Network Construction with Subgraph Connectivity Constraints

Dana Angluin · James Aspnes · Lev Reyzin

Abstract We consider the problem introduced by Korach and Stern in [17] 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 [18]: 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 also 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.

Keywords network optimization · subgraph constraints · connectivity

D. Angluin
Department of Computer Science, Yale University
51 Prospect St., New Haven, CT 06511,
E-mail: dana.angluin@yale.edu

J. Aspnes
Department of Computer Science, Yale University
51 Prospect St., New Haven, CT 06511
E-mail: james.aspnes@yale.edu

L. Reyzin
Department of Mathematics, Statistics, & Computer Science, University of Illinois at Chicago
851 South Morgan Street, Chicago, IL 60607
E-mail: lreyzin@math.uic.edu

1 Introduction

We consider the following problem introduced by Korach and Stern in [17] – 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 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 [17] 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 [18] 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 answer 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 [16] 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, 7, 8, 15, 20]. 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[1] 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 probability less than $1/2$, 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. Papers studying this and similar problems from the learning perspective include a conference version of our results [6], as well as works by Saito et al. [21] and Gomez-Rodriguez et al. [14].

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 $S = \{S_1, S_2, \ldots, S_r\}$ 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 class of graphs, e.g. trees, we mean that all constraints S_i can be satisfied in an optimal solution (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 $O(\log r)$ approximation, where r is the number of constraints. This matches the lower bound when r is polynomial in n. We note that Chockler et al. [11], in the context of studying overlay networks, independently give an NP-hardness bound for this problem. Ours is a hardness of approximation result.

---

[1] For more on CDC statistics, we direct the reader to `www.cdc.gov/datastatistics/`.

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 that any algorithm must have an $\Omega(\log n)$-competitive ratio, and we 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 offline Uniform Cost Network Construction problem.

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 $U = \{v_1, v_2, \ldots, v_n\}$ and $\{C_1, C_2, \ldots, C_j\}$ with $C_i \subseteq U$. The **Hitting Set** problem is to minimize $|H|$, where $H \subseteq U$ such that $\forall C_i$, $H \cap C_i \neq \emptyset$. We define an instance of the Uniform Cost Network Construction problem with n^3 by n vertices $v_{(i,j)}$, for all $1 \leq i \leq n^3$ (rows) and $1 \leq j \leq n$ (columns). For each i, the vertices in row i, $\{v_{(i,1)}, v_{(i,2)}, \ldots, v_{(i,n)}\}$, correspond to the elements $\{v_1, \ldots, v_n\}$ 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 $\binom{n^3}{2}$ connectivity constraints. Without loss of generality, let $C_i = \{v_1, v_2, \ldots, v_k\}$. For each pair $l \neq j$ such that $1 \leq l, j \leq n^3$ we add a connectivity constraint

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

in the Uniform Cost Network Construction problem. This enforces the Hitting Set constraints pairwise between the n^3 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 $1, \ldots, k$. 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 $\{v_{(l,x)}, v_{(j,y)}\}$ 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 may choose 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 $\Omega(\log n)$ [13,19], and there are $\Theta\binom{n^3}{2}$ pairs of Hitting Set instances (or rows). The optimal solution has $\left(n^3\binom{n}{2} + \text{OPT}\binom{n^3}{2}\right)$ edges – the first term counts the pairwise constraints in each row. So unless P=NP, the best polynomial time algorithm will require $\left(n^3\binom{n}{2} + \Omega\left(\log n\text{OPT}\binom{n^3}{2}\right)\right)$ edges, giving us the result. $\qquad\square$

Below, we give an algorithm that almost meets this lower bound, even in the arbitrary cost case when r is polynomial in n.

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

Proof The inputs are the vertices $V = \{v_1, v_2, \ldots, v_n\}$, the cost c_e of each edge $e = (v_i, v_j)$, and the constraints $\{S_1, S_2, \ldots, S_r\}$. Let $C(E)$ be a potential function that takes in a set of edges and sums over all constraints S_i, 1 minus the number of components S_i induces on (V, E). Let E be initially empty. Now, consider the following greedy algorithm: until all constraints are satisfied (while $C(E) < 0$), greedily add to E the edge that is $\arg\max_e \frac{C(E+e)-C(E)}{c_e}$.

We now notice that $C(E)$ is sub-modular in its edge set – as in, if $A \subseteq B$ then for all e, $C(A + e) - C(A) \geq C(B + e) - C(B)$. This is clear because A can induce additional components for e to reduce compared to B. Because each edge can increase the value of C by at most r, we can use Theorem 3 to get an approximation of $O(\log r)$. $\qquad\square$

Theorem 3 (Wolsey [22]) *Given the problem*

$$\min\left\{\sum_{j \in S} x_j : f(S) = f(N), S \subseteq N\right\},$$

where f is a nondecreasing submodular set function on a finite set N, with $f(\emptyset) = 0$. The value of the greedy heuristic never exceeds the optimal value by more than a factor

$$H(\max_x f(\{x\})),$$

where $H(m) = \sum_i^m \left(\frac{1}{i}\right)$.

3 Online Network Construction

Oftentimes, the algorithm must commit to its choices as constraints arrive. For example, when the constraints represent diseases, we may wish to commit resources to fight an epidemic. This leads us to consider the natural extension of network construction to the online setting.

In the online setting, the collection of connectivity constraints $S_1, S_2, \ldots, S_r$ 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 that the underlying graph is, for instance, a uniform cost path or star, we know that 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 for both the arbitrary and uniform cost cases, 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_{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_{lazy} does not, A_{lazy} remembers as possible edges for future rounds and adds them as needed to satisfy future constraints. It is clear that A_{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_{lazy} satisfies the constraints, and because A_{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 4 *The optimal competitive ratio for the **Online Uniform Cost Network Construction** problem on n nodes when the algorithm knows that an optimal solution forms a **star** graph 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 $O\left(\frac{|R||D_{i-1}|}{|C_{i-1}|}\right)$ edges. This clearly satisfies constraint S_i.

To see why this gives us the needed result, we notice that 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 $\sum_i O\left(\frac{|R||D_{i-1}|}{|C_{i-1}|}\right) \leq \sum_i O\left(\frac{|R_i|n}{|C_{i-1}|}\right)$. 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 structure – when the underlying graph is a path.

Theorem 5 *The optimal competitive ratio for the **Online Uniform Cost Network Construction** problem on n nodes when the algorithm knows an optimal solution forms a **path** graph 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 [9], 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

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

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.)

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

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 [9] characterized all of the possible updates to the pq-tree using 10 patterns: L, P1, P2, P3, P4, P5, P6, Q1, Q2, and Q3, given in the Appendix. 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 Φ.   □

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

Theorem 6 *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 of the 1 edges. However, 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. □

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 $\{S_1, S_2, \ldots, S_r\}$. 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. Algorithm 1 is a modification of the algorithm in **3.1** of Alon *et al.* [3], and this proof closely follows their logic.

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 say that the optimal solution OPT has cost α. We assume the value of α is known, and we can then assume all edges have cost between 1 and m.[2] 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

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

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 the weight update rule ensures that no edge gets weight more than 2, the potential function never exceeds α. The increase in the potential function with each weight augmentation step is also at least 1:

$$\begin{aligned}
\Delta\Phi &= \sum_{e \in E} c_e w_e^* \lg(w_e(1 + 1/c_e)) - \sum_{e \in E} c_e w_e^* \lg(w_e) \\
&= \sum_{e \in E} c_e w_e^* \lg(1 + 1/c_e) \\
&\geq \sum_{e \in E} w_e^* \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad (3) \\
&\geq 1.
\end{aligned}$$

To derive Equation 3, we observe that for $x \geq 1$, $2^x - x \geq 1$, which implies $x \lg(1 + 1/x) \geq 1$.

Finally, we look at the cost of our solution, $\sum_{e \in E} w_e c_e$, (which begins at ≤ 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 Φ increases by at least 1, the cost of our solution increases by no more than 1. This finishes the proof.   □

We can now use Lemma 2 to develop an algorithm that almost matches the lower bound from Theorem 6.

Theorem 7 *There is an $O(n \log n)$-competitive polynomial time algorithm for the **Online Network Construction** problem on n nodes.*

Proof We use Algorithm 1 together with a rounding scheme similar to the one considered by Buchbinder [10] 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.

---

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

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 and integer i, 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 $X(e,i) < w_e$ – we call this event A_i. Hence, the probability of $\cup_{i=1}^{2n} A_i$ is at most $2nw_e$, and by linearity of expectation, on every round, the expected cost of our solution is $O(n)$ times that of the fractional solution, which is an $O(\log n)$ approximation of OPT for the integral problem. 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 $\prod_{e \in C} (1 - w_e)^{2n} \leq e^{\left(-2n \sum_{e \in C} w_e\right)}$. 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 $\left(\frac{r2^n}{e^{2n}}\right)$. Because $r \leq 2^n < e^n$, the probability our solution is not feasible tends to 0 as n increases, completing the proof. $\qquad\square$

Proof (Alternate Algorithm)

We now give an alternate proof of the theorem by reducing the Online Network Construction problem to Online Set Cover and using an algorithm for the latter problem. In Online Set Cover, $X = \{1, 2, \ldots, n\}$ is a set of n elements and S is a family of m weighted subsets of X. S is given to the algorithm in advance, and elements of $X' \subseteq X$ arrive one at a time in arbitrary order online. While X is known to the algorithm, X' is not. The goal of the algorithm is to select a collection of sets from S of lowest weight, such that at any point in the algorithm, every element that has arrived is contained in some selected subset. Once a subset is selected, it cannot be unselected.

In reducing the Online Network Construction problem to Online Set Cover, we make the weighted edges correspond to the weighted sets given to the algorithm. For each constraint to arrive online, we make a set cover element for each possible cut through the constraint. A set covers an element if its corresponding edge crosses the cut corresponding to the element. In the Online Network Construction problem, each cut in a constraint must be crossed by some edge, and if each cut is crossed, the constraint is satisfied. The cost of the optimal solution to both problems is the same.

Hence, if the original Online Network Construction problem has n nodes, then the Online Set Cover instance has $O(n^2)$ sets and $O(2^n)$ possible elements (or partitions of vertices). Alon *et al.* [2] and Buchbinder [10] provide algorithms for Online Set Cover that give an $O(\log(m) \log(n))$-competitive ratio if m is the number of sets and n is the number of elements. For the Online Network Construction problem, this gives an $O(n \log n)$-competitive bound, completing the proof. $\qquad\square$

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. □

We also present a lower bound for the online uniform cost case.

Theorem 8 *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 $v \in R$, the adversary does runs through t stages, presenting one constraint per stage. At stage 1 the adversary sets $Q_{(v,1)} = Q$. At stage $t \geq 1$, the adversary gives the algorithm the constraint $S_{(v,t)} = Q_{(v,t)} \cup v$. Let $C_{(v,t)}$ be the set of vertices in Q which the algorithm connects to v in response to being presented $S_{(v,t)}$. The adversary sets $Q_{(v,t+1)} = Q_{(v,t)} \setminus C_{(v,t)}$ and continues to the next stage. The adversary stops when $Q_{(v,t)} = \emptyset$.

To analyze this strategy for the adversary, for each v, we order the edges from v to R by the stage in which the algorithm has placed them, breaking ties arbitrarily. It is clear that the last edge the algorithm places is sufficient to connect v to R for all constraints $S_{(v,t)}$. 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. The algorithm, however, places $\Omega(n)$ edges per vertex in Q, amounting to $\Omega(n\sqrt{n})$ edges in total, giving the desired result. □

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

Theorem 9 *There is a randomized polynomial time algorithm for the **Online Uniform Cost Network Construction** problem on n nodes and $r = \text{poly}(n)$ constraints that gives an expected $O(n^{2/3} \log^{2/3} n)$-competitive ratio against an oblivious adversary.*

Proof We assume that that each vertex appears in some constraint (hence, an optimal solution has $m = \Omega(n)$ edges). We then create an Erdös 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 $c > 1$ [12]. 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, 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, if and only if

it is not already satisfied, we 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 the optimal solution. Hence, this will ensure we place no more than OPT cliques in total. Altogether, this costs us $O(n^{5/3} \log^{2/3} n + \text{OPT } n^{2/3} \log^{2/3} n)$ edges in expectation, and because $m = \Omega(n)$, we have an $O(n^{2/3} \log^{2/3} n)$ approximation ratio. $\square$

4 Discussion and 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 $O(\log r)$ 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 can be 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 could require the vertices to be k-connected in the induced subgraphs.

5 Acknowledgments

References

1. Alon, N., Asodi, V.: Learning a hidden subgraph. SIAM J. Discrete Math. **18**(4), 697–712 (2005)
2. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: The online set cover problem. In: Proceedings of the 35th annual ACM symposium on Theory of computing, pp. 100–105 (2003)
3. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: A general approach to online network optimization problems. ACM Transactions on Algorithms **2**(4), 640–660 (2006)
4. Alon, N., Beigel, R., Kasif, S., Rudich, S., Sudakov, B.: Learning a hidden matching. SIAM J. Comput. **33**(2), 487–501 (2004)
5. Angluin, D., Aspnes, J., Reyzin, L.: Optimally learning social networks with activations and suppressions. In: 19th International Conference on Algorithmic Learning Theory, pp. 272–286 (2008)
6. Angluin, D., Aspnes, J., Reyzin, L.: Inferring social networks from outbreaks. In: 21st International Conference on Algorithmic Learning Theory, pp. 104–118 (2010)
7. Angluin, D., Chen, J.: Learning a hidden graph using $O(\log n)$ queries per edge. J. Comput. Syst. Sci. **74**(4), 546–556 (2008)
8. Beigel, R., Alon, N., Kasif, S., Apaydin, M.S., Fortnow, L.: An optimal procedure for gap closing in whole genome shotgun sequencing. In: RECOMB, pp. 22–30 (2001)
9. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. Syst. Sci. **13**(3), 335–379 (1976)
10. Buchbinder, N.: Designing competitive online algorithms via a primal-dual approach. Ph.D. thesis, Technion – Israel Institute of Technology, Haifa, Israel (2008)
11. Chockler, G., Melamed, R., Tock, Y., Vitenberg, R.: Constructing scalable overlays for pub-sub with many topics. In: PODC, pp. 109–118 (2007)
12. Erdös, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci **5**, 17–61 (1960)
13. Feige, U.: A threshold of ln for approximating set cover. J. ACM **45**(4), 634–652 (1998)
14. Gomez-Rodriguez, M., Leskovec, J., Krause, A.: Inferring networks of diffusion and influence. TKDD **5**(4), 21 (2012)
15. Grebinski, V., Kucherov, G.: Reconstructing a Hamiltonian cycle by querying the graph: Application to DNA physical mapping. Discrete Applied Mathematics **88**(1-3), 147–165 (1998)
16. Gupta, A., Krishnaswamy, R., Ravi, R.: Online and stochastic survivable network design. In: 41st Symposium on the Theory of Computing, pp. 685–694 (2009)
17. Korach, E., Stern, M.: The clustering matroid and the optimal clustering tree. Mathematical Programming **98**(1-3), 345–414 (2003)
18. Korach, E., Stern, M.: The complete optimal stars-clustering-tree problem. Discrete Applied Mathematics **156**(4), 444–450 (2008)
19. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In: STOC, pp. 475–484 (1997)
20. Reyzin, L., Srivastava, N.: Learning and verifying graphs using queries with a focus on edge counting. In: 18th International Conference on Algorithmic Learning Theory, pp. 285–297 (2007)
21. Saito, K., Nakano, R., Kimura, M.: Prediction of information diffusion probabilities for independent cascade model. In: KES (3), pp. 67–75 (2008)
22. Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica **2**(4), 385–393 (1982)

Appendix: Updating a pq-tree

We briefly describe the patterns in [9] for updating pq-trees, as broken down into 10 cases. This can be used as a guide for tracking the changes in Equation 2.

  L This pattern simply 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.