# Streaming Algorithms for Connectivity Augmentation

**Ce Jin**[1]
MIT, Cambridge, MA, USA

**Michael Kapralov**
EPFL, Lausanne, Switzerland

**Sepideh Mahabadi**
Microsoft Research–Redmond, WA, USA

**Ali Vakilian**
Toyota Technological Institute at Chicago (TTIC), IL, USA

## Abstract

We study the $k$-connectivity augmentation problem ($k$-CAP) in the single-pass streaming model. Given a $(k-1)$-edge connected graph $G = (V, E)$ that is stored in memory, and a stream of weighted edges (also called links) $L$ with weights in $\{0, 1, \ldots, W\}$, the goal is to choose a minimum weight subset $L' \subseteq L$ of the links such that $G' = (V, E \cup L')$ is $k$-edge connected. We give a $(2+\epsilon)$-approximation algorithm for this problem which requires to store $O(\epsilon^{-1} n \log n)$ words. Moreover, we show the tightness of our result: Any algorithm with better than 2-approximation for the problem requires $\Omega(n^2)$ bits of space even when $k = 2$. This establishes a gap between the optimal approximation factor one can obtain in the streaming vs the offline setting for $k$-CAP.

We further consider a natural generalization to the fully streaming model where both $E$ and $L$ arrive in the stream in an arbitrary order. We show that this problem has a space lower bound that matches the best possible size of a spanner of the same approximation ratio. Following this, we give improved results for spanners on weighted graphs: We show a streaming algorithm that finds a $(2t-1+\epsilon)$-approximate weighted spanner of size at most $O(\epsilon^{-1} n^{1+1/t} \log n)$ for integer $t$, whereas the best prior streaming algorithm for spanner on weighted graphs had size depending on $\log W$. We believe that this result is of independent interest. Using our spanner result, we provide an optimal $O(t)$-approximation for $k$-CAP in the fully streaming model with $O(nk + n^{1+1/t})$ words of space.

Finally we apply our results to network design problems such as Steiner tree augmentation problem (STAP), $k$-edge connected spanning subgraph ($k$-ECSS) and the general Survivable Network Design problem (SNDP). In particular, we show a single-pass $O(t \log k)$-approximation for SNDP using $O(kn^{1+1/t})$ words of space, where $k$ is the maximum connectivity requirement.

**2012 ACM Subject Classification** Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Graph algorithms analysis

**Keywords and phrases** streaming algorithms, connectivity augmentation

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2024.93

**Category** Track A: Algorithms, Complexity and Games

**Related Version** *Full Version*: https://arxiv.org/abs/2402.10806

## 1 Introduction

In the (weighted) *$k$-connectivity augmentation* problem ($k$-CAP), given a $(k-1)$-edge-connected $n$-vertex graph $G = (V, E)$ (possibly with parallel edges) together with a set of weighted candidate edges (also called *links*) denoted by $L \subseteq \binom{V}{2}$ and their weights

[1] Work done during an internship at Microsoft Research, Redmond.

$w\colon L \to \{0, 1, \ldots, W\}$, the goal is to find a minimum weight subset $S \subseteq L$ of the links such that $(V, E \cup S)$ is $k$-edge-connected. Augmenting connectivity is a crucial task for enhancing network reliability which can be used for strengthening the resilience of a network and ensuring uninterrupted access for all users. $k$-CAP is among the most elementary questions in Network Design, which is an important area of discrete optimization. The iterative rounding method of [29] provides a 2-approximation for a more general problem of *survivable network design* problem (SNDP). Untill very recently, nothing better than 2 approximation was known even for weighted *tree augmentation problem* (TAP). In a recent development, weighted $k$-CAP has witnessed breakthroughs with approximation factors below 2 [48, 49, 50]. The state-of-the-art for weighted $k$-CAP is $1.5 + \epsilon$ approximation.

In this work, we consider weighted $k$-CAP in the streaming model, which is one of the most common models for processing real-time and large-scale data. A graph streaming algorithm operates by processing a sequence of graph edges presented in any order (or in some applications in random order), reading them one by one. The primary objective is to design algorithms that can process the entire edge sequence and output an approximately efficient solution, making just one (or a few passes), while utilizing limited memory resources. Ideally, the space usage of the algorithm should be significantly smaller than the size of the $n$-vertex input graph (with possibly $O(n^2)$ edges), preferably $O(n \cdot \text{polylog}(n))$ memory, which is referred to as the *semi-streaming* model [18].

While graph problems such as minimum spanning tree [2, 47, 42], matching [39, 25, 5, 4, 30], spanners, sparsifiers and shortest paths [19, 7, 15, 2, 33, 28, 20, 21] have received significant attention in the streaming model, the connectivity augmentation problem, has received comparatively very limited study in this context. Prior to our result, only testing $k$-connectivity in streaming was studied [52, 13, 47], which showed that testing $k$-edge-connectivity in streaming requires $\tilde{\Theta}(nk)$ space in one pass, and $\tilde{\Theta}(n)$ space in two passes [46, 3]. See Appendix C for more discussion on related work.

## 1.1 Our Computational Models

In this work, we study graph augmentation problems in the streaming model of computation. The input to the $k$-CAP problem consists of two pieces of information, namely the $(k-1)$-connected network $G$ and the set of links that can be used to augment connectivity.

**Link arrival streaming.** In the link arrival streaming model the graph $G$ is presented to the algorithm first, and the cost of storing it does not count towards the space complexity of the algorithm. This is akin to the oracle model that is routinely used to study submodular function maximization in the streaming model (e.g., in [6, 43]): One thinks of having an oracle for the function being maximized. For submodular function maximization it is not always clear how to implement this oracle in small space, but in our case the actual cost of storing a sufficient representation of the graph $G$ can be easily made $O(nk)$, and, with some work, even $O(n)$, as we now explain.

Note that a minimally $k$-connected graph has size $O(nk)$. So if the graph has larger size, one can process the edges of $G$ (even in a streaming fashion) using a $k$-connectivity certificate of $G$ that preserves all cuts of value at most $k$, and store this compact representation in $O(nk)$ space. Finally, one can apply even a more efficient preprocessing that preserves a similar information via a *cactus* graph with $O(n)$ edges. Then the problem becomes streaming *cactus augmentation*. The cactus augmentation problem itself is a well-studied problem in particular for designing approximation algorithms for $k$-CAP. To simplify the notation, throughout the paper, we assume the latter compact representation of size $O(n)$.

**Fully streaming.** Besides the most natural link arrival model defined above, we study the more general model where the edges of $G$ and the links that can be used for augmentation may arrive in an interleaved fashion. This model is quite general: in particular, it allows for the edges of $G$ to arrive *after* the links, in which case the algorithm must maintain a compressed representation of the stream of links that allows augmenting any given graph $G$ presented later!

For the other graph problems studied in this paper, namely spanner, SNDP and $k$-edge connected spanning subgraph ($k$-ECSS), we consider the standard edge arrival streams in which edges of the input graph arrives one by one in an arbitrary order stream.

## 1.2 Our Results

In this paper, we focus on insertion-only streams, and provide the first streaming algorithms for $k$-CAP in link arrival streams and fully streaming. Table 1 summarizes our results.

**Graph augmentation in link arrival.** We show tight results for weighted $k$-CAP in link arrival streams (see first row in Table 1). Note that, while we can achieve a factor $2+\epsilon$ approximation in $O(\frac{n}{\epsilon}\log n)$ words of space, our lower bound shows that getting better than 2 approximation requires $\Omega(n^2)$ bits of memory. This establishes a gap between the streaming setting and the offline setting where strictly better than 2 approximation algorithms are known (e.g., see [50]). An easy argument shows that $\Omega(n)$ bits of space is necessary for achieving any approximation for $k$-CAP in link arrival streams (Proposition 2.12 in the full versoin), so our algorithm has nearly-tight space complexity. If one picks a $k$-connectivity certificate as the compact representation of $G$, the space complexity of the upper bound becomes $O(nk+\frac{n}{\epsilon}\log n)$.

Further, we study the Steiner tree augmentation problem (STAP) which is a generalization of the tree augmentation problem (TAP) in link arrival streams and provide matching upper and lower bounds (See the second row in Table 1). While our lower bound holds for link arrival streams, our algorithm works even in the more general fully streaming too. We remark that, while in the offline setting TAP and STAP admit similar approximations [45], there is a gap in their complexities in the streaming model.

**Graph augmentation in fully streaming.** We further show matching upper and lower bounds (up to a polylog$(n)$ factor) for $k$-CAP in the fully streaming setting (see the lower section in the first row of Table 1). The main component in our algorithm for solving $k$-CAP is an improved streaming algorithm for constructing *spanners on weighted graphs*. In particular, our upperbound implies that spanner is an optimal "universal" augmentation set for $k$-CAP.

**Improved streaming spanner in weighted graphs.** Given an $n$-vertex graph $G=(V,E)$ with a weight function $w\colon E\to\{0,\dots,W\}$, a subgraph $H\subseteq G$ is a $t$-spanner of $G$ if for every $(u,v)\in E$, the shortest $uv$-path in $H$ has weight at most $t\cdot w(uv)$. In streaming spanner, which is a well-studied problem [7, 15, 2, 33, 20], edges of $E$ arrive in an arbitrary order stream. While by using the standard weight-based partitioning trick, constructing an $O(t)$-spanner in $O(n^{1+1/t}\cdot\log W)$ words of space in one pass over the stream is straightforward (e.g., mentioned in [21]), it was not known whether the dependence on $\log W$ is crucial.[2]

[2] We remark that our contribution in removing the dependence on $\log W$ from the number of edges in spanner (and consequently from $k$-CAP) is conceptually interesting, as most graph streaming algorithms are mainly designed for unweighted graphs, and extending them to the weighted case typically incurs a $\log W$ loss.

**Table 1** Summary of our results for $k$-CAP, STAP, Spanner and SNDP in steaming models. All our problems are *weighted*. The space upper bounds are measured in words, while the lower bounds are in bits. We use $\tilde{O}(f)$ to mean $O(f \cdot \text{polylog} f)$ (it does not hide $\log W$ factors). All our algorithms are deterministic, whereas all lower bounds hold for randomized algorithms with constant success probability.

| Problem | Pass | Approx. | Space | Stream | Notes |
|---|---|---|---|---|---|
| $k$-CAP | 1 | $2+\epsilon$ | $O(\frac{n}{\epsilon}\log n)$ | link arrival | Theorem 1 |
| | | $2-\epsilon$ | $\Omega(n^2)$ bits | | Theorem 10 |
| | | $O(t)$ | $\tilde{O}(kn+n^{1+\frac{1}{t}})$ | fully streaming | Theorem 15 |
| | | | $\Omega(kn+n^{1+\frac{1}{t}})$ bits | | Theorem 11 |
| STAP | 1 | $O(t)$ | $\tilde{O}(n^{1+\frac{1}{t}})$ | fully streaming | Corollary 20 |
| | | | $\Omega(n^{1+\frac{1}{t}})$ bits | link arrival | Corollary 21 |
| Spanner | 1 | $O(t)$ | $\tilde{O}(n^{1+\frac{1}{t}})$ | edge arrival | Theorem 16 |
| | | | $\Omega(n^{1+\frac{1}{t}})$ bits | | Erdős' girth conjecture |
| SNDP | 1 | $O(t\log k)$ | $\tilde{O}(kn^{1+\frac{1}{t}})$ | edge arrival | Theorem 25 |
| | | $O(t)$ | $\Omega(n^{1+\frac{1}{t}})$ bits | | Corollary 21 |
| $k$-ECSS | $k$ | $O(\log k)$ | $O(kn\log n)$ | edge arrival | Corollary 26 |

Exploiting an even-odd bucketing approach, we provide a streaming algorithm with space complexity $O(n^{1+1/t}\cdot\log\min(W,n))$ words which by the well-known Erdős girth conjecture is basically the best one can hope for up to logarithmic factors. We further apply this even-odd bucketing to the $k$-CAP problem in the link arrival setting, and obtain a (more technical) algorithm (Theorem 1) with no dependence on $\log W$ in its space complexity.

**Streaming SNDP.** Finally, we describe an application of our results for designing the *first* one-pass streaming algorithms for the problem in insertion only edge arrival streams, where the edges of the input graph arrive in an arbitrary order stream.

In SNDP, given a graph $G=(V,E)$ with a weight function $w: E\to\{0,1,\ldots,W\}$ together with a *connectivity requirement* $r\colon V\times V\to\mathbb{Z}_{\geq 0}$, the goal is to find a minimum weight subgraph $H\subseteq G$ so that for every $s,t\in V$, $H$ contains $r(st)$ edge-disjoint paths connecting $s$ and $t$. A parameter of interest in SNDP is the maximum connectivity requirement $k=\max_{st} r(st)$. SNDP is a classic problem in combinatorial optimization and generalizes several well-studied problems such as MST, Steiner tree, $k$-edge connected spanning subgraph ($k$-ECSS), and $k$-CAP.

The fourth row of Table 1 shows our results for SNDP in edge arrival streams. In fact, our streaming algorithm works even for the more general problem of covering proper functions of the form $f: 2^V\to\{0,1,\ldots,k\}$ using the edges of $G$ (see Section B.2 for more details). $k$-ECSS, which itself is a basic problem in discrete optimization, is a variant of SNDP in which for every $s,t\in V$, $r(st)=k$. As a straightforward application of our algorithm for $k$-CAP in link arrival streams, we get a $k$-pass, $O(\log k)$-approximation for $k$-ECSS using $O(kn\log n)$ words of space. (See last row of Table 1).

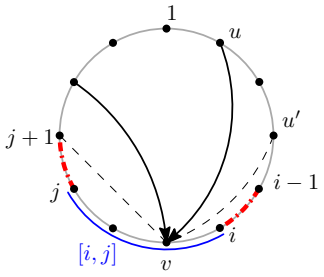
**Unweighted variant.** We remark that while we get tight algorithms for weighted $k$-CAP in both link arrival and fully streaming models, our lower bounds for link arrival does not hold for unweighted graphs. By a reduction from bipartite matching and invoking the result of [30], we observe a weaker lower bound that no streaming algorithm with $n\text{polylog}n$ space can achieve an approximation factor better than 1.409. Therefore, it remains an interesting open question to close the gap between 1.409 and 2 for unweighted $k$-CAP. Again, given that this lower bound is for tree augmentation, and the best known algorithm for (offline) TAP in unweighted graphs achieves an approximation factor of 1.326 [24], this again shows a gap between the two models for the problem in the unweighted variant.

## 1.3 Our Techniques

Given a streaming algorithm for the unweighted variants of both $k$-CAP and the spanner problems, an easy generalization to the weighted graphs is by partitioning the set of weights into $\log_{1+\epsilon} W$ number of classes and roughly running the unweighted sparsification on each class, resulting in $\epsilon^{-1} \log W$ blow up in the space usage. To remove the dependency on $\log W$ from the number of words, we follow an *even-odd bucketing* approach. More precisely, we partition the weights into much larger classes (i.e., *buckets*), such that the minimum and maximum weight in each class differ by $\text{poly}(n)/\epsilon$. This ensures that first, inside each class one can perform the weight-based partitioning to solve the problem while having only $\log n$ dependence in the space. Second, even picking all the edges from the $(i-2)$-th class $E_{i-2}$ is cheaper than picking any edge in the $i$-th class $E_i$ (i.e., it only introduces an extra $(1+\epsilon)$ multiplicative factor). This assumption allows us to infer additional properties about the graph once we are processing the edges in the class $E_i$, and shrink the problem significantly from each level $E_{i-2}$ to $E_i$. Thus our algorithm proceeds by separating the sparsification for the even-indexed buckets $E_{2i}$ and the odd-indexed buckets $E_{2i-1}$, and processes the buckets from smallest to largest weights.

**Spanner.** First, consider the spanner problem, and let $\mathcal{C} = \{C_1, \ldots, C_r\}$ be the set of connected components created by the edges from the classes upto $E_{i-2}$. The even-odd bucketing ensures that we only need to consider the edges from $E_i$ that are between two different components of $\mathcal{C}$. Thus, we shrink each connected component into a super-node and use the standard spanner algorithm with weight-based partitioning on this reduced graph. Note that the space usage of the algorithm is proportional to the number of *super-nodes with non-zero degree*. However, all such super-nodes will merge into bigger components for the next bucket $E_{i+2}$. Therefore the space usage of the algorithm for processing $E_i$ can be charged to the reduction in the number of super-nodes. Since the number of super-nodes starts from $n$ and goes down to 1, the total space usage of the algorithm can be bounded as a function of $n$. Finally, we need to perform the above process in a streaming setting: As we receive more edges in the stream, the components in $\mathcal{C}_i$ change but it is easy to maintain all required information in a streaming fashion.

**Link arrival $k$-CAP.** Our algorithm for $k$-CAP is more involved. First, by standard results in the literature, the problem reduces to cycle augmentation: given a cycle $C$, the goal is to augment it with a subset of edges from $L$ such that the resulting graph becomes 3-edge-connected. Let the nodes on the cycle be indexed 1 to $n$ in this order with vertex 1 being called the *root*. Now every cut of size 2 corresponds to two edges on the cycle. We specify such a cut with the interval $[i, j]$ with $1 < i \leq j \leq n$ that does not include the root. The goal is to cover all such cuts specified by these intervals.




First, using known ideas from [35, 36], we present a simple streaming algorithm for the unweighted variant of the problem as follows. We replace every link $uv$ by two directed links $\vec{uv}$ and $\vec{vu}$, (this is where the factor 2 in the approximation comes from), and we say that $\vec{uv}$ covers a cut $[i, j]$ if $v \in [i, j]$ and $u \notin [i, j]$. Now one can show that for $1 \leq u < u' < v$, it is always better to keep the edge $\vec{uv}$ than $\vec{u'v}$. Similarly, for $v < u' < u \leq n$, it is always better to keep the edge $\vec{uv}$ than $\vec{u'v}$. As a result, for each vertex, we keep at most two incoming edges. Therefore, the total space usage of the algorithm is only $O(n)$ in this case. Again this algorithm can be generalized to the weighted graphs using a weight-based partitioning, introducing a factor $\log W$.

To remove the dependency on $\log W$, again we consider the even-odd bucketing. This time, for each weight class $E_i$, we consider the 3-edge-connected components $C_1, \ldots, C_r$ formed by the edges in buckets upto $E_{i-2}$. Again using the even-odd bucketing plus the fact that the cycle is already 2-edge-connected, we can show that shrinking each of the 3-connected components into a super-node still works. The main challenge is that as opposed to the spanner setting, the problem on the super-node does not reduce to the same problem of cycle augmentation. This is because a single super-node does not necessarily span a consecutive set of vertices on the cycle. However, we note that in this case, the min-cuts on the cycle that do not fully include or fully exclude the vertices in a single super-node do not need to be considered. This allows us to reduce the space usage of the algorithm again to be proportional to the number of super-nodes and thus bound the total space usage of the algorithm as a function of $n$.

**Fully streaming $k$-CAP.** Our algorithm in this setting maintains two sketches. First, it keeps a $k$-connectivity certificate on the set of edges $E$ using a folklore streaming algorithm that keeps $k$ disjoint forests, which contains the information of all min-cuts of $E$ that need to be augmented in $k$-CAP. Second, employing our results on weighted spanners, the algorithm maintains a spanner for the set of (weighted) links. This means that every link $\ell$ of weight/length $w$ that we miss, can be replaced with a path of weight at most $O(t) \cdot w$, thus covering all the min-cuts originally covered by $\ell$. We show that this is a near-optimal algorithm one can get in this setting.

**Lower bounds.** Most of our lower bounds are via simple reductions from the INDEX problem in a two-party communication model, where we embed the bit-string held by Alice into edges of a graph, where by asking augmentation queries, Bob is able to tell whether edge $(u, v)$ exists in Alice's graph for any pair of vertices $u, v$. The most interesting one of our lower bounds (Theorem 12) shows that, in the fully streaming model, the space complexity for storing a spanner is essentially necessary. In the proof we let Alice hold a subgraph of a high-girth graph, and Bob wants to estimate the distance in this graph between $u, v$ (which is sufficient for telling whether $(u, v)$ is an edge, due to the high girth). Our proof reduces

this problem of estimating the distance between $u, v$ to the problem of augmenting a chain with end points $u, v$ into a 2-edge-connected graph. However, we also need rule out potential augmentation solutions that do not correspond to a $uv$-path.

**Applications.** Our algorithms for streaming connectivity augmentation also imply streaming algorithms for problems such as STAP, $k$-ECSS and SNDP. In particular, our one-pass algorithm for SNDP works by running $k$ instances of our streaming spanner algorithm in parallel, which store $k$ disjoint sparse subgraphs of the input graph that satisfy certain approximation guarantee. In particular, we show these $k$ disjoint "spanner-like" objects forms a coreset for SNDP instances with maximum connectivity requirement at most $k$.[3] Our approach follows the augmentation framework of [51, 26] to show the existence of an approximately good solution using edges from these $k$ sparse subgraphs.

## 1.4 Organization

In Section 2, we present our $(2+\epsilon)$-approximate algorithms for $k$-CAP in the link arrival model, and present a lower bound showing our approximation ratio is close to optimal. In Section 3, we study $k$-CAP in the fully streaming model, and present matching space lower bounds and upper bounds assuming our weighted spanner result. In Appendix A, we present our weighed spanner algorithm in the streaming model with better $\log W$ dependence. Finally in Appendix B we present further applications to other network design problems such as $k$-ECSS and SNDP. All missing proofs are deferred to the full version of the paper.

# 2 Connectivity Augmentation in Link Arrival Streams

In this section, we consider $k$-CAP, the problem of augmenting the connectivity of a given graph $G = (V, E)$ from $k-1$ to $k$ using a subset of weighted links $L \subseteq \binom{V}{2}$ in link arrival streams. To recall, in the link arrival model, a cactus representation of the graph $G$, which is of size $O(n)$ (see Definition 2 for the formal definition of cactus), is given to us in advance and the set $L$ arrives in the stream (see Section 1.1).

▶ **Theorem 1.** *The $k$-connectivity augmentation problem ($k$-CAP) on $(G = (V, E), L)$ in the link arrival model admits a one-pass $(2+\epsilon)$-approximation algorithm with total memory space $O(\frac{n}{\epsilon} \log \min(n, W))$ words, where $W = \max_{e \in E} w(e)$.*

Note that the augmentation set itself may have size $\Omega(n)$[4], so any algorithm that explicitly stores a solution must consume $\Omega(n)$ space. Moreover, we will show that just approximating the optimal total weight of the augmentation solution to any factor already requires $\Omega(n)$ bits of space. Hence, the space of our algorithm is tight up to a poly-logarithmic factor.

## 2.1 Preliminaries

**Cactus representation of min-cuts.** To increase the edge-connectivity of a $(k-1)$-connected graph $G$ to $k$, we need to add links to *cover* all min-cuts of size $k-1$. That is, for each cut $S$ of size $k-1$ (i.e., $|\delta_G(S)| = k-1$), we must add a link $e \in L$ such that $e \in \delta(S)$. Dinits, Karzanov, and Lomonosov [14] showed there is a compact representation of all min-cuts of an undirected graph by a *cactus graph*.

[3] In fact, the coreset guarantee holds even for the more general covering proper functions of the form $f : 2^V \to \{0, 1, \ldots, k\}$.

[4] As an example, consider a graph $G = (V, E)$ where $V = \{0, 1, \ldots, n-1\}$ and $E = \{(i, j) : j - i \in \{1, 2, \ldots, k\}\}$ (where indices are modulo $n$), which has edge connectivity $2k$. If the link set is $L = \{(i, i+1) : i \in [n]\}$, then at least $\lceil n/2 \rceil$ links are necessary to increase the edge connectivity by one.

▶ **Definition 2** (Cactus Graph). *A cactus graph is a 2-edge-connected graph $C = (V_C, E_C)$ where each edge in $E_C$ belongs to exactly one simple cycle. Note that we allow cycles of length 1 or 2 too.*

▶ **Lemma 3** ([14]). *Let $G = (V, E)$ be an undirected graph. There is a loopless cactus $C = (V_C, E_C)$ of size at most $2n - 1$ and a mapping $\varphi : V \to V_C$ so that a subset $S \in V$ is a min-cut of $G$ if and only if $\varphi(S)$ is a min-cut of $C$.*

*Moreover, when the min-cut size of $G$ is an odd integer, the cactus representation of $G$ is a spanning tree (we may still treat it as a cactus by duplicating each tree edge).*

The cactus representation is particularly useful for connectivity augmentation problems:

▶ **Corollary 4.** *Let $G = (V, E)$ be an undirected graph. Let $C$ denote the cactus representation of min-cuts in $G$. Then a link $(u, v) \in E \setminus E_H$ crosses a min-cut $S$ in $G$ if and only if the corresponding link $(\varphi(u), \varphi(v))$ crosses $\varphi(S)$ in $C$.*

▶ Remark 5. We remark that there is a simple streaming algorithm for constructing the cactus representation with space complexity $\tilde{O}(kn)$: First, construct a $k$-connectivity certificate $H$ of $G$ (recall that a $k$-connectivity certificate for a graph $G$ is a subgraph $H$ of $G$ that contains all edges crossing cuts of size $k$ or less in $G$, and at least $k$ edges from each cut of size more than $k$) with $O(kn)$ edges with space complexity $O(kn)$ words in polynomial time, using a simple algorithm by [41]. Then, we apply the algorithm of [34] for computing the cactus representation of the subgraph $H$ in $\tilde{O}(|E(H)|) = \tilde{O}(kn)$ time and space. It is straightforward to verify that the constructed cactus is a cactus representation of $G$, given $G$ is a $(k-1)$-connected graph.

We then get the following as a corollary of Theorem 1: If the algorithm receives a $k$-connectivity certificate as a representation of $G$ or the edges of $G$ arrive in the stream before any link arrives, we can construct a cactus representation of $G$ in $O(kn)$ space first and then run our algorithm in this section for cactus augmentation and the overall space complexity will be $O(nk + \frac{n}{\epsilon} \log n)$.

**Transforming cactus to cycle.** In the (weighted) *cactus augmentation* problem, without loss of generality, we can assume the cactus is a single cycle. The latter problem is known as weighted *cycle augmentation*. To reduce an instance on a general cactus to the single cycle case (without losing approximation factor), we apply the technique observed in [23, 50]: Unfold the cactus into its Eulerian circuit, then add additional zero-weight edges (which we can use to augment at no cost) to connect the nodes corresponding to the same junction node in the cactus. See Section 3 in [50] for a detailed description.

▶ **Lemma 6** (Theorem 3 in [23]; see also Lemma 2.2 in [50]). *Let $\alpha > 1$. If there is an $\alpha$-approximation algorithm for the weighted cycle augmentation problem, then the weighted cactus augmentation problem admits an $\alpha$-approximation.*

Note that this reduction only produces $O(n)$ extra zero-weight edges, so it does not affect the space complexity of the streaming algorithm. We can apply the unfolding technique in the preprocessing step and in the rest of this section, we assume that the cactus is a single cycle.

## 2.2 Main Step: Cycle Augmentation in Link Arrival Streams

We arbitrarily assign a root node on the cycle, and let its index be 0. Then let the vertices of the cycle be $V = \{0, 1, \ldots, n-1\}$, with edges $C = \{e_1, e_2, \ldots, e_n\}$ where $e_i = (i-1, i)$ (with indices modulo $n$). We first describe a 2-approximation for the unweighted case, using an idea from [35, 36].

▶ **Theorem 7.** *There exists a one-pass 2-approximation algorithm for the cycle augmentation problem on unweighted graphs with total memory space $O(n)$ edges.*

**Proof.** Following [35, 36], we consider a directed version of the problem defined as follows: given a set $E$ of *directed* edges, augment a minimum size subset $E' \subseteq E$ to the cycle, such that for every 2-cut $(L, V \setminus L)$ of the cycle where $0 \in V \setminus L$ (i.e., $L = \{l, l+1, \ldots, r\}$ for some $1 \leq l \leq r \leq n-1$), there exists $x\vec{y} \in E'$ with $y \in L$ and $x \in V \setminus L$ (we say $x\vec{y}$ covers $L$ in this case). To reduce the original (undirected) cycle augmentation instance to this directed problem, simply replace each input edge $(u, v)$ by two arcs $u\vec{v}, v\vec{u}$, incurring a 2-factor approximation: any directed solution $\{x\vec{y}\}$ implies an undirected solution $\{(x, y)\}$ of the same cost, and any undirected solution $\{(x, y)\}$ implies a directed solution $\{x\vec{y}\} \cup \{y\vec{x}\}$ of twice the cost.

Now we solve the directed instance exactly by an $O(n)$-space streaming algorithm. For each $v \in V$, we only need to keep the input arc $u\vec{v}$ with minimum indexed $u$, and keep the input arc $\vec{uv}$ with maximum indexed $u$. In this way we store only $O(n)$ arcs in total, and finally we run an offline exact algorithm (e.g., [22], which was also used by [36]) for the directed problem on these stored arcs. This does not affect optimality, because when $0 \leq u < u' < v$, any 2-cut, $U = \{l, l+1, \ldots, r\}$ covered by $u'\vec{v}$ is also covered by $\vec{uv}$, so we can discard $u'\vec{v}$ if we already have $\vec{uv}$ (a similar argument applies to the $v < u' < u \leq n-1$ case). ◀

By a simple scaling, this algorithm can be modified into a $(2+\epsilon)$-approximate algorithm for the weighted case with total space $O(\frac{n}{\epsilon} \log W)$ edges. Now we improve this $\log W$ dependency.

▶ **Theorem 8.** *The cycle augmentation problem on weighted graphs admits a one-pass $(2+\varepsilon)$-approximation streaming algorithm with total memory space $O(\frac{n}{\epsilon} \log \min(W, n))$ edges.*

**Proof.** We assume $\epsilon > 1/n$; otherwise use the trivial $O(n^2)$-space algorithm that stores the cheapest edge between every pair of vertices.

Define weight intervals $I_k = [(n/\epsilon)^k, (n/\epsilon)^{k+1})$. Let $E_k$ be the set of input edges $e$ that have arrived so far with weights $w(e) \in I_k$. Note that[5]

$$\frac{\min_{e \in E_{k+2}} w(e)}{\max_{e \in E_k} w(e)} > n/\epsilon. \tag{1}$$

These weight intervals do not contain zero, so we separately use a zero-weight class $E_{-1}$ to hold edges of zero weight. But for notational simplicity, we will not specially mention this zero weight class in later description. One can check that this does not affect the correctness of the algorithm.

Recall $C$ is the base cycle of length $n$. For each $k \in \{0, 1, \ldots, \lceil \log_{n/\epsilon} W \rceil\}$, define graph

$$G_k := C \cup \bigcup_{i \geq 0} E_{k-2i}. \tag{2}$$

Let $Q_k$ denote the collection of 3-edge-connected components of $G_k$, which form a partition of the $n$ vertices. See Figure 1 for an illustration. Let $1 \leq |Q_k| \leq n$ denote the number of components. Since $G_k \subseteq G_{k+2}$, $Q_k$ refines $Q_{k+2}$, and $|Q_k| \geq |Q_{k+2}|$.

[5] This inequality is meaningful only if both $E_k$ and $E_{k+2}$ are nonempty. This issue does not affect our overall argument since our algorithm can simply ignore the empty weight classes.

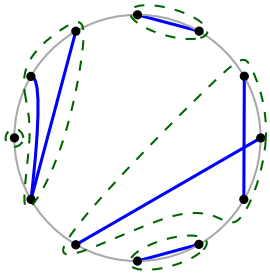


**Figure 1** An example of 3-edge-connected components $Q_k$ of the graph $G_k$. Thin black edges denote the base cycle, and thick blue edges denote the links from the set $\bigcup_{i\geq 0} E_{k-2i}$; together they form $G_k$. The dashed green lines describe the 3-edge-connected components of graph $G_k$.

**Algorithm description.** At any point, our streaming algorithm always stores a subset of the input edges $E = \bigcup_k E_k$, which includes the following:

1. **Undirected edges $F_k$:** We store edge subsets $F_k \subseteq E_k$, such that for all $k$ the subgraph $C \cup \bigcup_{i\geq 0} F_{k-2i} \subseteq G_k$ has the same 3-edge-connected components as $Q_k$.
2. **Directed arcs $S_k$:** For each $k$ and 3-edge-connected component $U \in Q_k$, and every weight interval $J_i = [(1+\epsilon)^i, (1+\epsilon)^{i+1}) \subseteq I_{k+2}$, we store the arc $x\vec{y}$ with minimum (and maximum) indexed $x$ where $(x, y) \in E_{k+2}$, $y \in U, x \notin U$, and $w(x, y) \in J_i$. The set of these arcs is denoted by $S_k$.

Now we describe how to maintain this information when a new edge $(u, v) \in E_{k'}$ arrives.

- **Maintain Item 1:** Note that adding this edge could potentially cause the components in $Q_{k'+2i}$ $(i = 0, 1, 2, \dots)$ to merge. To maintain Item 1 (and hence the knowledge of all $Q_k$), we insert $(u, v)$ into the current $F_{k'}$, and then run a clean up procedure to remove redundant edges: Start from the graph $H \leftarrow C \cup \bigcup_{j\geq 1} F_{k'-2j}$ which encodes the 3-connectivity information of the graph formed using edges prior to $E_{k'}$, and iterate over the edges $e \in F_{k'+2i}$ (in increasing order of $i = 0, 1, 2, \dots$). If adding $e$ to $H$ does not change the 3-edge-connected components of $H$, then remove $e$ from $F_{k'+2i}$. Otherwise add $e$ to $H$. It is clear that this clean up procedure preserves all the 3-connectivity information, since we start from the base graph $C$ which is already 2-edge-connected.
- **Maintain Item 2:** To maintain Item 2, we simply use arcs $u\vec{v}$ and $v\vec{u}$ to replace the existing ones that become dominated. When two 3-edge-connected components $U, U' \in Q_k$ merge, we also merge the stored information for $U, U'$ (compare the best arcs stored for these two components and keep the better one).
- **Offline step:** In the end, we run an offline exact algorithm (such as [22]) that solves the directed problem (see proof of Theorem 7) on the stored arcs in Item 2 and directed versions of the stored edges in Item 1.

**Space complexity.** For Item 1 the total space is $\sum_k |F_k| = \sum_j |F_{2j}| + \sum_j |F_{2j+1}|$ edges. We bound both terms separately. Due to our clean up procedure, there should be no redundant edges in $F_{\text{even}} = \bigcup_j F_{2j}$: starting from the base cycle $H \leftarrow C$, we can iterate over the edges $e \in F_{\text{even}}$ in certain order so that adding edge $e$ to $H$ always strictly decreases the number of 3-edge-connected components of $H$. Hence $|F_{\text{even}}| \leq n - 1$, and similarly $|F_{\text{odd}}| \leq n - 1$, so

$$\sum_k |F_k| \leq 2(n-1). \tag{3}$$

Define $c_{k+2}$ to be the number of 3-edge-connected components $U \in Q_k$ for which there exists $(u, v) \in E_{k+2}$ with $u \in U$ and $v \notin U$. Then the space for Item 2 is $\sum_k 2c_{k+2} \cdot \log_{1+\epsilon} \frac{\max_{e \in E_{k+2}} w(e)}{\min_{e \in E_{k+2}} w(e)} \leq \sum_k c_{k+2} \cdot O(\log(n/\epsilon)/\epsilon) \leq \sum_k c_{k+2} \cdot O(\log(n)/\epsilon)$ edges. We need the following lemma.

▶ **Lemma 9.** $c_{k+2} \leq 2(|Q_k| - |Q_{k+2}|)$.

Using this lemma, the space complexity for Item 2 is

$$\begin{aligned} O(\frac{\log n}{\epsilon}) \sum_k c_{k+2} &\leq O(\frac{\log n}{\epsilon}) \sum_k (|Q_k| - |Q_{k+2}|) \\ &\leq O(\frac{\log n}{\epsilon}) \cdot 2(n-1) \\ &\leq O(\frac{n \log n}{\epsilon}) \qquad \triangleright \text{ by summing over even and odd } k \text{ separately} \end{aligned}$$

So the total space complexity is $O(\epsilon^{-1} n \log n)$ edges.

**Proof of Lemma 9.** We first shrink the graph $G_k = C \cup \bigcup_{i \geq 0} E_{k-2i}$ into graph $H_k$. Let each node of $H_k$ represent a 3-edge-connected component $U \in Q_k$, and for every $(u, v) \in C$ (recall $C$ is the set of edges on the base cycle) with $u \in U \in Q_k$ and $v \in V \in Q_k$, we connect $U, V$ in $H_k$ by an edge (allowing self-loops and parallel edges). As a standard fact, $H_k$ is a cactus (allowing self loops), and $C$ corresponds to an Eulerian circuit of $H_k$.

Let $\mathcal{C}_k$ denote the collection of simple cycles (cycles with distinct vertices; we view a self loop as a simple cycle as well) of the cactus $H_k$. Then $\mathcal{C}_k$ can be viewed as a partition of the $n$ edges on the base cycle $C$, where $e, e' \in C$ belong to the same partition if and only if $\{e, e'\}$ is a 2-cut of $G_k$. Observe that $|\mathcal{C}_k| = n + 1 - |Q_k|$.[6] Hence, in the following it suffices to prove $c_{k+2} \leq 2 \cdot (|\mathcal{C}_{k+2}| - |\mathcal{C}_k|)$. Note that $\mathcal{C}_{k+2}$ is a finer partition of $C$ than $\mathcal{C}_k$.

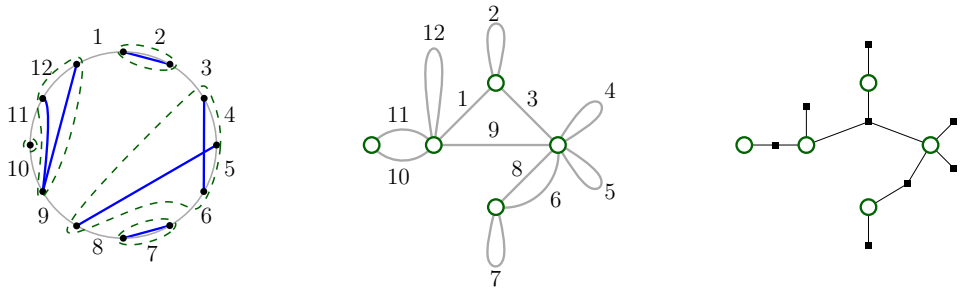



**Figure 2** A picture of the cactus $H_k$ (middle) produced by shrinking $G_k$ (left). The tree $T_k$ (right) is produced from cactus $H_k$.

We now consider how adding edges $E_{k+2}$ into $G_k$ can refine $\mathcal{C}_k$. Convert the cactus $H_k$ into a tree $T_k$ as follows. Let $T_k$ be a bipartite graph with vertex bipartition $(\mathcal{C}_k, Q_k)$, in which $D \in \mathcal{C}_k$ is connected to every $U \in Q_k$ that lies on the simple cycle $D$ in cactus $H_k$. Observe this bipartite graph $T_k$ is indeed a tree. For each $(u, v) \in E_{k+2}$, let $u \in U \in Q_k$ and $v \in V \in Q_k$, and we mark all the tree-edges on the unique path connecting $U$ and $V$ in $T_k$.

[6] To see this equality, consider removing one arbitrary edge from each simple cycle of the cactus, and the remaining edges should form a tree. As there are $|Q_k|$ vertices, the number of edges in the remaining tree is $|Q_k| - 1$, so the number of edges $n$ in the original cactus equals $|Q_k| - 1 + |\mathcal{C}_k|$, since we removed $|\mathcal{C}_k|$ edges in the removal step.

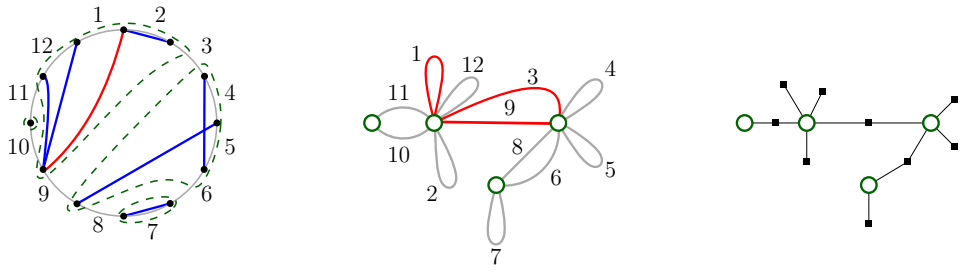



**Figure 3** After adding an edge from $E_{k+2}$ (depicted in red), the partition $\mathcal{C}_{k+2}$ refines the old partition $\mathcal{C}_k$: $\{1,3,9\}$ breaks into $\{1\}$ and $\{3,9\}$.

For each $D \in \mathcal{C}_k$, let $d(D)$ denote the number of marked tree-edges incident to the tree-node $D$ in $T_k$. Then, observe that $d(D) \in \{0\} \cup \{2,3,4,\dots\}$, and $D \in \mathcal{C}_k$ (viewed as a subset of $C$) breaks into $\max\{d(D),1\}$ subsets in the partition $\mathcal{C}_{k+2}$.

By assumption, there are at least $c_{k+2}$ many tree-nodes $U \in Q_k$ that are incident to at least one marked tree-edge in $T_k$, so $T_k$ contains at least $c_{k+2}$ marked tree-edges. Hence,

$$\begin{aligned}|\mathcal{C}_{k+2}| = \sum_{D\in\mathcal{C}_k} \max\{d(D),1\} &\geq \sum_{D\in\mathcal{C}_k} (1+d(D)/2) \quad \triangleright \text{since } d(D) \in \{0\}\cup\{2,3,4,\dots\} \\ &= |\mathcal{C}_k| + \frac{1}{2}\sum_{D\in\mathcal{C}_k} d(D) \geq |\mathcal{C}_k| + \frac{1}{2}c_{k+2},\end{aligned}$$

which completes the proof. ◀

**Approximation factor.** Let $\mathrm{OPT} \subseteq E = \bigcup_{k=-\infty}^{+\infty} E_k$ denote the optimal solution for the (undirected) cycle augmentation problem. Let $k^*$ be the maximum $k^*$ such that $\mathrm{OPT} \cap E_{k^*} \neq \varnothing$. Then by (1) we have

$$w(\mathrm{OPT}) > (n/\epsilon) \cdot \max_{e\in E_{k^*-2}} w(e). \tag{4}$$

Bidirecting OPT gives a solution $\mathrm{OPT}'$ for the directed problem with total cost $w(\mathrm{OPT}') = 2w(\mathrm{OPT})$. In the following we convert $\mathrm{OPT}'$ into a solution SOL for the directed problem that only uses arcs stored by the streaming algorithm, with total cost $w(\mathrm{SOL}) \leq (1 + O(\epsilon))w(\mathrm{OPT}') \leq (2 + O(\epsilon))w(\mathrm{OPT})$. This establishes that our streaming algorithm achieves $2 + O(\epsilon)$ approximation ratio for the (undirected) cycle augmentation problem.

In SOL we first include both directed versions of all $(u,v) \in \bigcup_{k\leq k^*-2} F_k$, with total cost at most

$$\begin{aligned}\sum_{k\leq k^*-2} 2|F_k| \cdot \max_{e\in F_k} w(e) &\leq \sum_k 2|F_k| \cdot \max_{e\in E_{k^*-2}} w(e) \\ &\leq 4(n-1)\cdot\frac{\epsilon}{n} w(\mathrm{OPT}) \qquad \triangleright \text{by (3) and (4)} \\ &\leq 4\epsilon w(\mathrm{OPT}).\end{aligned}$$

Then, for every arc $x\vec{y} \in \mathrm{OPT}'$ with weight $w(x\vec{y}) \in I_k$ where $k \in \{k^*-1, k^*\}$, we will find a replacement arc $x'\vec{y}' \in S_k$ stored by Item 2: Let $y \in U \in Q_k$. If $x \notin U$, then by Item 2 we can pick a stored arc $x'\vec{y}' \in S_k$ with $y' \in U$ and $w(x'\vec{y}') < (1+\epsilon)w(x\vec{y})$, such that $x' \leq x$ (if $x < y$) or $x' \geq x$ (if $x > y$). We include $x'\vec{y}'$ in SOL. (in the case of $x \in U$ we do not need to do anything)

By definition we immediately have $w(\text{SOL}) \leq 4\epsilon w(\text{OPT}) + (1+\epsilon)w(\text{OPT}') = (2+6\epsilon)w(\text{OPT})$. To show SOL is a feasible solution for the directed problem, we verify that each 2-cut $L = \{l, l+1, \ldots, r\}$ (where $1 \leq l \leq r \leq n-1$) is covered. There are three cases:

- **Case 1: $(L, V \setminus L)$ is not a 2-cut of $G_{k^*-2}$.** By Item 1, $G_{k^*-2}$ and $C \cup \bigcup_{i \geq 0} F_{k^*-2-2i}$ have the same 3-edge-connected components, and hence have the same 2-cuts, so $(L, V \setminus L)$ is also not a 2-cut of $C \cup \bigcup_{i \geq 0} F_{k^*-2-2i}$. Hence, there exists $(u', v') \in \bigcup_{i \geq 0} F_{k^*-2-2i}$ such that $u' \in V \setminus L, v' \in L$. Then, $\vec{u'v'}$ covers $L$, and by construction we have $\vec{u'v'} \in \text{SOL}$.
- **Case 2: $(L, V \setminus L)$ is not a 2-cut of $G_{k^*-3}$.** This case is similar to case 1.
- **Case 3: Otherwise.** In this case, $(L, V \setminus L)$ is a 2-cut of both $G_{k^*-2}$ and $G_{k^*-3}$. From the feasibility of OPT, we know there must exist arc $\vec{xy} \in \text{OPT}'$ that covers $L$ (i.e., $y \in L, x \in V \setminus L$) with weight $w(\vec{xy}) \in I_k$ where $k \in \{k^*-1, k^*\}$. Let $y \in U \in Q_{k-2}$. Since $(L, V \setminus L)$ is a 2-cut of $G_{k-2}$, we know $x, y$ cannot be in the same 3-edge-connected component of $G_{k-2}$, so $x \notin U$. Now let $\vec{x'y'} \in \text{SOL}$ be the replacement arc we found for $\vec{xy}$. By definition, $y' \in U$. We consider the case of $x < y$ (the other case $y < x$ is similar), and hence $x' \leq x$. In this case we must have $x < l \leq y \leq r$, so $x' < l$ and hence $x' \notin L$. Suppose for contradiction that $\vec{x'y'}$ does not cover $L$. Then we must have $y' \notin L$. But this would mean $(L, V \setminus L)$ is a 2-cut in $G_{k-2}$ separating $y$ and $y'$, contradicting the assumption that $y, y' \in U$ belong to the same 3-edge-connected component of $G_{k-2}$. This proves that the replacement arc $\vec{x'y'} \in \text{SOL}$ indeed covers $L$. ◀

The following shows that the approximation factor of our algorithm is close to optimal.

▶ **Theorem 10.** *Any streaming algorithm that solves the weighted TAP in the link arrival model with better than 2-approximation needs $\Omega(n^2)$ bits of space.*

## 3 Connectivity Augmentation in the Fully Streaming Setting

In this section, we first prove a space lower bound for $k$-CAP in the fully streaming model. Then, we show a streaming algorithm with nearly matching space complexity.

### 3.1 Lowerbound for Estimating Connectivity Augmentation Cost

Our main lower bound statement is the following.

▶ **Theorem 11.** *For any constant integer $t \geq 1$, the (unweighted) $k$-CAP (even when $k$ is known) in the fully streaming model requires space complexity $\Omega(kn + n^{1+1/t})$ bits (assuming the Erdős's girth conjecture) to approximate the solution size to a factor better than $2t+1$.*

It follows from combining two lower bound results Theorem 12 and Theorem 13.

**Lower bound in terms of approximation factor ($t$).** We first describe the space lower bound in terms of the approximation factor. As is standard in the spanner literature, the proof is based on high-girth graphs, but here we need to be more careful to make the connection between tree-augmentation and shortest paths.

▶ **Theorem 12.** *Consider the (unweighted) TAP where $E$ is the base tree and $L$ is the set of edges to augment, and $E \cup L$ arrive as a stream in an arbitrary order.*

*For any constant integer $t \geq 1$, any (randomized) streaming algorithm $\mathcal{A}$ that can output the size of a better than $(2t+1)$-approximate solution requires $\Omega(\gamma(n, 2t+1))$ bits of space, where $\gamma(n, 2t+1)$ denotes the maximum possible number of edges in an $n$–vertex graph with girth $> 2t+1$.*

We remark that the same lower bound of Theorem 12 also generalizes to $k$-CAP for higher values of $k > 2$, provided that we allow the base graph $E$ to have parallel edges.

**Lower bound in terms of connectivity parameter ($k$).** Zelke [53] gave a simple proof that computing the size of the minimum cut of an (unweighted) undirected graph requires $\Omega(n^2)$ bits of space for any one-pass streaming algorithm. In Zelke's construction the input graph has minimum cut size as large as $\Theta(n)$. Here we observe that Zelke's proof can be adapted to graphs with minimum cut size $\Theta(k)$, and show lower bounds for the connectivity augmentation problem.

We remark that [47] also obtained an $\Omega(kn)$-bit randomized lower bound and an $\Omega(kn \log n)$-bit deterministic lower bound for the $k$-CAP using a different proof.

▶ **Theorem 13.** *The $k$-CAP (where $k$ is known) in the fully streaming model (with unweighted links) requires $\Omega(nk)$ bits of space to approximate to any finite factor.*

▶ Remark 14. We remark that the same $\Omega(nk)$ lower bound also holds for the task of constructing a cactus representation of a graph (Lemma 3), even assuming the edge connectivity value $k$ is known. This is because the cactus representation immediately allows to distinguish between the cases of having two minimum cuts $C_1, C_2$ or one minimum cut $C_2$, and thus the proof above still applies.

## 3.2 Tight Algorithm

The main result of this section, whose details are deferred to the full version of the paper, is a single-pass algorithm that outputs a $(2t-1+\epsilon)$-approximate solution in $O(nk+\epsilon^{-1}n^{1+1/t}\log n)$ space, nearly matching the lower bounds of Theorem 12 and 13.

▶ **Theorem 15.** *The $k$-CAP in the fully-streaming model can be solved by a single-pass streaming algorithm with approximation ratio $(2t-1+\epsilon)$ in $O(nk+\epsilon^{-1}n^{1+1/t}\log n)$ space.*

## References


1 Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Comput. Sci. Rev.*, 37:100253, 2020.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Symposium on Principles of Database Systems*, pages 5–14, 2012.

3 Sepehr Assadi and Aditi Dudeja. A simple semi-streaming algorithm for global minimum cuts. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 172–180. SIAM, 2021.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proceedings of the Symposium on Discrete Algorithms*, pages 1723–1742, 2017.

5 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1345–1364, 2016.

6 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the international conference on Knowledge discovery and data mining*, pages 671–680, 2014.

7 Surender Baswana. Streaming algorithm for graph spanners–single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008.

8 Jarosław Byrka, Fabrizio Grandoni, and Afrouz Jabal Ameli. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to steiner tree. In *Symposium on Theory of Computing*, pages 815–825, 2020.

9 Federica Cecchetto, Vera Traub, and Rico Zenklusen. Bridging the gap between tree and connectivity augmentation: unified and stronger approaches. In *Symposium on Theory of Computing*, pages 370–383, 2021.

10 Chandra Chekuri, Alina Ene, and Ali Vakilian. Prize-collecting survivable network design in node-weighted graphs. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 98–109, 2012.

11 Chandra Chekuri, Alina Ene, and Ali Vakilian. Node-weighted network design in planar and minor-closed families of graphs. *ACM Transactions on Algorithms (TALG)*, 17(2):1–25, 2021.

12 Joseph Cheriyan and László A Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM Journal on Computing*, 43(4):1342–1362, 2014.

13 Michael S. Crouch, Andrew McGregor, and Daniel M. Stubbs. Dynamic graphs in the sliding-window model. In *Algorithms - ESA 2013 - 21st Annual European Symposium*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013.

14 E A Dinits, Alexander V Karzanov, and Micael V Lomonosov. On the structure of a family of minimal weighted cuts in a graph. *Studies in Discrete Optimization*, pages 290–306, 1973.

15 Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algorithms*, 7(2):20:1–20:17, 2011.

16 Guy Even, Jon Feldman, Guy Kortsarz, and Zeev Nutov. A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *Transactions on Algorithms (TALG)*, 5(2):1–17, 2009.

17 Jittat Fakcharoenphol and Bundit Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the $k$-vertex connected spanning subgraph problem. In *Proceedings of the Symposium on Theory of Computing*, pages 153–158, 2008.

18 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *International Colloquium on Automata, Languages, and Programming*, pages 531–543, 2004.

19 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *Journal on Computing*, 38(5):1709–1727, 2008.

20 Manuel Fernández V, David P Woodruff, and Taisuke Yasuda. Graph spanners in the message-passing model. In *Innovations in Theoretical Computer Science Conference*, 2020.

21 Arnold Filtser, Michael Kapralov, and Navid Nouri. Graph spanners by sketching in dynamic streams and the simultaneous communication model. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 1894–1913, 2021.

22 Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995.

23 Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, and Krzysztof Sornat. On the cycle augmentation problem: hardness and approximation algorithms. *Theory of Computing Systems*, 65:985–1008, 2021.

24 Mohit Garg, Fabrizio Grandoni, and Afrouz Jabal Ameli. Improved approximation for two-edge-connectivity. In *Symposium on Discrete Algorithms (SODA)*, pages 2368–2410, 2023.

25 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Symposium on Discrete Algorithms (SODA)*, pages 468–485, 2012.

26 MX Goemans, AV Goldberg, S Plotkin, DB Shmoys, É Tardos, and DP Williamson. Improved approximation algorithms for network design problems. In *Symposium on Discrete Algorithms (SODA)*, pages 223–232, 1994.

27 Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In *Symposium on Theory of Computing*, pages 632–645, 2018.

28 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76:654–683, 2016.

29 K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

30 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1874–1893, 2021.

31 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.*, 46(1):456–477, 2017.

32 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 1814–1833, 2020.

33 Michael Kapralov and David Woodruff. Spanners and sparsifiers in dynamic streams. In *Proceedings of the Symposium on Principles of Distributed Computing*, pages 272–281, 2014.

34 David R. Karger and Debmalya Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In Claire Mathieu, editor, *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 246–255, 2009.

35 Samir Khuller and Ramakrishna Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.

36 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, 1994.

37 Guy Kortsarz and Zeev Nutov. Approximating $k$-node connected subgraphs via critical graphs. *SIAM Journal on Computing*, 35(1):247–257, 2005.

38 Guy Kortsarz and Zeev Nutov. A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *Transactions on Algorithms*, 12(2):1–20, 2015.

39 Andrew McGregor. Finding graph matchings in data streams. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 170–181. Springer, 2005.

40 Hiroshi Nagamochi. An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree. *Discrete Applied Mathematics*, 126(1):83–113, 2003.

41 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse $k$-connected spanning subgraph of a $k$-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

42 Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the Symposium on Discrete Algorithms*, pages 1844–1860, 2019.

43 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *International Conference on Machine Learning*, pages 3829–3838, 2018.

44 Zeev Nutov. Approximating steiner networks with node-weights. *SIAM Journal on Computing*, 39(7):3001–3022, 2010.

45 R Ravi, Weizhong Zhang, and Michael Zlatin. Approximation algorithms for steiner tree augmentation problems. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2429–2448, 2023.

46 Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science Conference, ITCS*, volume 94 of *LIPIcs*, pages 39:1–39:16, 2018.

47 Xiaoming Sun and David P Woodruff. Tight bounds for graph problems in insertion streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

48 Vera Traub and Rico Zenklusen. A better-than-2 approximation for weighted tree augmentation. In *Foundations of Computer Science (FOCS)*, pages 1–12, 2022.

49 Vera Traub and Rico Zenklusen. Local search for weighted tree augmentation and steiner tree. In *Symposium on Discrete Algorithms (SODA)*, pages 3253–3272, 2022.

50 Vera Traub and Rico Zenklusen. A $(1.5+\varepsilon)$-approximation algorithm for weighted connectivity augmentation. In *Symposium on Theory of Computing*, pages 1820–1833, 2023.

51 David P Williamson, Michel X Goemans, Milena Mihail, and Vijay V Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 708–717, 1993.

52 Mariano Zelke. $k$-connectivity in the semi-streaming model. *arXiv preprint cs/0608066*, 2006. arXiv:cs/0608066.

53 Mariano Zelke. Intractability of min-and max-cut in streaming graphs. *Information Processing Letters*, 111(3):145–150, 2011.


# A Streaming Algorithm for Spanners on Weighted Graphs

In this section, we prove the following theorem on computing spanners for weighted graphs in the streaming model.

▶ **Theorem 16.** *For any integer $t \geq 1$, there is a one-pass streaming algorithm for computing a $(2t-1+\epsilon)$-spanner of size $O(\epsilon^{-1}n^{1+1/t}\log n)$ of a weighted graph, with space complexity $O(\epsilon^{-1}n^{1+1/t}\log n)$ words.*

Let $G=(V,E)$ be a weighted graph. We denote the weight function by $w : E \to \mathbb{R}^+$. Moreover, We normalize the weights so that $w(e) \in \{0\} \cup [1, W]$. For each $j \in [0, \lceil \log_{1+\epsilon} W \rceil]$, our algorithm stores $E_j$, a subset of edges of $G$ that have weights in $[(1+\epsilon)^j, (1+\epsilon)^{j+1})$. (These intervals do not contain zero, so we separately use a zero-weight class $E_{-1}$ to hold edges of zero weight. For notational simplicity, we will not mention this zero weight class in later description. One can check that this does not affect the correctness of the algorithm.)

Our algorithm is as follows (see Algorithm 2). As an edge $e$ arrives, round its weight to the nearest power of $(1+\epsilon)$ and place it in the corresponding weight class $E_j$. As usual, we keep the edge $e$ iff it does not close a cycle of length at most $2t$ in $E_j$, for some given parameter $t$. After processing the edge, we run the Sparsify subroutine described below in Algorithm 1.

**Sparsify subroutine.** Let $C > 0$ be a sufficiently large constant. Define intervals $I_k = [k \cdot (C/\epsilon)\log n, (k+1)\cdot(C/\epsilon)\log n]$. For all $k$ let $\tilde{E}_k := \bigcup_{j \in I_k} E_j$. For each $k$ let $E^{\text{even}}_{\leq k} = \bigcup_{j=-\infty}^{k} \tilde{E}_{2j}$ and $E^{\text{odd}}_{\leq k} = \bigcup_{j=-\infty}^{k} \tilde{E}_{2j+1}$. Let $E^{\text{even}} = \bigcup_j \tilde{E}_{2j}$, and we define $E^{\text{odd}}$ similarly. Our Sparsify procedure operates independently on these two sets. We will ensure that each set contains $O(\epsilon^{-1}n^{1+1/t}\log n)$ edges, independent of the weight bound $W$. We now describe how Sparsify operates on $E^{\text{even}}$ (the operations are the same for $E^{\text{odd}}$).

▷ Claim 17. Let the constant $C$ in the definition of the sets $\tilde{E}_k$ be chosen sufficiently large. Let $k$ be an integer. Let $H = (V, E^{\text{even}}_{\leq k-1})$. Then for any edge $e=(u,v) \in \tilde{E}_{2k}$ such that $u$ and $v$ belong to the same connected component in $H$, one has $w_e \geq \text{dist}_H(u,v)$.

Our procedure Sparsify$(k)$ performs the following step for each $k$ from $k_{\max}$ down to $k_{\min}$. Collapse the connected components induced by $E^{\text{even}}_{\leq k-1}$ into supernodes, and consider the multigraph with edges $\tilde{E}_{2k}$ on this set of supernodes. We convert this multigraph into a simple graph in the following natural way. For each edge $e=(u,v) \in \tilde{E}_{2k}$,

- delete $e$ if it is a self loop in this graph (i.e. $u, v$ belong to the same connected component)
- delete $e$ if there is a shorter edge that is parallel to $e$.

This is summarized in Algorithm 1. The algorithm is summarized in Algorithm 2.

▶ **Lemma 18.** *The edges stored by Algorithm 2 form a $(2t-1)\cdot(1+\epsilon)$-spanner of $G$.*

▶ **Lemma 19.** *Throughout the algorithm, the total number of edges stored by Algorithm 2 is always at most $O(\epsilon^{-1}n^{1+1/t}\log n)$.*

**Algorithm 1** SPARSIFY.

```
1: procedure SPARSIFY
2:     for k = k_max down to k_min do    ▷ The same procedure for the set E^odd
3:         Let H = (V, E^even_{≤k−1}) and let C_1, ..., C_r be the connected components of H.
4:         for e = (u, v) ∈ Ẽ_{2k} do
5:             if u, v ∈ C_i for some i then
6:                 delete e from Ẽ_{2k}
7:             end if
8:             if ∃(u', v') ∈ Ẽ_{2k} s.t. w_{(u',v')} ≤ w_e, and u, u' ∈ C_i, v, v' ∈ C_j for some i, j
   then
9:                 delete e from Ẽ_{2k}
10:            end if
11:        end for
12:    end for
13: end procedure
```

**Algorithm 2** Overall algorithm.

```
1: procedure SPANNER
2:     for each edge e = (u, v) in the stream do
3:         Round weight of e to power of 1 + ε. Let j be the weight class of e.
4:         Add e to E_j iff dist_{E_j}(u, v) > (2t − 1) · w_e.
5:         Call SPARSIFY
6:     end for
7: end procedure
```

# B Further Applications of Streaming Connectivity Augmentation

In this section, we show applications of our streaming algorithms for $k$-CAP for following well-studied *network design* problems: STAP, SNDP and $k$-ECSS.

## B.1 Steiner Tree Augmentation Problem (STAP) in Streaming

In STAP, we are given a set of vertices $V$ partitioned into *terminal* nodes ($R$) and *Steiner* nodes ($V \setminus R$), and a Steiner tree $T$ spanning the terminal set $R$. Then given a set of weighted links $L \subseteq \binom{V}{2}$, the goal is to find a minimum weight set of links $S \subseteq L$ such that $H = (V, E(T) \cup S)$ has 2 edge-disjoint paths between any pair of terminals. The problem is a special case of SNDP and can be approximate within a factor of 2 by iterative rounding method of Jain [29]. In light of recent developments for approximating tree augmentation and connectivity augmentation problems [49], Ravi, Zhang, and Zlatin [45] provided a $(1.5 + \epsilon)$-approximation for Steiner tree augmentation problem in polynomial time.

**Algorithm in fully streaming setting.** First, we observe that our results imply an algorithm for STAP in the fully streaming setting.

▶ **Corollary 20.** *STAP in the fully streaming model can be solved by a single-pass streaming algorithm with approximation ratio $(2t-1+\epsilon)$ and space complexity $O(\epsilon^{-1}n^{1+1/t}\log n)$ words.*

Note that the same fully streaming algorithm from Corollary 20 can also be used to solve STAP in the easier link arrival streams.

**Lower bound in link arrival streams.** Now we show that STAP has a lower bound nearly matching Corollary 20 *even in link arrival streams*. This shows a separation of STAP from the easier TAP: the latter problem has a better streaming algorithm in link arrival streams than in the fully streaming setting, whereas the former problem does not.

▶ **Corollary 21.** *For any constant integer $t \geq 1$, weighted STAP in link arrival streams requires space complexity $\Omega(n^{1+1/t})$ bits (assuming the Erdős's girth conjecture) to approximate the solution cost to a factor better than $2t+1$.*

## B.2 SNDP in Edge Arrival Streams

In this section, using our results and techniques from $k$-CAP and weighted spanners, we present a streaming algorithm for the general SNDP problem in edge arrival streams. We remark that our result in this section provide coresets for *covering functions* defined on cuts.

▶ **Lemma 22.** *Consider a weighted graph $G = (V, E)$ in an edge arrival stream. For integer $k \geq 1$ there is a one-pass streaming algorithm that computes $k$ disjoint edge subsets $S_1 \uplus S_2 \uplus \cdots \uplus S_k \subseteq E$ each of size $|S_i| \leq O(\epsilon^{-1} n^{1+1/t} \log n)$, in total space $O(k\epsilon^{-1} n^{1+1/t} \log n)$ words such that, for every $i \in [k]$ and every $e = (u, v) \in E \setminus (S_1 \cup S_2 \cup \cdots \cup S_i)$, there is be a path $P \subseteq S_i$ connecting $u, v$ with total length $w(P) \leq (2t - 1 + \epsilon)w(e)$.*

One of the main algorithmic approaches for SNDP is the augmentation framework pioneered by [51]. In this approach, the solution is constructed in $k$ phases and by the end of the phase $\ell$, the connectivity of every pair $u, v$ in the so-far-constructed solution is at least $\min\{\ell, r(st)\}$. So, the optimization problem of each phase is to increase connectivity of subset of pairs by one. More precisely, in each phase $\ell$, we need to pick a minimum-weight subgraph $H$ to cover a function $f_\ell : 2^V \to \{0, 1\}$. We say that a subgraph $H$ covers $f$ iff for every $U \subset V$, $\delta_H(U) \geq f(s)$. In the case of SNDP, for every $\ell \leq k$, $f_\ell$ is a skew-supermodular function and admits a 2-approximation via a primal-dual algorithm [51].

Next, We use Lemma 22 to show a coreset for covering $\{0, 1\}$ functions $f : 2^V \to \{0, 1\}$:

▶ **Definition 23.** *Given a weighted graph $G = (V, E)$, and a function $f : 2^V \to \{0, 1, \ldots, k\}$, find an edge subset $H \subseteq E$ with minimum total weight such that for all $U \subseteq V$ it holds that $|\delta_H(U)| \geq f(U)$. Throughout this section, we consider the functions $f$ arising from an instance of SNDP on $G$ with connectivity requirement function $r$ with maximum requirement $k$. Then, for every $U \subset V$, $f(U) := \max_{s \in U, t \in V \setminus U} r(st)$.*[7]

▶ **Lemma 24.** *Given a weighted graph $G = (V, E)$, let $S = S_1 \cup \cdots \cup S_k$ be the set of edges returned by the algorithm of Lemma 22. Then, the optimal solution for covering a function $f : 2^V \to \{0, 1, \cdots, k\}$ (arising from a SNDP instance on $G$) on graph $G' = (V, S)$ is an $O(t \log k)$-approximation of the optimal solution for covering $f$ on $G = (V, E)$.*

▶ **Theorem 25.** *SNDP with maximum connectivity requirement $k$ on a weighted graph $G = (V, E)$ admits a single-pass streaming algorithm with space complexity $O(kn^{1+1/t})$ words and approximation ratio $O(t \log k)$.*

Note that SNDP generalizes STAP, so the same lower bound for STAP from Corollary 21 also applies to SNDP. Specifically, for any constant integer $t \geq 1$, weighted SNDP requires space complexity $\Omega(n^{1+1/t})$ bits (assuming the Erdős's girth conjecture) to approximate the solution cost to a factor better than $2t + 1$.

[7] All results hold for a more general class of *proper* functions too. The function $f$ is called proper if $f(V) = 0$, $f(U) = f(V \setminus U)$ for every $U \subset V$ (symmetry), and $f(U_1 \cup U_2) \leq \max\{f(U_1), f(U_2)\}$ whenever $U_1$ and $U_2$ are disjoint (maximality).

**Min-Weight $k$-ECSS.** As a corollary of Theorem 1 for CAP in link arrival streams, we have the following guarantee for the problem of finding minimum-weight $k$-edge-connected spanning subgraph ($k$-ECSS), where given a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geq 0}$, the goal is to find a minimum-weight $k$-edge-connected subgraph $H \subseteq G$.

▶ **Corollary 26.** *There exists a $k$-pass $O(\log k)$-approximation algorithm for minimum-weight $k$-ECSS with total memory space $O(nk + n \log \min(n, W))$ where $W = \max_{e \in E} w(e)$.*

## C Related Work

**Approximation algorithms of $k$-CAP.** The edge-connectivity of a graph plays a central role in a wide range of network design problems, spanning both classical and modern problems. While the celebrated iterative rounding technique of [29] provides a 2-approximation for most of these problems, any better than 2-approximation for them are among main open problems within the field of approximation algorithms.

Significant progress has been made in achieving better than a 2-approximation for specific instances of the weighted $k$-CAP.Notably, extensive research focusing on the well-studied unweighted TAP has led to breakthroughs [40, 16, 38, 27, 9], culminating in an approximation factor of 1.326 [24]. Remarkably, this same factor has also been achieved for the unweighted $k$-CAP [9], a problem that recently saw significant advancements surpassing the 2-approximation barrier [8]. Moreover, in a recent development, the weighted TAP and $k$-CAP have witnessed breakthroughs with approximation factors below 2 [48, 49, 50]. It is noteworthy that these advancements in the weighted variants are relatively recent in the research landscape.

The Steiner tree augmentation problem, in which given a Steiner tree $T \subset G = (V, E)$ over terminals $R \subset V$ the goal is to find a minimum weight set of edges $H \subseteq G \setminus T$ that increases the connectivity of the set $R$ to 2, has also been studied and recently [45] provides $(1.5 + \epsilon)$-approximation generalizing some of the techniques in [49].

**SNDP.** Similarly to $k$-ECSS, the augmentation variant of SNDP has been extensively studied and is significant in the development of approximation algorithms for different variations of SNDP. Notably, the augmentation variant of SNDP generalizes well-studied problems such as TAP, STAP and $k$-CAP. The augmentation variant of SNDP was originally studied to analyze the primal-dual methods for SNDP, leading to $k$ and $\log k$ approximations [51, 26], and compared to the state-of-the-art 2-approximation iterative rounding technique of [29] has the advantage of applicability to other variants of SNDP such as node-weighted SNDP [44, 11, 10] or vertex-connectivity SNDP [37, 17, 12].

**Spanners and sparsifiers.** Graph spanners are important tools for graph compression in which the distances between the nodes are preserved. See [1] for a survey on graph spanners in general. Spanners have also been studied extensively in the streaming setting, see e.g., [7, 2, 15, 33, 21]. For other notions of graph sparsifiers in the streaming model, see e.g., [31, 32].