are subsets of $\delta_G(A \setminus B)$, while S_5 and S_6 are subsets of $\delta_G(A \cap B)$. This means that every edge in $\delta_G(A)$ is in an empty cut, and so all edges in $\delta_G(A)$ are in F'. Thus A is an empty cut, contradicting the assumption of the lemma. Thus at least one of $A \setminus B$ and $A \cap B$ is nonempty. If we instead assume that $B \setminus A$ and $A \cap B$ are empty cuts, then we can use a similar argument to prove that B is an empty cut. This proves that at least one of $B \setminus A$ and $A \cap B$ are nonempty. Hence if $A \cap B$ is empty, then both $A \setminus B$ and $B \setminus A$ are nonempty, proving the lemma.

Now suppose that $A \setminus B$ and $A \cup B$ are both empty cuts. Each edge in $\delta_G(B)$ is in S_2, S_3, S_4, or S_6. Additionally, S_2 and S_3 are subsets of $\delta_G(A \setminus B)$, while S_4 and S_6 are subsets of $\delta_G(A \cup B)$. This means that every edge in $\delta_G(B)$ is in an empty cut, and so all edges in $\delta_G(B)$ are in F'. Thus B is an empty cut, contradicting the assumption of the lemma. Thus at least one of $A \setminus B$ and $A \cup B$ is nonempty. If we instead assume that $B \setminus A$ and $A \cup B$ are empty cuts, then we can use a similar argument to prove that A is empty, and hence at least one of $B \setminus A$ and $A \cup B$ is nonempty. Hence if $A \cup B$ is empty, then both $A \setminus B$ and $B \setminus A$ are nonempty, proving the lemma.

Thus either both $A \setminus B$ and $B \setminus A$ are nonempty, or both $A \cap B$ and $A \cup B$ are nonempty, proving the lemma. ◀

We can now prove the main technical result: $f_{F'}$ is locally weakly supermodular.

▶ **Theorem 12** (Local Weak Supermodularity). *For any $F' \supseteq F$, the cut requirement function $f_{F'}$ is locally weakly supermodular with respect to F'.*

Proof. Let $F' \supseteq F$, and suppose A and B are nonempty cuts. Let

$$S_1 = \delta_G(A \setminus B, V \setminus (A \cup B)), \quad S_2 = \delta_G(A \setminus B, B \setminus A), \quad S_3 = \delta_G(A \setminus B, A \cap B),$$
$$S_4 = \delta_G(B \setminus A, V \setminus (A \cup B)), \quad S_5 = \delta_G(B \setminus A, A \cap B), \quad S_6 = \delta_G(A \cap B, V \setminus (A \cup B)).$$

We also let $s_i = |S_i \cap F'|$ for $i \in [6]$.

A and B are nonempty cuts, so A and B must be large cuts and $\min(k, |\delta_G(A)|) = \min(k, |\delta_G(B)|) = k$. Each edge in $\delta_G(A)$ is in exactly one of S_1, S_2, S_5, and S_6, and each edge in $\delta_G(B)$ is in exactly one of S_2, S_3, S_4, and S_6, so we have that $|\delta_G(A) \cap F'| = s_1 + s_2 + s_5 + s_6$ and $|\delta_G(B) \cap F'| = s_2 + s_3 + s_4 + s_6$. We therefore have the following:

$$\begin{aligned} f_{F'}(A) &= \min(k, |\delta_G(A)|) - |\delta_G(A) \cap F| = k - s_1 - s_2 - s_5 - s_6 \\ f_{F'}(B) &= \min(k, |\delta_G(B)|) - |\delta_G(B) \cap F| = k - s_2 - s_3 - s_4 - s_6 \\ \implies f_{F'}(A) + f_{F'}(B) &= 2k - s_1 - 2s_2 - s_3 - s_4 - s_5 - 2s_6. \end{aligned} \tag{1}$$

A and B are nonempty so by Lemma 11, either $A \setminus B$ and $B \setminus A$ are nonempty cuts, or $A \cap B$ and $A \cup B$ are nonempty cuts. Suppose first that $A \setminus B$ and $B \setminus A$ are nonempty cuts, which implies that $\min(k, |\delta_G(A \setminus B)|) = \min(k, |\delta_G(B \setminus A)|) = k$. Each edge in $\delta_G(A \setminus B)$ is in exactly one of S_1, S_2, and S_3, and each edge in $\delta_G(B \setminus A)$ is in exactly one of S_2, S_4, and S_5, so we have that $|\delta_G(A \setminus B) \cap F'| = s_1 + s_2 + s_3$ and $|\delta_G(B \setminus A) \cap F'| = s_2 + s_4 + s_5$. Putting this all together, we get the following for $f_{F'}(A \setminus B)$ and $f_{F'}(B \setminus A)$:

$$\begin{aligned} f_{F'}(A \setminus B) &= \min(k, |\delta_G(A \setminus B)|) - |\delta_G(A \setminus B) \cap F'| = k - s_1 - s_2 - s_3 \\ f_{F'}(B \setminus A) &= \min(k, |\delta_G(B \setminus A)|) - |\delta_G(B \setminus A) \cap F'| = k - s_2 - s_4 - s_5 \\ \implies f_{F'}(A \setminus B) + f(B \setminus A) &= 2k - s_1 - 2s_2 - s_3 - s_4 - s_5. \end{aligned}$$

This and (1) imply that $f_{F'}(A) + f_{F'}(B) \le f_{F'}(A \setminus B) + f_{F'}(B \setminus A)$ if $A \setminus B$ and $B \setminus A$ are nonempty cuts.

Now suppose that $A \cap B$ and $A \cup B$ are nonempty cuts, and so $\min(k, |\delta_G(A \setminus B)|) = \min(k, |\delta_G(B \setminus A)|) = k$. Each edge in $\delta_G(A \cap B)$ is in exactly one of S_3, S_5, and S_6, and each edge in $\delta_G(A \cup B)$ is in exactly one of S_1, S_4, and S_6, so we have that $|\delta_G(A \cap B) \cap F'| = s_3 + s_5 + s_6$ and $|\delta_G(A \cup B) \cap F'| = s_1 + s_4 + s_6$. Putting this all together, we get the following for $f_{F'}(A \cap B)$ and $f_{F'}(A \cup B)$:

$$\begin{aligned} f_{F'}(A \cap B) &= \min(k, |\delta_G(A \cap B)|) - |\delta_G(A \cap B) \cap F'| = k - s_3 - s_5 - s_6 \\ f_{F'}(A \cup B) &= \min(k, |\delta_G(A \cup B)|) - |\delta_G(A \cup B) \cap F'| = k - s_1 - s_4 - s_6 \\ \implies f_{F'}(A \cap B) + f_{F'}(A \cup B) &= 2k - s_1 - s_3 - s_4 - s_5 - 2s_6. \end{aligned}$$

This and (1) imply that $f_{F'}(A) + f_{F'}(B) \le f_{F'}(A \cap B) + f_{F'}(B \cup A)$ if $A \cap B$ and $A \cup B$ are nonempty cuts. ◀

2.2 Unweighted k-EFTS

To prove Theorem 4 we need to look inside [19]. The following two lemmas from [19] are the main "uncrossing" lemmas which depend on weak supermodularity, and in which we can use local weak supermodularity instead without change. As in [19], for each $S \subseteq V$ we use $\mathcal{A}_G(S)$ to denote the row of the constraint matrix corresponding to S. In other words $\mathcal{A}_G(S)$ is a vector indexed by elements of $E \setminus F$ which has a 1 in the entry for e if $e \in \delta_G(S) \setminus F$, and otherwise has a 0 in that entry.

▶ **Lemma 13** (Lemma 4.1 of [19]). *If two sets A and B are tight then at least one of the following must hold*
1. *$A \setminus B$ and $B \setminus A$ are also tight, and $\mathcal{A}_G(A) + \mathcal{A}_G(B) = \mathcal{A}_G(A \setminus B) + \mathcal{A}_G(B \setminus A)$*
2. *$A \cap B$ and $A \cup B$ are also tight, and $\mathcal{A}_G(A) + \mathcal{A}_G(B) = \mathcal{A}_G(A \cap B) + \mathcal{A}_G(A \cup B)$*

Let $\mathcal{T}$ denote the family of all tight sets. For any family $\mathcal{F}$ of tight sets, let Span($\mathcal{F}$) denote the vector space spanned by $\{\mathcal{A}_G(S) : S \in \mathcal{F}\}$.

▶ **Lemma 14** (Lemma 4.2 of [19]). *For any maximal laminar family $\mathcal{L}$ of tight sets, Span($\mathcal{L}$) = Span($\mathcal{T}$).*

Recall that n_h is the number of high-degree nodes, i.e., nodes of degree at least k in G. Then we have the following lemma, which is a modification of Lemma 4.3 of [19] where we give a stronger bound on the number of sets.

▶ **Lemma 15.** *The dimension of Span($\mathcal{T}$) is at most $2n_h - 1$.*

Proof. Let $\mathcal{L}$ be a maximal laminar family of tight sets. Lemma 14 implies that Span($\mathcal{L}$) = Span($\mathcal{T}$), so it suffices to upper bound the number of sets in $\mathcal{L}$. And since we care about the span, if there are two sets S, S' with $\mathcal{A}_G(S) = \mathcal{A}_G(S')$ then we can remove one of them from $\mathcal{L}$ arbitrarily, so no two sets in $\mathcal{L}$ have identical rows in the constraint matrix.

Any set that consists of exclusively low degree nodes cannot be tight, since the set has no corresponding row in the constraint matrix. Thus, all sets in $\mathcal{L}$ must contain at least one high degree node, and hence all minimal sets in $\mathcal{L}$ have at least one high degree node.

Let $S \in \mathcal{L}$, and let $S' \supset S$ so that every node in $S' \setminus S$ is a low-degree node. Then every edge edge in $(\delta_G(S) \setminus \delta_G(S')) \cup (\delta_G(S') \setminus \delta_G(S))$ must be incident on at least one low-degree node and hence is in F. Thus $\mathcal{A}_G(S) = \mathcal{A}_G(S')$, and hence S' is not in $\mathcal{L}$. Therefore, any superset S' in the laminar family of some other set S in the laminar family must have at least one more high degree node than S.

Since any minimal set in $\mathcal{L}$ has at least one high degree node, and every set in $\mathcal{L}$ contains at least one more high degree node than any set in $\mathcal{L}$ that it contains, if we restrict each set in $\mathcal{L}$ to the high-degree nodes then we have a laminar family on the high-degree nodes. Thus $|\mathcal{L}| \leq 2n_h - 1$. ◀

We can now prove Theorem 4.

Proof of Theorem 4. We first solve LP(F) using Lemma 8 to get some basic feasible solution x. Since there are $|E \setminus F|$ variables, this point is defined by $|E \setminus F|$ linearly independent tight constraints. Lemma 15 implies that at most $2n_h - 1$ of these are from tight sets, and hence all of the other tight constraints must be of the form $x_e = 0$ or $x_e = 1$ for some edge $e \in E \setminus F$. Thus at most $2n_h - 1$ edges are assigned a fractional value in x. Hence if we include all such edges in our solution H, together with all edges with $x_e = 1$ and all edges in

F, we have a solution which is feasible (by Lemma 7). Note that any high-degree node must have degree at least k in any feasible solution, and thus $OPT \geq \frac{k}{2}n_h$. Hence our solution H has size at most

$$|H| \leq \sum_{e \in E \setminus F} x_e + |F| + 2n_h \leq OPT + 2n_h \leq OPT + \frac{4}{k}OPT = \left(1 + \frac{4}{k}\right) OPT. \quad ◀$$

2.3 Weighted k-EFTS

Jain's approximation algorithm solves the initial LP, rounds up and removes any edges with $x_e \geq 1/2$ which results in a residual problem, and repeats. This is obviously a 2-approximation (see [19] for details), but requires proving that there is always at least one edge with $x_e \geq 1/2$ so we can make progress (even in the residual problems). This is accomplished by proving Lemmas 13 and 14 to show that the tight constraints can be "uncrossed" into a laminar family. This requires weak supermodularity, but as discussed, since in our LP every tight constraint must be a nonempty constraint, it is sufficient to replace this with local weak supermodularity. Jain then uses a complex counting argument based on this laminar family of tight constraints to prove that some edge e must have $x_e \geq 1/2$. Importantly, nothing in this counting argument depends on the cut requirement having any particular structure (e.g., weak supermodularity); it depends only on the fact that the family of tight constraints can be uncrossed to be laminar.

Since local weak supermodularity is sufficient to uncross the tight constraints into a laminar family, we can simply apply Jain's counting argument on this family for LP(F') to obtain the following lemma (as in Theorem 3.1 of [19]).

▶ **Lemma 16.** *For all $F' \supseteq F$, in any basic feasible solution x of LP(F') there is at least one $e \in E \setminus F'$ with $x_e \geq 1/2$.*

Hence we have the following iterative rounding algorithm for weighted k-EFTS:

- Let $F' = F$
- While F' is not a feasible solution:
  - Let x be a basic feasible solution for LP(F') (obtained in polynomial time using Lemma 8)
  - Let $E_{1/2} = \{e \in E \setminus F' : x_e \geq 1/2\}$, which must be nonempty by Lemma 16
  - Add $E_{1/2}$ to F'

This clearly returns a feasible solution, and the analysis of [19] (particularly Theorem 3.2) implies that this is a 2-approximation, which implies Theorem 3.

3 2-Connectivity and $k = 2$

We will now move on from k-EFTS to the more general RSND problem. It turns out to be relatively straightforward to handle cuts of size 1: removing such cuts gives a tree of 2-connected components, and we can essentially run an algorithm independently inside each component. This gives the following theorem, the proof of which can be found in the full version [13].

▶ **Theorem 17.** *If there exists an α-approximation algorithm for RSND on 2-edge connected graphs, then there is an α-approximation algorithm for RSND on general graphs.*

Extending this slightly gives the following theorem (proof in the full version [13]), where 2-RSND denotes the special case of the RSND problem where $k_i \leq 2$ for all i.

▶ **Theorem 18.** *There is a 2-approximation algorithm for 2-RSND.*

4 RSND with a Single Demand: $k = 3$

In this section we prove Theorem 5. In the Single Demand RSND problem, we are given a graph $G = (V, E)$ (possibly with edge weights $w : E \to \mathbb{R}^+$) and a k-relative fault tolerance demand for a single vertex pair (s, t). In other words, the set of connectivity demands is just $\{(s, t, k)\}$. We give a $7 - \frac{1}{4} = \frac{27}{4}$-approximation algorithm for the $k = 3$ Single Demand RSND problem. The main idea is to partition the input graph using important separators, prove a structure lemma which characterizes the required connectivity guarantees within each component of the partition, and then achieve these guarantees using a variety of subroutines: a min-cost flow algorithm, a 2-RSND approximation algorithm (Theorem 18), and a Steiner Forest approximation algorithm [1].

4.1 Decomposition

By Theorem 17, an α-approximation algorithm for RSND on 2-connected graphs implies an α-approximation algorithm for RSND on general graphs. Hence going forward, we will assume the input graph G is 2-connected. In this section we define important separators and describe how to construct what we call the $s - t$ *2-chain* of G.

▶ **Definition 19.** *Let X and Y be vertex sets of a graph G. An (X, Y)-separator of G is a set of edges S such that there is no path between any vertex $x \in X$ and any vertex $y \in Y$ in $G \setminus S$. An (X, Y)-separator S is minimal if no subset $S' \subset S$ is also an (X, Y)-separator. If $X = \{x\}$ and $Y = \{y\}$, we say that S is an (x, y)-separator.*

The next definition, which is a slight modification of the definition due to [22], is a formalization of a notion of a "closest" separator.

▶ **Definition 20.** *Let S be an (X, Y)-separator of graph G, and let R be the vertices reachable from X in $G \setminus S$. Then S is an important (X, Y)-separator if S is minimal and there is no (X, Y)-separator S' such that $|S'| \leq |S|$ and $R' \subset R$, where R' is the set of vertices reachable from X in $G \setminus S'$.*

This definition corresponds to a "closest" separator, while the original definition of [22] correspond to a "farthest" separator. Important separators have been studied extensively due to their usefulness in fixed-parameter tractable algorithms, and so much is known about them. For our purposes, we will only need the following lemma, which follows directly from Theorem 2 of [22].

▶ **Lemma 21.** *Let $X, Y \subseteq V$ be two sets of vertices in graph $G = (V, E)$, and let $d \geq 0$. An important (X, Y)-separator of size d can be found in time $4^d \cdot n^{O(1)}$ (if one exists), where $n = |V|$.*

By Lemma 21, we can find an important (X, Y)-separator of size 2 in polynomial time. We now describe how to use this to construct what we call the $s - t$ *2-chain* of G. First, if there are no important (s, t)-separators of size 2 in G, then every (s, t)-separator has size at least 3. Hence we can just use the 2-approximation for Survivable Network Design [19] with demand $(s, t, 3)$ to solve the problem (or can exactly solve it by finding the cheapest three pairwise disjoint $s - t$ paths in polynomial time using a min-cost flow algorithm).

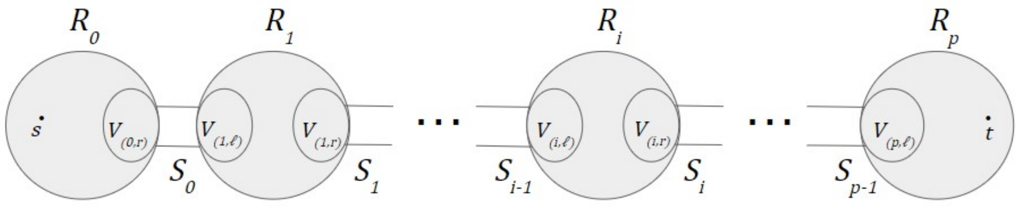

Figure 1 The $s - t$ 2-chain of G.

If such an important separator exists, then we first find an important (s, t)-separator S_0 of size 2 in G, and let R_0 be the set of vertices reachable from s in $G \setminus S_0$. We let $V_{(0,r)}$ be the nodes in R_0 incident on S_0, and let $V_{(1,\ell)}$ be the nodes in $V \setminus R_0$ incident on S_0. We then proceed inductively. Given $V_{(i,\ell)}$, if there is no important $(V_{(i,\ell)}, t)$ separator of size 2 in $G \setminus (\cup_{j=0}^{i-1} R_j)$ then the chain is finished. Otherwise, let S_i be such a separator, let R_i be the nodes reachable from $V_{(i,\ell)}$ in $(G \setminus (\cup_{j=0}^{i-1} R_j)) \setminus S_i$, let $V_{(i,r)}$ be the nodes in R_i incident on S_i, and let $V_{(i+1,\ell)}$ be the nodes in $V \setminus (\cup_{j=0}^{i} R_j)$ incident on S_i.

After this process completes we have our $s-t$ 2-chain, consisting of components $R_0, \dots, R_p$ along with important separators $S_0, \dots, S_{p-1}$ between the components. See Figure 1.

We can now use this chain construction to give a structure lemma which characterizes feasible solutions. Informally, the lemma states that a subgraph H of G is a feasible solution if and only if in the $s - t$ 2-chain of G, all edges between components are in H, and in every component R_i certain connectivity requirements between $V_{(i,\ell)}$ and $V_{(i,r)}$ are met.

Let $G = (V, E)$ be a graph, and let H be a subgraph of G. Going forward, we will say that in H, a vertex set $A \subset V$ has a path to (or is reachable from) another vertex set $B \subset V$ if there is a path from a vertex $a \in A$ to a vertex $b \in B$ in H. Additionally, let X and Y be vertex sets. We also say that H satisfies the RSND demand (X, Y, k) on input graph G if the following is true: for every $F \subseteq E$ with $|F| < k$, if there is a path from at least one vertex in X to at least one vertex in Y in $G \setminus F$ then there is a path from at least one vertex in X to at least one vertex in Y in $H \setminus F$. The demand (X, Y, k) on input G is equivalent to contracting all nodes in X to create super node v_X, contracting all nodes in Y to create super node v_Y, and including demand (v_X, v_Y, k). We will also let $G[R_i]$ and $H[R_i]$ be the subgraphs of G and H, respectively, induced by the component R_i.

▶ **Lemma 22** (Structure Lemma). *Let G be the input graph, and let H be a subgraph of G. Additionally, let $R_0, \dots, R_p$ denote the components in the $s - t$ 2-chain of G, and let $S_0, \dots, S_{p-1}$ denote the edge sets between components in the chain, as defined previously. Let $G_i = G[R_i]$, and $H_i = H[R_i]$. Then H is a feasible solution to the $k = 3$ Single Demand RSND problem if and only if all edges in $S_0, \dots, S_{p-1}$ are included in H, and H_i has the following properties for every i:*

1. *There are at least 3 edge-disjoint paths from $V_{(i,\ell)}$ to $V_{(i,r)}$.*
2. *H_i is a feasible solution to RSND on input graph G_i with demands*
$$\{(V_{(i,\ell)}, v_r, 2) : v_r \in V_{(i,r)}\} \cup \{(V_{(i,r)}, v_\ell, 2) : v_\ell \in V_{(i,\ell)}\}.$$
3. *H_i is a feasible solution to RSND on input graph G_i with demands*
$$\{(u, v, 1) : (u, v) \in V_{(i,\ell)} \times V_{(i,r)}\}.$$

The proof of this structure lemma is a highly technical case analysis, which due to space constraints can be found in the full version [13]. At a very high level, though, our proof is as follows. For the "only if" direction, we first assume that we are given some feasible solution H. Then for each of the properties in Lemma 22, we assume it is false and derive a contradiction by finding a fault set $F \subseteq E$ with $|F| \leq 2$ where there is a path from s to t in $G \setminus F$, but not in $H \setminus F$. The exact construction of such an F depends on which of the properties of Lemma 22 we are analyzing.

For the more complicated "if" direction, we assume that H satisfies the conditions of Lemma 22 and consider a fault set $F \subseteq E$ with $|F| \leq 2$ where s and t are connected in $G \setminus F$. We want to show that s and t are connected in $H \setminus F$. We analyze two subchains of the $s - t$ 2-chain of G: the minimal prefix of the chain which contains at least 1 fault, and the minimal prefix of the chain which contains both faults. We first show that the set of vertices reachable from s at the end of the first subchain is the same in $G \setminus F$ and in $H \setminus F$. We then use this to show that there is at least one reachable vertex at the end of the second subchain in $H \setminus F$, even though (unlike the first subchain) the set of reachable vertices at the end of the second subchain may be smaller in $H \setminus F$ than in $G \setminus F$. From there we show that there is a path to t in $H \setminus F$ from this one reachable vertex. There are a large number of cases depending on the structure of F (whether it intersects some of the separators in the chain, whether both faults are in the same component, etc.), and we have to use different properties of Lemma 22 in different cases, making this proof technically involved.

4.2 Algorithm and Analysis

We can now use Lemma 22 to give a $7 - \frac{1}{4} = \frac{27}{4}$-approximation algorithm for the $k = 3$ setting of Single Demand RSND on 2-connected graphs which, by Theorem 17, gives a $\frac{27}{4}$-approximation algorithm for the $k = 3$ Single Demand RSND problem on general graphs.

Our algorithm uses a variety of subroutines, including an algorithm for min-cost flow, the 2-RSND approximation algorithm of Theorem 18, and a Steiner Forest approximation algorithm. For reference, we state the latter of these.

▶ **Lemma 23** ([1]). *There is a $\left(2 - \frac{1}{k}\right)$-approximation algorithm for the Steiner Forest problem, where k is the number of terminal pairs in the input.*

We can now give our algorithm. Given a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{\geq 0}$ and demand $\{(s, t, 3)\}$, we first create the $s - t$ 2-chain of G in polynomial time, as described in Section 4.1. After building the chain, within each component we run a set of algorithms to satisfy the demands characterized by Lemma 22: a combination of min-cost flow, 2-RSND, and Steiner Forest algorithms. We include the outputs of these algorithms in our solution H, together with all edges in the separators $S = S_1 \cup S_2 \cup \cdots \cup S_{p-1}$.

We first create an instance of min-cost flow on $G[R_i]$ (in polynomial time). Contract the vertices in $V_{(i,\ell)}$ and contract the vertices in $V_{(i,r)}$ to create super nodes v_ℓ and v_r, respectively. Let v_ℓ be the source node and v_r be the sink node. For each edge $e \in E(R_i)$ set the capacity of e to 1 and set the cost of e to $w(e)$. Require a minimum flow of 3, and run a polynomial-time min-cost flow algorithm on this instance [12]. Since all capacities are integers the algorithm will return an integral flow, so we add to H all edges with non-zero flow.

We then create our first instance of 2-RSND on $G[R_i]$. Contract the vertices in $V_{(i,\ell)}$ to create super node v_ℓ, and set demands $\{(v_\ell, u, 2) : u \in V_{(i,r)}\}$. For our second instance of 2-RSND on R_i, contract $V_{(i,r)}$ to create super node v_r, and set demands $\{(u, v_r, 2) : u \in V_{(i,\ell)}\}$. We run the 2-RSND algorithm (Theorem 18) on each of these instances and include all selected edges in H.

Finally, we create an instance of the Steiner Forest problem on $G[R_i]$. For each vertex pair $(v_\ell, v_r) \in V_{(i,\ell)} \times V_{(i,r)}$, we check in polynomial time if v_ℓ and v_r are connected in $G[R_i]$. If they are connected, then we include (v_ℓ, v_r) as a terminal pair in the Steiner Forest instance. Additionally, for $e \in E(R_i)$, we set the cost of e to $w(e)$. We run the Steiner Forest approximation algorithm (Lemma 23) on this instance, and add all selected edges to H.

The following lemma is essentially directly from Lemma 22 (the structure lemma) and the description of our algorithm.

▶ **Lemma 24.** *H is a feasible solution.*

Proof. For each i, let H_i denote the subgraph of H induced by R_i and let G_i denote the subgraph of G induced by R_i. We will show that H satisfies the conditions of Lemma 22, and hence is feasible. By construction, H contains all edges S in the important separators.

To show property 1 of Lemma 22, recall that in each H_i we included the edges selected via a min-cost flow algorithm from $V_{(i,\ell)}$ to $V_{(i,r)}$ with flow 3. Since there are at least three edge-disjoint paths from $V_{(i,\ell)}$ to $V_{(i,r)}$ in G_i (by Lemma 22 since G itself is feasible), this will return three edge-disjoint paths from $V_{(i,\ell)}$ to $V_{(i,r)}$. Hence H satisfies the first property.

Property 2 of Lemma 22 is direct from the algorithm, since H_i includes the output of the 2-RSND algorithm from Theorem 18 when run on demands $\{(V_{(i,\ell)}, v_r, 2) : v_r \in V_{(i,r)}\} \cup \{(V_{(i,r)}, v_\ell, 2) : v_\ell \in V_{(i,\ell)}\}$. Similarly, within each component H_i in the $s-t$ 2-chain, the edges selected by the Steiner Forest algorithm form a path from vertex $v_\ell \in V_{(i,\ell)}$ to vertex $v_r \in V_{(i,r)}$ if v_ℓ and v_r are connected in G. This satisfies Property 3 in Lemma 22. ◀

Let H^* denote the optimal solution, and for any set of edges $A \subseteq E$, let $w(A) = \sum_{e \in A} w(e)$. The next lemma follows from combining the approximation ratios of each of the subroutines used in our algorithm.

▶ **Lemma 25.** $w(H) \leq \frac{27}{4} \cdot w(H^*)$

Proof. Let $H_i = H[R_i]$ be the subgraph of H induced by R_i, and let $H_i^* = H^*[R_i]$ be the subgraph of the optimal solution induced by R_i. We also let H_i^M denote the subgraph of H_i returned by the min-cost flow algorithm run on R_i (i.e., the set of edges with non-zero flow), let $H_i^{N^1}$ and $H_i^{N^2}$ denote the subgraphs returned by the first and second 2-approximation 2-RSND algorithms run on R_i, respectively, and we let H_i^F denote the subgraph of H_i returned by the Steiner Forest algorithm on R_i. We also let M_i^* be the optimal solution to the Minimum-Cost Flow instance on R_i, let $N_i^{1^*}$ and $N_i^{2^*}$ be the optimal solutions to the first and second 2-RSND instances on R_i, respectively, and let F_i^* be the optimal solution to the Steiner Forest instance on R_i. Subgraph H_i^M is given by an exact algorithm, subgraphs $H_i^{N^1}$ and $H_i^{N^2}$ are given by a 2-approximation algorithm, and subgraph H_i^F is given by a $\left(2 - \frac{1}{k}\right)$-approximation algorithm. Note that there are at most 4 terminal pairs in the Steiner Forest instance, so $k \leq 4$ and the algorithm gives a $\frac{7}{4}$-approximation. Hence we have the following for each component R_i:

$$w(H_i^M) = w(M_i^*), \quad w(H_i^{N^1}) \leq 2w(N_i^{1^*}), \quad w(H_i^{N^2}) \leq 2w(N_i^{2^*}), \quad w(H_i^F) \leq \frac{7}{4} w(F_i^*).$$

Summing over all components in the chain, we get the following:

$$\sum_{i=0}^{p} w(H_i^M) = \sum_{i=0}^{p} w(M_i^*), \qquad \sum_{i=0}^{p} w(H_i^{N^1}) \leq 2 \cdot \sum_{i=0}^{p} w(N_i^{1^*}),$$

$$\sum_{i=0}^{p} w(H_i^{N^2}) \leq 2 \cdot \sum_{i=0}^{p} w(N_i^{2^*}), \qquad \sum_{i=0}^{p} w(H_i^F) \leq \frac{7}{4} \cdot \sum_{i=0}^{p} w(F_i^*).$$

We also have that

$$w(H_i) \leq w(H_i^M) + w(H_i^{N^1}) + w(H_i^{N^2}) + w(H_i^F).$$

Summing over all components in the chain and then substituting the above, we get the following:

$$\begin{aligned}\sum_{i=0}^{p} w(H_i) &\leq \sum_{i=0}^{p} w(H_i^M) + \sum_{i=0}^{p} w(H_i^{N^1}) + \sum_{i=0}^{p} w(H_i^{N^2}) + \sum_{i=0}^{p} w(H_i^F) \\ &\leq \sum_{i=0}^{p} w(M_i^*) + 2 \cdot \sum_{i=0}^{p} w(N_i^{1^*}) + 2 \cdot \sum_{i=0}^{p} w(N_i^{2^*}) + \frac{7}{4} \cdot \sum_{i=0}^{p} w(F_i^*).\end{aligned}$$

The optimal subgraph H^* is a feasible solution, so by Lemma 22, each property in the lemma statement must be met on subgraph H_i^* for all i. For all properties in the lemma to be satisfied on H_i^*, the set of edges $E(H_i^*)$ must be a feasible solution to each of the Minimum-Cost Flow, 2-RSND, and Steiner Forest instances on R_i. Therefore, the cost of H_i^* must be at least the cost of the optimal solution to each of the Minimum-Cost Flow, 2-RSND, and Steiner Forest instances. We therefore have the following:

$$\sum_{i=0}^{p} w(H_i) \leq \sum_{i=0}^{p} w(H_i^*) + 2\sum_{i=0}^{p} w(H_i^*) + 2\sum_{i=0}^{p} w(H_i^*) + \frac{7}{4}\sum_{i=0}^{p} w(H_i^*) \leq \frac{27}{4}\sum_{i=0}^{p} w(H_i^*).$$

Finally, we must account for the edges between components in the $s - t$ 2-chain. Let S be the set of edges between components in the chain that are included in the algorithm solution, and let S^* be the set of edges between components included in the optimal solution. By Lemma 22, any feasible solution must include all edges between the components of the chain. We therefore have that $S = S^*$ and we get the following:

$$\begin{aligned}w(H) = \sum_{i=0}^{p} w(H_i) + w(S) \leq \frac{27}{4}\sum_{i=0}^{p} w(H_i^*) + w(S) \leq \frac{27}{4}\left(\sum_{i=0}^{p} w(H_i^*) + w(S^*)\right) \\ \leq \frac{27}{4} w(H^*).\end{aligned}$$ ◀

Theorem 5 is directly implied by Lemmas 24 and 24 together with the obvious observation that our algorithm runs in polynomial time.

References

1 Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. doi:10.1137/S0097539792236237.

2 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: Generic and optimal. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 509–518, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897648.

3 Greg Bodwin, Michael Dinitz, and Yasamin Nazari. Vertex fault-tolerant emulators. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215 of *LIPIcs*, pages 25:1–25:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.25.

4 Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1884–1900. SIAM, 2018.

5 Greg Bodwin, Michael Dinitz, and Caleb Robelle. Optimal vertex fault-tolerant spanners in polynomial time. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, 2021.

6 Greg Bodwin, Michael Dinitz, and Caleb Robelle. Optimal vertex fault-tolerant spanners in polynomial time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 2924–2938. SIAM, 2022. doi:10.1137/1.9781611976465.174.

7 Greg Bodwin and Shyamal Patel. A trivial yet optimal solution to vertex fault tolerant spanners. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, pages 541–543, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331588.

8 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. In Mark de Berg and Ulrich Meyer, editors, *Algorithms – ESA 2010*, pages 84–96, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

9 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault tolerant spanners for general graphs. *SIAM J. Comput.*, 39(7):3403–3423, 2010.

10 Joseph Cheriyan and Ramakrishna Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM Journal on Computing*, 30(2):528–560, 2000. doi:10.1137/S009753979833920X.

11 Julia Chuzhoy and Sanjeev Khanna. An $o(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. *Theory Comput.*, 8(1):401–413, 2012. doi:10.4086/toc.2012.v008a018.

12 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

13 Michael Dinitz, Ama Koranteng, and Guy Kortsarz. Relative survivable network design, 2022. doi:10.48550/ARXIV.2206.12245.

14 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 169–178, 2011.

15 Michael Dinitz and Caleb Robelle. Efficient and simple algorithms for fault-tolerant spanners. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing*, pages 493–500. ACM, 2020. doi:10.1145/3382734.3405735.

16 Harold N. Gabow and Suzanne R. Gallagher. Iterated rounding algorithms for the smallest k-edge connected spanning subgraph. *SIAM Journal on Computing*, 41(1):61–103, 2012. doi:10.1137/080732572.

17 Harold N. Gabow, Michel X. Goemans, Éva Tardos, and David P. Williamson. Approximating the smallest k-edge connected spanning subgraph by lp-rounding. *Networks*, 53(4):345–357, 2009.

18 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

19 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/s004930170004.

20 Lap-Chi Lau, R. Ravi, and Mohit Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, USA, 1st edition, 2011.

21 Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. A brief note on single source fault tolerant reachability, 2019. doi:10.48550/ARXIV.1904.08150.

22 Dániel Marx. Important separators and parameterized algorithms. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 5–10. Springer, 2011.

23 K. Steiglitz, P. Weiner, and D. Kleitman. The design of minimum-cost survivable networks. *IEEE Transactions on Circuit Theory*, 16(4):455–460, 1969. doi:10.1109/TCT.1969.1083004.

24 David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. *Combinatorica*, 15(3):435–454, 1995. doi:10.1007/BF01299747.

A Counterexamples from Section 2

We show some counterexample to obvious approaches to k-EFTS; in particular, we show that our cut requirement function f_F is not weakly supermodular, and the most obvious cut requirement function $f(S) = \min(k, |\delta_G(S)|)$ is also not weakly supermodular.

Recall that $\delta_G(S)$ denotes the edges in G with exactly one endpoint in S. We extend this notation for disjoint sets A, B by letting $\delta_G(A, B)$ denote the edges with one endpoint in A and one endpoint in B.

▶ **Theorem 26.** *The function f_F is not weakly supermodular.*

Proof. Consider the following example. Set $k = 100$. We create a graph $G = (V, E)$ which has two sets $A, B \subseteq V$ with the following properties.

$$|\delta_G(A \setminus B, V \setminus (A \cup B))| = 49 \qquad |\delta_G(B \setminus A, V \setminus (A \cup B))| = 105$$
$$|\delta_G(A \cap B, V \setminus (A \cup B))| = 3 \qquad |\delta_G(A \setminus B, B \setminus A)| = 0$$
$$|\delta_G(A \setminus B, A \cap B)| = 2 \qquad |\delta_G(B \setminus A, A \cap B)| = 49$$

Anything not specified is extremely dense and well-connected, so an edge is in F if and only if it is part of a small cut made up of the above sets. It is not hard to see that the small cuts are precisely $A \setminus B$ (since $|\delta_G(A \setminus B)| = 49 + 0 + 2 = 51 < 100$) and $A \cap B$ (since $|\delta_G(A \cap B) = 3 + 2 + 49 = 54 < 100$). All other cuts are large. Hence F consists of all edges involving A or B other than $\delta_G(B \setminus A, V \setminus (A \cup B))$, or more specifically,

$$\begin{aligned} F =& \delta_G(A \setminus B, V \setminus (A \cup B)) \cup \delta_G(A \cap B, V \setminus (A \cup B)) \\ & \cup \delta_G(A \setminus B, A \cap B) \cup \delta_G(B \setminus A, A \cap B). \end{aligned}$$

We can now calculate f_F on the subsets we care about:

$$f_F(A) = 100 - 49 - 3 - 49 = -1$$
$$f_F(B) = 100 - 3 - 2 = 95$$
$$f_F(A \setminus B) = 0 \qquad (A \setminus B \text{ is small})$$
$$f_F(B \setminus A) = 100 - 49 = 51$$
$$f_F(A \cap B) = 0 \qquad (A \cap B \text{ is small})$$
$$f_F(A \cup B) = 100 - 49 - 3 = 48$$

Thus

$$f_F(A) + f_F(B) = 94 \qquad f_F(A \setminus B) + f_F(B \setminus A) = 51 \qquad f_F(A \cup B) + f_F(A \cap B) = 48$$

Hence f_F is not weakly supermodular. ◀

Note that the above example is not a contradiction of f being *locally* weakly supermodular since A is an empty cut.

▶ **Theorem 27.** *The function $f = \min(k, |\delta_G(S)|)$ is not weakly supermodular.*

Proof. Consider the following example. Set $k = 100$. We create a graph $G = (V, E)$ which has two sets $A, B \subseteq V$ with the following properties. All of $A \setminus B$ and $B \setminus A$ and $A \cap B$ and $V \setminus (A \cup B)$ are extremely large and dense (e.g., large cliques). There are no edges between $A \setminus B$, $B \setminus A$, or $A \cap B$. The other cut sizes are:

$$|\delta_G(A \cap B, V \setminus (A \cup B))| = 55$$
$$|\delta_G(A \setminus B, V \setminus (A \cup B))| = 95$$
$$|\delta_G(B \setminus A), V \setminus (A \cup B))| = 95$$

Then it is easy to see that

$$\begin{aligned} f(A) &= 100 & f(b) &= 100 \\ f(A \setminus B) &= 95 & f(B \setminus A) &= 95 \\ f(A \cup B) &= 100 & f(A \cap B) &= 55 \end{aligned}$$

Hence f is not weakly supermodular. ◀