

# Network Construction with Subgraph Connectivity Constraints

Dana Angluin, James Aspnes\*, and Lev Reyzin\*\*

Department of Computer Science, Yale University  
51 Prospect Street, New Haven, CT 06511  
{dana.angluin, james.aspnes, lev.reyzin}@yale.edu

**Abstract.** We consider the problem introduced by Korach and Stern in [14] of building a network given connectivity constraints. A network designer is given a set of vertices  $V$  and constraints  $S_i \subseteq V$ , and seeks to build the lowest cost set of edges  $E$  such that each  $S_i$  induces a connected subgraph of  $(V, E)$ . First, we answer a question posed by Korach and Stern in [15]: for the offline version of the problem, we prove an  $\Omega(\log(n))$  hardness of approximation result for uniform cost networks (where edge costs are all 1) and give an algorithm that almost matches this bound, even in the arbitrary cost case. Then we consider the online problem, where the constraints must be satisfied as they arrive. We give an  $O(n \log(n))$ -competitive algorithm for the arbitrary cost online problem, which has an  $\Omega(n)$ -competitive lower bound. We look at the uniform cost case as well and give an  $O(n^{2/3} \log^{2/3}(n))$ -competitive algorithm against an oblivious adversary, as well as an  $\Omega(\sqrt{n})$ -competitive lower bound against an adaptive adversary. We examine cases when the underlying network graph is known to be a star or a path, and prove matching upper and lower bounds of  $\Theta(\log(n))$  on the competitive ratio for them.

## 1 Introduction

We consider the following problem introduced by Korach and Stern in [14] – users in a **trusted set** in a network want to send messages among themselves without having the messages travel outside the group. Trusted sets of users can overlap, creating complicated structures. We can imagine Yale University wanting to participate in a trusted set with some of its peer universities, but also wanting to participate in a trusted set of large institutions in Connecticut. These two separate trusted sets overlap on large universities in Connecticut.

Thus, each trusted set imposes a constraint on the network – namely that it be connected in its induced subgraph. The goal of the network designer is to build a network, at lowest cost, that satisfies the connectivity requirements presented to him. This task presents various natural variations. We can consider

---

\* Supported in part by NSF grant CNS-0435201.

\*\* This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

what happens if the constraints are given to the network designer in advance, and when the constraints arrive online. If they arrive online, they can be chosen adversarially or obliviously. We can imagine all edges in a network having the same cost, or that edges in a network have arbitrary costs. There are also cases when some information about the underlying network is known, for example, that there exists a path that satisfies all the constraints, as is the case in a serial network.

## 1.1 Past Work

In [14] Korach and Stern analyze the offline version of the Network Construction problem for the case where the constraints can be satisfied by a tree. They give a polynomial time algorithm that finds the optimal solution in the tree-realizable case. In [15] Korach and Stern consider this problem for the case where the optimal solution forms a tree, and all of the connectivity constraints must be satisfied by stars. They pose as an open question the case of general graphs. Among our results, we settle their question in this paper.

In a different line of work [3] Alon et al. explore a wide range of network optimization problems, including generalized connectivity, cuts, facility location, and multicast. The connectivity problem they study involves ensuring that a network with fractional edge weights has a flow of 1 over cuts specified by the constraints. In [2] Alon et al. also study approximation algorithms for the Online Set Cover problem, which has connections to Network Construction problems which we explore in this paper. In [13] Gupta et al. also consider a network design problem for pairwise vertex connectivity constraints.

In the area of active learning, the problem of discovering networks from connectivity queries has been much studied [1, 4, 6, 7, 12, 16]. In active learning of hidden networks, the object of the algorithm is to learn the network exactly. Our model is similar, except the algorithm only has the constraints it is given, and the task is to output the cheapest network consistent with the constraints.

Another motivation for studying this problem is to discover social networks from observations. This problem has also been studied in the active learning context [5]. Yet, we often passively observe phenomena that hint at the structure of an underlying social network. In the United States, the Centers for Disease Control and Prevention release various data<sup>1</sup> on persons affected by illnesses, and we can consider affected persons as connected subsets in a population. The problem would then be to find a maximum likelihood social network given the disease data. If the prior distribution on edges is independent and each edge appears with low probability, this is equivalent to finding a graph that satisfies all the connectivity constraints while minimizing the sum of log-likelihood costs for each edge we include.

---

<sup>1</sup> For more on CDC statistics, we direct the reader to [www.cdc.gov/datastatistics/](http://www.cdc.gov/datastatistics/).

## 1.2 Preliminaries

In this paper, we consider the following **Network Construction** problem.  $V$  is a set of vertices, and for each undirected edge  $e = (v_i, v_j)$ ,  $c_e$  is the cost of constructing edge  $e$ . A collection of connectivity constraints  $S = \{S_1, S_2, \dots, S_r\}$  is given, where each  $S_i$  is a subset of  $V$ . The task is to construct a set  $E$  of edges between vertices of  $V$  such that for each  $i$ , the set  $S_i$  induces a connected subgraph of  $G = (V, E)$ . The quality of the solution is measured by comparing the sum of the costs of all the edges in  $E$  with the optimal cost of satisfying all the connectivity constraints.

In the offline version of the problem, the algorithm knows all of the constraints at the outset; in the **Online Network Construction** problem, the constraints are given to the algorithm one by one, and edges must be added to  $G$  to satisfy each new constraint. By default, we allow the edges to have **arbitrary costs**, but in the **uniform cost** version of the problem the edge costs are all equal to 1.

When we restrict the **underlying graph** in a problem to a smaller class of graphs, we mean that all constraints  $S_i$  can be satisfied (for the online case, in hindsight), by a graph from that class.

## 1.3 Our Results

In Section 2 we analyze the offline problem, where we show that the Uniform Cost Network Construction problem (and therefore the arbitrary cost one) has a hardness of approximation lower bound of  $\Omega(\log(n))$  of the optimal solution. We give an algorithm that gives an  $O(\log(n) + \log(r))$  approximation, where  $r$  is the number of constraints. This matches the lower bound when  $r$  is polynomial in  $n$ .

In Section 3, we look at the Online Network Construction Problem. First, in Subsection 3.1, we look at the case when the underlying uniform cost graph is a star or path. In both cases, we show the the optimal algorithm has an  $\Omega(\log(n))$ -competitive ratio and give a matching  $O(\log(n))$ -competitive algorithm. We also show that in the case when edges have costs, there is no  $cn$ -competitive algorithm for any  $c < 1$ , even when the underlying graph is a path.

Then, in Subsection 3.2 we consider the general case of Online Network Construction, where the topology of the underlying graph is unrestricted. There we give an  $O(n \log(n))$ -competitive algorithm for the arbitrary cost case, that almost matches our lower bound. For the Uniform Cost Network Construction problem, we give an  $\Omega(\sqrt{n})$ -competitive lower bound, and in the case of an oblivious adversary, we give an  $O(n^{2/3} \log^{2/3}(n))$ -competitive algorithm.

## 2 Offline Network Construction

We first examine the Uniform Cost Network Construction problem in the offline case.

**Theorem 1.** *If  $P \neq NP$ , the approximation ratio for the **Uniform Cost Network Construction** problem on  $n$  nodes is  $\Omega(\log(n))$ .*

*Proof.* We reduce from the Hitting Set problem. The inputs to Hitting Set are  $U = \{v_1, v_2, \dots, v_n\}$  and  $\{C_1, C_2, \dots, C_j\}$  with  $C_i \subseteq U$ . The **Hitting Set** problem is to minimize  $|H|$ , where  $H \subseteq U$  such that  $\forall C_i, H \cap C_i \neq \emptyset$ . Let  $k$  be a constant. We define an instance of the Uniform Cost Network Construction problem with  $n^k$  by  $n$  vertices  $v_{(i,j)}$ , for all  $1 \leq i \leq n^k$  (rows) and  $1 \leq j \leq n$  (columns). For each  $i$ , the vertices in row  $i$ ,  $\{v_{(i,1)}, v_{(i,2)}, \dots, v_{(i,n)}\}$ , correspond to the elements  $\{v_1, \dots, v_n\}$  in the Hitting Set instance.

Now we define the connectivity constraints for the Uniform Cost Network Construction problem. First we enforce that all pairs of vertices in each row  $i$  are connected, by adding a connectivity constraint for each pair of vertices  $\{v_{(i,j)}, v_{(i,k)}\}$ . For each constraint  $C_i$  in the Hitting Set problem, we create  $\binom{n^k}{2}$  connectivity constraints. Without loss of generality, let  $C_i = \{v_1, v_2, \dots, v_k\}$ . For each pair  $l \neq j$  such that  $1 \leq l, j \leq n^k$  we add a connectivity constraint

$$S_{C_i}^{l,j} = \{v_{(l,1)}, v_{(l,2)}, \dots, v_{(l,k)}, v_{(j,1)}, v_{(j,2)}, \dots, v_{(j,k)}\} \quad (1)$$

in the Uniform Cost Network Construction problem. This enforces the Hitting Set constraints pairwise between the  $n^k$  rows of the network construction problem.

Each pair of rows in our new instance contains the original Hitting Set instance. First, the algorithm has no choice but to place a clique on each row. Then, let equation (1) be a constraint. To satisfy  $S_{C_i}^{l,j}$ , the algorithm must choose some edge between row  $l$  and row  $j$  among vertices  $1, \dots, k$ . We observe that if the algorithm chooses an edge between two vertices corresponding to different elements in the two rows, it could do at least as well by choosing the edge going between two copies of one of the two elements. To see this, if edge  $(v_{(l,x)}, v_{(j,y)})$ , with  $x \neq y$ , is chosen to satisfy the constraint  $S_{C_i}^{l,j}$ , edge  $(v_{(l,x)}, v_{(j,x)})$  would have also satisfied the constraint (and corresponds to choosing element  $x$  in the Hitting Set). Then, for any other constraint between the two rows,  $(v_{(l,x)}, v_{(j,y)})$  will satisfy it only if  $(v_{(l,x)}, v_{(j,x)})$  will. Hence an optimal algorithm chooses edges in one-to-one correspondence with the elements in the original Hitting Set instance.

Because Hitting Set is the complement of Set Cover, if  $P \neq NP$ , its optimal approximation ratio is  $\Omega(\log(n))$  [11], and there are  $\Theta\left(\binom{n^k}{2}\right)$  pairs of Hitting Set instances (or rows). The optimal solution has  $\left(n^k \binom{n}{2} + \text{OPT}\left(\binom{n^k}{2}\right)\right)$  edges – the first term counts the pairwise constraints in each row. So unless  $P=NP$ , the best polynomial time algorithm will require  $\left(n^k \binom{n}{2} + \Omega\left(\log(n)\text{OPT}\left(\binom{n^k}{2}\right)\right)\right)$  edges. Because  $k$  can be chosen to be arbitrarily large, this gives us the result.  $\square$

Below, we give an algorithm that almost meets this lower bound, even in the arbitrary cost case.

**Theorem 2.** *There is an  $O(\log(n) + \log(r))$ -competitive polynomial time approximation algorithm for the **Network Construction** problem on  $n$  nodes and  $r$  constraints.*

*Proof.* The inputs are the vertices  $\{v_1, v_2, \dots, v_n\}$ , the cost  $c_e$  of each edge  $e = (v_i, v_j)$ , and the constraints  $\{S_1, S_2, \dots, S_r\}$ . Let  $C$  be a potential function that sums over all constraints  $S_i$ , the number of components  $S_i$  induces in  $G$  minus 1. As long as  $C > 0$ , the constraints are not satisfied. We have  $C_0 = \sum_i (|S_i| - 1)$  because the graph initially has no edges, and each constraint  $S_i$  induces  $|S_i|$  components in  $G$ . Now, consider the following greedy algorithm: until all constraints are satisfied (while  $C > 0$ ), take the edge  $e$  that has the lowest ratio of  $c_e$  to  $\Delta C$ , the amount by which  $e$  reduces  $C$ . If in adding one edge, we have reduced  $C$  by  $k$ , then we say we have done  $k$  “reductions,” and paid a “price” per reduction of  $c_e/\Delta C$ . After  $C_0$  reductions, we are done, so we consider the reductions in the order that they occurred.

Now we consider how much we paid to do the  $k$ th reduction. Because  $(k-1)$  reductions had been done already,  $C_{k-1}$  was at most  $(C_0 - k + 1)$ . The optimal algorithm did all the reductions in OPT, so the average price it paid for all the remaining reductions is at most  $\text{OPT}/(C_0 - k + 1)$ . However, because we are choosing greedily, for the  $k$ th reduction, we pay a price no more than the optimal algorithm. So we look at the total price we paid (which is the cost of our algorithm), and it is  $\sum_{k=1}^{C_0} \text{OPT}/(C_0 - k + 1) = \text{OPT} \log(C_0)$ , which we can bound from above by  $\text{OPT} \log(nr) = \text{OPT}(\log(n) + \log(r))$ .  $\square$

### 3 Online Network Construction

In the online setting, the collection of connectivity constraints  $S_1, S_2, \dots, S_r$  is now given one at a time, and we say that upon being presented  $S_i$  the algorithm is on **round**  $i$ . Also, let  $E_i$  be the edge set after the algorithm satisfies constraint  $S_i$ . To explore the worst-case performance of our algorithms, unless otherwise stated, we assume an **adaptive adversary**, meaning that the adversary can wait for the algorithm to satisfy constraint  $S_i$  before determining constraint  $S_{i+1}$ .

In this section, we are interested in competitive analysis. An algorithm is  $c$ -**competitive** if the cost of its solution is less than  $c$  times OPT, where OPT is the best solution in hindsight. In the case when we know the underlying graph is, for instance, a uniform cost path or star, we know that  $\text{OPT} = (n-1)$ .

First, we prove a lemma helpful for analyzing online algorithms.

**Lemma 1.** *Let  $n(G, S)$  be the number of connected components  $S \subseteq V$  induces in  $G$ , and let  $G_i = (V, E_i)$ . For every algorithm for the **Online Network Construction** problem, there is an algorithm that performs at least as well and adds exactly  $(n(G_i, S_{i+1}) - 1)$  edges on every round  $i$ .*

*Proof.* Let  $A$  be any algorithm for the online network construction problem. We can make a new algorithm called  $A_{\text{lazy}}$ , that on each round inserts only a subset of edges that  $A$  has inserted up to that round, enough to keep the constraints satisfied. Each edge that  $A$  inserts but  $A_{\text{lazy}}$  does not,  $A_{\text{lazy}}$  remembers as possible edges for future rounds and adds them as needed to satisfy future constraints.

It is clear that  $A_{\text{lazy}}$  needs to put down a spanning tree on the components induced by constraint  $i$ , which is  $(n(G_i, S_{i+1}) - 1)$  edges; any fewer edges would not satisfy the constraint. Thereby,  $A_{\text{lazy}}$  satisfies the constraints, and because  $A_{\text{lazy}}$  uses a subset of the edges of  $A$ , it performs at least as well.  $\square$

### 3.1 Stars and Paths

First, we examine the case when the underlying graph is a star. This is a natural framework for Network Construction Problems because it corresponds to the case when the constraints can be satisfied by a network with one server (the star's center) and the rest clients (the leaves.)

**Theorem 3.** *The optimal competitive ratio for the **Uniform Cost Network Construction** problem on  $n$  nodes when the algorithm knows the underlying graph is a **star** is  $\Theta(\log(n))$ .*

*Proof.* We first prove the *lower bound* – that the competitive ratio for any algorithm is  $\Omega(\log(n))$ . The adversary maintains a partition of the vertices into two sets:  $C$ , the possible centers, and  $D = (V - C)$ , the non-centers. Initially  $C$  has  $(n - 1)$  vertices and  $D$  has one vertex, and the initial two constraints given to the algorithm are  $V$  and  $C$ . At every step, the adversary looks for a vertex  $v \in C$  that maximizes the number of edges  $(u, v)$  with  $u \in D$  given by the algorithm, and moves  $v$  from  $C$  to  $D$ , that is,  $C' = C \setminus \{v\}$  and  $D' = D \cup \{v\}$ . The new constraints given by the adversary are  $C' \cup u$  for all  $u \in D'$ . Thus, the algorithm must ensure at least one edge from each element of  $D'$  to some element of  $C'$ . The adversary continues until it has moved all but one vertex from  $C$  to  $D$ .

To analyze, we consider the edges from elements of  $D$  to the element  $v$  moved from  $C$  to  $D$  when  $|C| = i$ . Each element of  $D$  must have at least one edge to an element of  $C$ , so the maximum number of edges from  $D$  to one element of  $C$  is at least the average:  $(n - i)/i$ . These edges are all distinct, so the algorithm must produce at least  $\sum_{i=2}^{n-1} (n - i)/i = \Omega(n \log(n))$  edges in all. Yet, all these constraints can be satisfied by a star with  $(n - 1)$  edges. This completes the proof of the lower bound.

For the *upper bound*, we give an  $O(\log(n))$ -competitive algorithm. The algorithm will keep track of a set  $C_i$  of potential centers and  $D_i = V - C_i$  known non-centers at round  $i$ . Any node not appearing in some constraint cannot be a center. The algorithm keeps nodes in  $C_i$  connected by a path, and each node in  $D_i$  is connected to some node in  $C_i$ , such that the number of edges going into each node in  $C_i$  from  $D_i$  is no more than  $\lceil (|D_i|)/|C_i| \rceil$ , meaning that all nodes in  $C_i$  have close to the same degree. Initially,  $C_0 = V$  and is connected by an arbitrary path (costing  $O(n)$  edges). At any stage of the algorithm, when a constraint  $S_i$  comes in, if it does not eliminate any potential centers, it is easy to see  $S_i$  is already satisfied. Otherwise, we remove any potential centers  $R_i \subset C_{i-1}$  that are now known to be non-centers from  $C_{i-1}$  (to form  $C_i$ ), and we add them to  $D_{i-1}$  (to form  $D_i$ ). Further, we ignore all edges to nodes in  $R_i$ . We re-stitch the path connecting nodes in  $C_i$ , which takes at most  $|R_i| + 1$  edges. Then, we

connect (in such a way that keeps the degrees of the nodes in  $C_i$  about equal) all nodes in  $R_i$  to nodes in  $C_i$ , which takes  $|R_i|$  edges, and also all nodes in  $D_{i-1}$  that became disconnected from  $C_i$  because were connected to nodes in  $R_i$ , which takes  $O\left(\frac{|R_i||D_{i-1}|}{|C_{i-1}|}\right)$  edges. This clearly satisfies constraint  $S_i$ .

To see why this gives us the needed result, we notice at at most  $n$  centers can be removed from  $C$ , and therefore connections involving nodes in  $R_i$  take  $\sum_{i=1}^n O(|R_i|) = O(n)$ . The rest of the connections, by the analysis in the paragraph above, cost  $\sum_i O\left(\frac{|R_i||D_{i-1}|}{|C_{i-1}|}\right) \leq \sum_i O\left(\frac{|R_i|n}{|C_{i-1}|}\right)$ . If we consider removing one center at a time (as opposed to in groups  $R_i$ ), we can bound this from above by  $O(n \sum_{i=1}^n \frac{1}{n-i}) = O(n \log(n))$ .  $\square$

Next, we examine another natural network structure – when the underlying network is a path. This is the case when the constraints can be realized by a serial network

**Theorem 4.** *The optimal competitive ratio for the **Uniform Cost Network Construction** problem on  $n$  nodes when the algorithm knows the underlying graph is a **path** is  $\Theta(\log(n))$ .*

*Proof.* First we prove the *lower bound*, that any algorithm has a competitive ratio of  $\Omega(\log(n))$ . We show an adversarial strategy that forces the algorithm to use  $O(n \log(n))$  edges when the optimal solution in hindsight uses only  $(n - 1)$  edges. The adversary first shows all the nodes, which by Lemma 1 the optimal algorithm connects using  $(n - 1)$  edges. Then the adversary divides the nodes into two independent sets and presents each of them to the algorithm in arbitrary order. The optimal algorithm must connect the two subgraphs with trees (again by Lemma 1), and the adversary repeats this process recursively. We say that each depth in the recursion is a new level in this process. Because the algorithm puts down  $O(n)$  edges per level, given this strategy for the adversary, the optimal algorithm needs to put down a path at each step so as to balance the sizes of two following independent sets and limit the algorithm to  $O(\log(n))$  levels. Hence, the algorithm uses  $\Omega(n \log(n))$  edges, but it is clear that knowing the sets in advance, one can satisfy the connectivity requirements using  $O(n)$  edges - by simply connecting the smallest sets and then merging them accordingly into a path. This gives us the desired  $\Omega(\log(n))$  gap.

Now we prove the *upper bound* by giving an  $O(\log(n))$ -competitive algorithm. We first observe that every constraint  $S_i$  is a sub-interval of the path, and the algorithm must put down enough edges to capture a permutation of the vertices consistent with the  $S_i$ 's. The algorithm we introduce maintains a **pq-tree** – a data structure, introduced by Booth and Lueker in [8], that keeps track of all consistent orderings of nodes given contiguous intervals in a permutation. A pq-tree is a tree that consists of leaf nodes, p-nodes, and q-nodes. A **leaf node** is an element (or vertex in our case), A **p-node** (permutation node) has 2 or more children of any type, and its children form a contiguous interval, but can be ordered in any order. A **q-node** has 3 or more children of any type and its children form an interval in the given order or its reverse. Each new interval

constraint updates the pq-tree, and then the algorithm adds edges to satisfy the new constraint.

We will show that the algorithm can satisfy the constraints using  $O(n \log(n))$  edges by using a potential function to keep track of the evolution of the pq-tree. Let  $P$  be the set of p-nodes in a given tree and  $Q$  be the set of q-nodes. Also for any node  $p$ , let  $c(p)$  count  $p$ 's children. For constants  $a$  and  $b$ , our potential function is

$$\Phi = a \sum_{p \in P} ((c(p) - 1)(\log(c(p) - 1)) + b|Q|). \quad (2)$$

We observe that the pq-tree before any constraints arrive has one p-node at the root, and all its children are leaf nodes. This corresponds to an arbitrary permutation of the vertices. So at the beginning,  $\Phi = \Theta(n \log(n))$ . In comparison, when the permutation is specified, the root is a q-node and the rest of the nodes are leaves. In that case,  $\Phi = \Theta(1)$ .

Now we look at what happens when a constraint comes in. We will argue that the number of edges we need to insert into our graph is a lower bound on the drop in the potential function, and because it is always the case that  $\Phi \geq 0$ , this will complete the proof.

We first analyze the most common type of update to a pq-tree. A constraint comes in and splits a known interval into two, that is, it splits a p-node with  $m$  children into two p-nodes (one with at most  $(k + 1)$  children and the other with at most  $(m - k)$  children), and attaches them to a q-node parent. So the drop in the potential function is as follows (where  $H(p)$  is the binary entropy function.)

$$\begin{aligned} -\Delta\Phi &= a((m - 1) \log(m - 1) - ((k) \log(k) + (m - k - 1) \log(m - k - 1))) - b \\ &= a(m - 1) \left( \log(m - 1) - \frac{k}{m - 1} \log(k) - \frac{m - k - 1}{m - 1} \log(m - k - 1) \right) - b \\ &= a(m - 1) \left( -\frac{k}{m - 1} \log\left(\frac{k}{m - 1}\right) - \frac{m - k - 1}{m - 1} \log\left(\frac{m - k - 1}{m - 1}\right) \right) - b \\ &= a(m - 1) H\left(\frac{k}{m - 1}\right) - b \\ &\geq a(m - 1) \min\left(\frac{k}{m - 1}, \frac{m - k - 1}{m - 1}\right) - b \\ &= a \min(k, m - k - 1) - b. \end{aligned}$$

Now,  $2 \min(k, m - k - 1)$  is exactly how much is required in the worst case to stitch up a split interval – because we have to connect up all of the nodes in the smaller new interval, and patch at most as many gaps in the larger interval (similar to the reasoning in the proof of the lower bound). It takes at most 4 more edges to connect up the ends of the two new intervals to the rest of the graph, and this can be paid for if  $a = 10$  and  $b = 4$ . We remember  $\min(k, m - k - 1) \geq 1$ , so we spend 2 on splitting the p-node, 4 on re-stitching, and 4 on the new q-node, and thus  $a = 10$ .

Booth and Lueker in [8] characterized all of the possible updates to the pq-tree using 10 patterns: L, P1, P2, P3, P4, P5, P6, Q1, Q2, and Q3. They are

given in Appendix A. Neither L, Q1, nor P1 changes the number of p-nodes or q-nodes. P2-P6 split at most one p-node and create at most one q-node, and are covered by our analysis above. Q2 and Q3 require us to reconnect at most 2 pairs of endpoints (with 4 edges), but also reduce the number of q-nodes by 1 or 2 (this is why  $b = 4$ ), and the edges are paid for by the drop in the potential function.  $\square$

In the arbitrary cost case, the competitive ratio becomes considerably worse.

**Theorem 5.** *There is no  $(cn)$ -competitive algorithm for  $c < 1$  for the **Online Network Construction** problem on  $n$  vertices, even when the underlying graph is a **path**.*

*Proof.* We let all edges among  $(n - 1)$  of the vertices have cost 0, and all edges from the remaining vertex,  $s$ , have cost 1. The adversary first tells the algorithm that all the vertices are connected. When the algorithm satisfies this constraint, the adversary excludes from the next constraint all vertices the algorithm has chosen to directly connect to  $s$ . This continues until the adversary forces the algorithm to use all the 1 edges. But because each constraint is a subset of the previous constraint, the optimal solution only needs to contain the final cost 1 edge, and can connect the remaining vertices using a path that goes through the vertices in the order they were excluded in the adversary's choice of constraints. Hence, the algorithm was forced to pay a cost of  $(n - 1)$ , while the optimal solution pays a cost of 1.  $\square$

### 3.2 General Graphs

We introduce the **Online Fractional Network Construction** problem, in which the algorithm is similarly given a set of vertices  $V$  and edge costs  $c_e$  for all  $e = (v, w) \in V$ , and sees a sequence of constraints  $\{S_1, S_2, \dots, S_r\}$ . The task is to assign fractional weights  $w_e$  to the edges (or pairs of vertices), such that for each  $i$ , the maximum flow between each pair of vertices in  $S_i$  is at least 1, given the weights  $w_e$  (to be interpreted as edge capacities). The quality of the solution is measured by comparing  $\sum c_e w_e$  with the optimal cost of satisfying all the connectivity constraints. In the online problem, the algorithm may not decrease any edge weights from round to round.

**Lemma 2.** *There is an  $O(\log(n))$ -competitive polynomial time algorithm for the **Online Fractional Network Construction** problem on  $n$  nodes.*

*Proof.* We give Algorithm 1 for the Online Fractional Network Construction problem.

In Appendix B, we prove Lemma 3, that Algorithm 1 gives a  $\log(n)$  approximation to the Online Fractional Network Construction problem. Algorithm 1 is a modification of the algorithm in **3.1** of Alon et al. [3], and the proof in Appendix B closely follows their logic.  $\square$

---

**Algorithm 1** An  $O(\log(n))$ -competitive Algorithm for the Online Fractional Network Construction Problem

---

Let  $|V| = n$  and  $|E| = m$   
Upon seeing first constraint, set all  $w_e = \frac{1}{m^2}$   
**for** each constraint  $S$  **do**  
  **for** each pair  $v, w \in S$  **do**  
    **if** the flow from  $v$  to  $w$  in  $S$  is at least 1 **then**  
      do nothing  
    **else**  
      **while** the flow from  $v$  to  $w$  in  $S$  is less than 1 **do**  
        compute a min-weight cut  $C$  between  $v$  and  $w$  in  $S$ .  
        **for** each edge  $e \in C$ ,  $w_e = w_e(1 + 1/c_e)$   
      **end while**  
    **end if**  
  **end for**  
**end for**

---

We can now use Lemma 2 to develop an algorithm that almost matches the lower bound from Theorem 5.

**Theorem 6.** *There is an  $O(n \log(n))$ -competitive polynomial time algorithm for the **Online Network Construction** problem on  $n$  nodes.*

*Proof.* We take the algorithm for solving the Online Fractional Network Construction problem from Lemma 2, and use it together with a rounding scheme similar to the one considered by Buchbinder [9] for solving linear programs, to get our result.

For each edge  $e$ , we choose  $2n$  random variables  $X(e, i)$  independently and uniformly from  $[0, 1]$ . For each edge, we let threshold  $T(e) = \min_{i=1}^{2n} X(e, i)$ . Then we run the algorithm for the Online Fractional Network Construction problem, and whenever  $w_e \geq T(e)$ , we add  $e$  to our integral solution, and continue. Now we claim the following.

1. The integral solution has expected cost  $O(n)$  times the fractional solution.
2. The integral solution satisfies all the constraints with high probability.

To prove the first claim, for any edge  $e$ , the probability that  $X(e, i) < w_e$  is  $w_e$ . The probability that  $e$  is chosen to be in the integral solution is the probability that some  $X(i, s) < w_e$  – we call this event  $A_i$ . Hence, the probability of  $\cup_{i=1}^{2n} A_i$  is  $2nw_e$ , and by linearity of expectation, on every round, the expected cost of our solution is  $O(n)$  times the fractional solution, which is a  $O(\log(n))$  approximation of OPT. Hence our solution is an  $O(n \log(n))$  approximation of OPT in expectation.

To prove the second claim, we pick a constraint  $S$ . The constraint  $S$  is satisfied if and only if for every cut  $C \in S$ , there exists an edge crossing  $C$  in our solution. We fix a cut  $C$ . The probability the cut is not crossed is the probability we have not chosen any edge crossing the cut. This probability is  $\prod_{e \in C} (1 - w_e)^{2n} \leq$

$e^{(-2n \sum_{e \in C} w_e)}$ . And because the cut is crossed with a flow of 1 in the fractional solution (i.e.  $\sum_{e \in C} w_e \geq 1$ ) at the time it is considered by the algorithm, we can bound this by  $\frac{1}{e^{2n}}$ . There are  $r$  constraints and at most  $2^n$  cuts per constraint, so by the union bound, the probability our solution is not feasible is  $(\frac{r2^n}{e^{2n}})$ . Because  $r < 2^n < e^n$ , the probability our solution is not feasible tends to 0 as  $n$  increases, completing the proof.  $\square$

We note that it is tempting to try to improve the bound for the Online Network Construction problem by reducing it to Online Set Cover with the hope of getting better bounds by using algorithms from [2] or [9]. In Online Set Cover, the sets to be covered are given to the algorithm in advance, and the elements come in online. In Online Network Construction, the graph is known in advance, but there are exponentially many constraints in the size of the graph that can arrive online. While a reduction that makes edges in the network graph correspond to sets and all cuts induced by the connectivity constraints correspond to elements in the Online Set Cover problem is possible, the resulting bound on the competitiveness ratio is  $O(n \log(n))$ , which yields no improvement over our algorithm.

We now make a simple observation for the uniform cost case.

**Proposition 1.** *There is an  $O(n)$ -competitive polynomial time algorithm for the **Online Uniform Cost Network Construction** problem on  $n$  nodes.*

*Proof.* Consider the algorithm that puts down a clique for each constraint presented to it. Let  $q \leq n$  be the number of nodes that appear in at least one constraint. Our algorithm uses  $O(q^2)$  edges, but the optimal algorithm must clearly use at least  $\Omega(q)$  edges.  $\square$

We also present a lower bound for the Online Uniform Cost Network Construction problem.

**Theorem 7.** *The **Online Uniform Cost Network Construction** problem on  $n$  nodes has an  $\Omega(\sqrt{n})$ -competitive lower bound.*

*Proof.* We divide the vertices into two sets  $Q$  and  $R$ , with  $|Q| = \sqrt{n}$  and  $|R| = n - \sqrt{n}$ . For each  $v_i \in R$ , the adversary does the following. He begins at stage  $t = 1$  and sets  $Q_{(i,1)} = Q$ . At stage  $t$ , the adversary gives the learner the constraint  $S_{(i,t)} = Q_{(i,t)} \cup v_i$ . Let  $C_{(i,t)}$  be the set of vertices in  $Q$  which the learner connects to  $v_i$  in response to being presented  $S_{(i,t)}$ . The adversary sets  $Q_{(i,t+1)} = Q_{(i,t)} \setminus C_{(i,t)}$  and continues to the next stage. The adversary stops when  $Q_{(i,t)} = \emptyset$ .

To analyze this strategy for the adversary, for each  $v_i$ , we order the edges from  $v_i$  to  $R$  by the stage in which the learner has placed them, breaking ties arbitrarily. It is clear that the last edge the learner places is sufficient to connect  $v_i$  to  $R$  for all constraints  $S_{(i,t)}$ . Hence, all of these constraints can be satisfied in retrospect by placing a clique on  $Q$  using  $\binom{\sqrt{n}}{2} = O(n)$  edges and one edge per vertex in  $R$ , also using  $O(n)$  edges. However, the learner places  $\Omega(n)$  edges

per vertex in  $Q$ , amounting to  $\Omega(n\sqrt{n})$  edges in total, giving the desired result.  $\square$

We now consider the Online Network Construction problem with an **oblivious adversary** – an adversary who commits to the constraints  $\{S_1, S_2, \dots, S_r\}$  before presenting any of them to the algorithm.

**Theorem 8.** *There is a polynomial time algorithm for the **Online Uniform Cost Network Construction** problem on  $n$  nodes that gives an  $O(n^{2/3} \log^{2/3}(n))$ -competitive ratio against an oblivious adversary.*

*Proof.* We assume that the optimal solution has  $m = \Omega(n)$  edges (that each vertex appears in some constraint). We then create an Erdős Rényi random graph on our graph  $G$ , by putting in edges independently with a specified probability. Random graph connectivity has a sharp threshold of  $\frac{c \log(n)}{n}$  for  $c > 1$  [10]. When  $p = \frac{c \log^{2/3}(n)}{n^{1/3}}$ ,  $G$  has  $O(n^{5/3} \log^{2/3}(n))$  edges in expectation. Now, our algorithm is simple – for each constraint  $S_i$  such that  $|S_i| \geq n^{1/3} \log^{1/3}(n)$ , because of our choice of  $p$ ,  $S_i$  is already connected with high probability in  $G$ . Because we assume that there are only polynomially many constraints (even in the offline case, as in Theorem 2), for large enough  $c$ , all such constraints are satisfied in expectation. For every constraint  $S_i$  of size  $< n^{1/3} \log^{1/3}(n)$  that we see, we can put a clique with  $O(n^{2/3} \log^{2/3}(n))$  edges on that constraint, and each time we do that, we are guaranteed to hit at least one edge in OPT. Hence, this costs us  $O(n^{5/3} \log^{2/3}(n) + n^{2/3} \log^{2/3}(n) \text{OPT})$  edges in expectation, and because  $m = \Omega(n)$ , we have an  $O(n^{2/3} \log^{2/3}(n))$  approximation ratio.  $\square$

## 4 Open Problems

We leave open some interesting questions. In the offline case, we give an  $\Omega(\log(n))$  hardness of approximation lower bound and an  $O(\log(n) + \log(r))$  approximation algorithm for both the arbitrary cost and uniform cost Network Construction problems. If  $r$  is polynomial in  $n$  these bounds match, but otherwise there is a gap. We also have a  $\log(n)$  asymptotic gap for the Online Network Construction problem. For the Online Uniform Cost Network Construction problem, we have an  $\Omega(\sqrt{n})$  adversarial lower bound and an  $O(n^{2/3} / \log^{1/3}(n))$  algorithm for the oblivious case. Improving these bounds is an important problem.

Another open problem is to find tight bounds for trees in the uniform cost case. For stars and paths, the bounds are tight, and our arguments can be adapted to give a  $\Omega(\log(n))$ -competitive lower bounds against an oblivious adversary. Perhaps an  $O(\log(n))$ -competitive algorithm can be found for trees in general, but our algorithms for paths and stars rely on their specific properties and do not immediately generalize. Finally, one can consider generalizations of the Network Construction Problem, for example constraints can require the vertices to be  $k$ -connected in the induced subgraphs.

## Acknowledgments

We thank Kevin Chang, David Eisenstat, Khaled Elbassioni, Rajiv Raman, and Danny Segev for helpful discussions.

## References

1. ALON, N., AND ASODI, V. Learning a hidden subgraph. *SIAM J. Discrete Math.* 18, 4 (2005), 697–712.
2. ALON, N., AWERBUCH, B., AZAR, Y., BUCHBINDER, N., AND NAOR, J. The online set cover problem. In *STOC* (2003), pp. 100–105.
3. ALON, N., AWERBUCH, B., AZAR, Y., BUCHBINDER, N., AND NAOR, J. A general approach to online network optimization problems. *ACM Transactions on Algorithms* 2, 4 (2006), 640–660.
4. ALON, N., BEIGEL, R., KASIF, S., RUDICH, S., AND SUDAKOV, B. Learning a hidden matching. *SIAM J. Comput.* 33, 2 (2004), 487–501.
5. ANGLUIN, D., ASPNES, J., AND REYZIN, L. Optimally learning social networks with activations and suppressions. In *ALT* (2008), pp. 272–286.
6. ANGLUIN, D., AND CHEN, J. Learning a hidden graph using  $O(\log n)$  queries per edge. *J. Comput. Syst. Sci.* 74, 4 (2008), 546–556.
7. BEIGEL, R., ALON, N., KASIF, S., APAYDIN, M. S., AND FORTNOW, L. An optimal procedure for gap closing in whole genome shotgun sequencing. In *RECOMB* (2001), pp. 22–30.
8. BOOTH, K. S., AND LUEKER, G. S. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.* 13, 3 (1976), 335–379.
9. BUCHBINDER, N. *Designing Competitive Online Algorithms Via A Primal-Dual Approach*. PhD thesis, Technion – Israel Institute of Technology, Haifa, Israel, 2008.
10. ERDÖS, P., AND RÉNYI, A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.* 5 (1960), 17–61.
11. FEIGE, U. A threshold of  $\ln$  for approximating set cover. *J. ACM* 45, 4 (1998), 634–652.
12. GREBINSKI, V., AND KUCHEROV, G. Reconstructing a Hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discrete Applied Mathematics* 88, 1-3 (1998), 147–165.
13. GUPTA, A., KRISHNASWAMY, R., AND RAVI, R. Online and stochastic survivable network design. In *STOC* (2009), pp. 685–694.
14. KORACH, E., AND STERN, M. The clustering matroid and the optimal clustering tree. *Mathematical Programming* 98, 1-3 (2003), 345–414.
15. KORACH, E., AND STERN, M. The complete optimal stars-clustering-tree problem. *Discrete Applied Mathematics* 156, 4 (2008), 444–450.
16. REYZIN, L., AND SRIVASTAVA, N. Learning and verifying graphs using queries with a focus on edge counting. In *ALT* (2007), pp. 285–297.

## APPENDIX

### A Updating a pq-tree

We briefly describe the patterns in [8] for updating pq-trees, as broken down into 10 cases.<sup>2</sup> This can be used as a guide for tracking the changes in Equation 2.

- L This pattern simple relabels some leaf nodes.
- P1 This pattern simply relabels a p-node.
- P2 This pattern moves some children of a p-node into their own p-node.
- P3 This pattern moves some children of a p-node into their own p-node and creates a parent q-node.
- P4 This pattern moves some children of a p-node to be children of a newly created p-node, whose parent is a q-node that is a child of the original p-node.
- P5 This pattern moves some children of a p-node into their own p-node that is the child of the original p-node, which becomes transformed to a q-node.
- P6 This pattern moves some children of a p-node to their own p-node that is moved to be the child of a newly created q-node formed by merging two q-nodes.
- Q1 This pattern simply relabels a q-node.
- Q2 This pattern deletes a q-node and moves its children to become children of its parent q-node.
- Q3 This pattern deletes two q-nodes and merges their children to become children of their parent q-node.

### B Analysis of Algorithm 1

**Lemma 3.** *Algorithm 1 gives an  $O(\log(n))$  approximation to the **Online Fractional Network Construction** problem.*

*Proof.* We say that the optimal solution OPT has cost  $\alpha$ . We assume the value of  $\alpha$  is known, and we can then assume all edges have cost between 1 and  $m$ .<sup>3</sup> We now follow the argument in Alon et al. [3], which works for Algorithm 1 almost without modification. First we note that the algorithm generates a feasible solution. This is clear from its termination condition.

Now we will prove that the number of weight augmentation steps performed during the run of the algorithm is  $O(\alpha \log(m))$ . Consider the potential function

$$\Phi = \sum_{e \in E} c_e w_e^* \lg(w_e),$$

---

<sup>2</sup> A helpful guide to understanding the pq-tree update patterns appears on [knol.google.com/k/gregory-grothaus/pq-trees-and-the-consecutive-ones/](https://knol.google.com/k/gregory-grothaus/pq-trees-and-the-consecutive-ones/).

<sup>3</sup> Alon et al. [3] argue that we can use all edges of cost less than  $\alpha/m$  and stay within our bound, and we can ignore all edges with cost greater than  $\alpha$ , and then rescale. They also show how to guess  $\alpha$  to within a factor of 2, justifying the assumption that  $\alpha$  is known in advance.

where  $w_e^*$  is the weight of edge  $e$  in OPT. It is clear from the initial edge weights that the potential function begins as  $\Phi_0 = -O(\alpha \lg(m))$ . Because no edge gets weight more than 2, the potential function never exceeds  $2\alpha$ . And the increase in the potential function with each weight augmentation step is at least 1:

$$\begin{aligned}
\Delta\Phi &= \sum_{e \in E} c_e w_e^* \lg(w_e(1 + 1/c_e)) - \sum_{e \in E} c_e w_e^* \lg(w_e) \\
&= \sum_{e \in E} c_e w_e^* \lg(1 + 1/c_e) \\
&\geq \sum_{e \in E} w_e^* \\
&\geq 1.
\end{aligned}$$

Finally, we look at the cost of our solution,  $\sum_{e \in E} w_e c_e$ , (which begins at  $\leq 1$ ) and notice that in a weight augmentation step, it does not exceed  $\sum_{e \in E} \frac{w_e}{c_e} c_e \leq 1$ . So, whenever  $\Phi$  increases by at least 1, the cost of our solution increases by no more than 1. This gives us an  $O(\log(m)) = O(\log(n))$  approximation to the Online Fractional Network Construction problem.  $\square$