Algorithms on Evolving Graphs

Aris Anagnostopoulos
Sapienza University
Rome, Italy
aris@dis.uniroma1.it

Ravi Kumar
Yahoo! Research
Sunnyvale, CA, USA
ravikumar@yahoo-inc.com

Mohammad Mahdian
Yahoo! Research
Sunnyvale, CA, USA
mahdian@yahoo-inc.com

Eli Upfal
Brown University
Providence, RI, USA
eli@cs.brown.edu

Fabio Vandin
Brown University
Providence, RI, USA
vandinfa@cs.brown.edu

ABSTRACT

Motivated by applications that concern graphs that are evolving and massive in nature, we define a new general framework for computing with such graphs. In our framework, the graph changes over time and an algorithm can only track these changes by explicitly probing the graph. This framework captures the inherent tradeoff between the complexity of maintaining an up-to-date view of the graph and the quality of results computed with the available view. We apply this framework to two classical graph connectivity problems, namely, path connectivity and minimum spanning trees, and obtain efficient algorithms.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computation on discrete structures*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Probabilistic computation*

General Terms

Theory

Keywords

evolving graphs; algorithms; path connectivity; minimum spanning tree;

1. INTRODUCTION

Graphs are ubiquitous in today's world. Such graphs range from social networks to recommendation networks to communication and information networks to hyperlink-induced web graphs. Two characteristics common to all of these graphs make computing with them daunting: their evolving and decentralized nature and, often, their scale. The classical computational paradigm, which assumes a fixed data as an input to an algorithm that terminates, is insufficient for many modern data processing needs. Consequently, computational and data models that depart from the conventional paradigm have been proposed: the data stream model where the input is given as a stream and the algorithm does not have access to enough memory to store the data; the dynamic model where the input changes and the algorithm, aware of these changes, must be faster than a naive recomputation; the property testing model where the algorithm needs to produce the answer after probing a small portion of the input; and so on.

Unfortunately, none of these models by itself is sufficient to address all the challenges posed by evolving and massive graphs. For instance, the dynamic and the data streaming models assume knowledge of all the changes in the data, thereby imposing severe restrictions on their capabilities. The property testing model does not make this assumption, but it only works on static (i.e., not evolving) input.

Our contributions. To address some of the challenges of interest that are not captured by the aforementioned models, we propose a new computational framework on graphs where the underlying graph changes over time and an algorithm can only observe these changes by probing the graph in a limited way. This evolving graph framework is inspired by the work of Anagnostopoulos et al. [2], who studied sorting and selection problems in an evolving (scalar) data setting. In our framework, the graph changes over time without notifying the algorithm and therefore the algorithm needs to periodically probe the graph to learn about new changes and update its solution accordingly. The fact that the algorithm is *unaware* of the changes in the input makes this framework different from the previous work on dynamic algorithms, and suitable for modeling real-world settings such as the following: the computations of search engines on the web graph, where the web graph is continually changing and the search engine can only learn about these changes by periodically crawling the web pages; the computation by a third-party vendor on social networks such as Twitter or Facebook, where the network is constantly evolving and the vendor can access the network only through a rate-limited oracle (i.e., an API).

We focus on two basic graph connectivity questions in this framework—path connectivity and finding minimum spanning trees (MST). In the path connectivity problem, the set of edges of the graph changes in time, and we obtain an algorithm that outputs an invalid path at each time with probability $O((\log n)/n)$; the corresponding lower bound is $\Omega\left(\frac{\log n}{n(\log\log n)^2}\right)$. In the weighted MST problem, the order of weights between the edges changes in time and we obtain an algorithm that outputs a tree with sum of ordinal weights that is within distance $O(\log^2 n)$ from the optimal; we generalize our MST algorithm to general matroid settings, although with a loss in the bound achievable by the algorithm. The positive results we obtain for these problems suggest that our framework, while capturing the evolving nature of graphs, remains amenable to algorithmic development. We also believe that our techniques would be useful in more general settings of graph evolution.

1.1 Related Work

Models for dealing with dynamic graphs have been extensively studied in the algorithmic community from various points of view. However, none of these models captures the three crucial aspects of the scenario depicted earlier: the slow evolution of the underlying graph, learning about the changes only through probing a limited portion of the graph, and the observation of the system for an infinite (or very long) amount of time. We now discuss some of the models most related to ours.[1]

(i) *Dynamic graph model [5].* In this setting, a graph changes over time and the goal is to keep track of the changes so as to be able to efficiently answer graph queries. The main difference is that here when a change is performed to a graph, the algorithm is notified of the change; instead in our setting the algorithm does not know the change but it has to perform queries to learn it. In addition, the expensive resource in our setting is the number of queries, while we do not pose any additional restrictions on the time/space complexity of the algorithms.

(ii) *Data streams [8].* Here the algorithm observes a sequence of events (e.g., edge addition or deletion) and has to maintain an approximate solution. Computational resources, typically space, are limited and the algorithm should maintain an approximate solution under these limitations, while being able to observe the entire stream of changes. In contrast, our limited resource is the number of data probes.

(iii) *Property testing [10, 11].* Here the goal is to find whether a graph has some property or is far from satisfying the property using a limited number of queries. Unlike our setting, this model is static. Furthermore, in our model the underlying problem does not need to be a decision problem and there are no restrictions placed on the input (such that if the structure does not satisfy the property then it is far from satisfying it).

(iv) *Multi-armed–bandit model [7].* In the standard multi-armed bandit setting there are k slot machines (one-armed bandits) and pulling a lever in a slot machine gives a reward, which depends on the machine, and reveals information about the machine. The objective is to select the machines to query so as to maximize the total reward; as in our case, the number of queries in every time step is limited. In particular, similar tradeoffs with those in our setting are studied in [13], where the distribution of the rewards changes over time. The setting studied in the current paper has a different goal (producing the correct answer every time, as opposed to maximizing the reward) and works on an underlying graph structure.

(v) *Parametric optimization and kinetic problems [3, 1].* In the parametric optimization model, the edge weights are known continuous functions of a real parameter λ (often referred to as "time"), and the goal is to identify how the solution changes as λ varies. A kinetic problem combines parametric optimizations and dynamic data structures for insertions and deletions. In such a problem, at the beginning a parametric problem of parameter λ is given; as the time λ progresses, the weight functions change and objects (e.g., edges) are inserted or deleted. The goal is to efficiently maintain the optimal solution at each point in time. For both settings the main difference with our model is that at each point in time the algorithm is notified of the changes, while in our setting probes are required to learn the changes. Moreover, the resource of interest in our model is the number of probes, without additional restrictions on the time/space complexity of the algorithms.

2. MODEL

We follow the general framework defined in [2] for algorithms on dynamic data. In the case of evolving graphs, this model can be described as follows. The time is assumed to proceed in discrete steps, numbered by positive integers. At each time step t, the data is given by a (possibly weighted) graph G_t. The data is changing gradually, i.e., the graph G_{t+1} is obtained from G_t by a small random *change*.[2] At each time step t, the algorithm is allowed to *probe* a small portion of the graph G_t, and then must output a solution for the problem under consideration. We would like this solution to be *close* to the "correct" solution for the graph G_t. In this paper we do not impose any constraint on the amount of memory the algorithm maintains or the running time of the algorithm, although all of the algorithms we present are quite efficient with respect to these factors.

To complete the definition of the model for a specific problem, we need to fix three aspects: how the graph evolves, what types of probe the algorithm is allowed to perform, and how we measure the quality of the solution.

(i) *Change.* To keep the model simple, we assume that the number of nodes and edges of the graph and also the labels of the nodes remain fixed. Let V denote the node set of G_t, E_t denote the edge set of G_t, $n = |V|$, and $m = |E_t|$. The manner in which the graph evolves depends on whether the graph is unweighted or weighted.

For unweighted graphs, the edges change by a random swap, i.e., by removing a random edge of the graph and adding an edge between a random pair of nodes. More formally, let $(u, v) \in E_t$ be a uniformly chosen edge and let $u', v' \in V$ be uniformly chosen nodes such that $(u', v') \notin E_t \setminus (u, v)$. In words, (u', v') is a randomly selected *nonedge*—pair of nodes that are not connected. Then, $E_{t+1} = (E_t \setminus \{(u, v)\}) \cup \{(u', v')\}$.

For weighted graphs, we assume an ordinal model of change similar to the one in [2]: the edge set is fixed (i.e.,

[1]As mentioned earlier, our evolving graph framework is inspired by the model in [2], which dealt with scalar data.

[2]It is easy to show trivial lower bounds if the changes are adversarial; thus, random changes are needed to make the model interesting.

$E_t = E$ for all t), and only the weights change. At each time step t, the weights of the edges induce an ordering π^t on the edge set E. The ordering π^{t+1} is obtained from π^t by swapping a random consecutive pair. As argued in [2], this is perhaps the most natural model for gradual change with ordinal weights, and as we will see, an ordinal weighted model is rich enough to capture the evolving MST problem.[3]

(ii) *Probe.* For unweighted problems, we allow the algorithm to perform a *node probe* in each time step, i.e., query a node $u \in V$ and learn about the set of neighbors of u.[4] For weighted graphs, we allow the algorithm to probe a pair of edges e, e' and observe which of these edges has a larger weight; this is the natural probing model for ordinal weights.

(iii) *Quality measure.* The benchmarks we use to measure the quality of the solution depends on the problem. In general, for optimization problems, this measure is an indication of how much the value of the optimal solution differs from the value produced by the algorithm. This is easy to define for unweighted problems, but requires some work (see Section 4) for ordinal weights. For non-optimization problems, we measure the quality of the solution by the probability that it is a valid feasible solution.

In the model described above we assumed that in each time step, the graph evolves by one random change and the algorithm can make one probe. A more general model is to allow the graph to change more rapidly than the probing rate of the algorithm (i.e., for every probe of the algorithm, the graph undergoes $\alpha \geq 1$ changes), or the other way around. Our evolving MST algorithm is presented in this more general model and our evolving connectivity algorithm can easily be generalized to constant α.

3. PATH CONNECTIVITY

In this section we consider the basic connectivity question in unweighted evolving graphs: given two fixed nodes $S, T \in V$, designated as the *source* and the *sink* respectively, maintain a path between S and T, assuming one exists. At each time t, the algorithm must output a path $P_t \subseteq V$ such that the probability that P_t is not a valid ST-path in G_t is negligible.

Note that as $t \to \infty$, the distribution of the graph G_t approaches the distribution of $\mathcal{G}(n,m)$, a graph chosen uniformly at random from all graphs with n nodes and m edges. Thus, if $m = o(n \ln n)$, then there is a non-trivial probability that there is no ST-path in the graph. Likewise, if $m = \omega(n^{3/2})$, then the problem becomes easy (either S and T are connected or they have some common neighbors, whp), and hence it is easy to discover an ST path. Therefore, the interesting range of parameters for this problem is when $m = \Omega(n \ln n)$ and $m = O(n^{3/2})$; we assume this[5]. For simplicity of exposition, we will assume that the graph $G_0 \sim \mathcal{G}(n,m)$ so that the graph $G_t \sim \mathcal{G}(n,m)$, and for convenience we assume that at time $t = 1$ our algorithm knows a valid ST-path (if at least one exists); otherwise our results hold for all t large enough. (In Appendix A, we show that the convergence to $\mathcal{G}(n,m)$ is fast and thus our results essentially hold with any starting configuration.)

We continue this section by giving a simple algorithm based on iteratively finding a path between S and T. We then improve upon this by keeping track of two disjoint paths and using the second path as the *emergency* output at times when the first path is found to be invalid. One might wonder if it is possible to improve even further by taking three or more disjoint paths. However, as we will show, the bound obtained by the two-path algorithm is asymptotically close to optimal. We finish by generalizing our results to multiple source-destination pairs.

3.1 The One-Path Algorithm

We start with a simple algorithm that works in phases: iteratively find a path between S and T, and at any moment, output the path that the last finished run of the path-finding algorithm has produced.

In each phase we execute the following path-finding algorithm. It first grows a ball B_S around S as follows. Initially, $B_S = \{S\}$ and S is marked *unvisited*. In every step, the algorithm picks an unvisited node in B_S closest to S, marks it as *visited*, and adds all its neighbors to B_S using a node probe; any new node added to B_S will be marked as unvisited. The algorithm stops after $R = \lceil \sqrt{c_0 n / \ln n} \rceil$ steps for a constant c_0, i.e., when B_S contains R visited nodes. Next, the algorithm grows another set B_T defined similarly, but starting from T instead of S. Again, this set is grown until there are R visited nodes in B_T. If at this point, the algorithm finds a node that is in common between B_S and B_T, it outputs the path defined through this node; otherwise, it declares that no path between S and T exists.

Intuitively, the graph should behave close to a $\mathcal{G}(n,m)$ random graph during the entire period that the algorithm is executed. In the next lemma we show that this is indeed the case, from which it follows (whp) the algorithm described manages to find a path between S and T.

LEMMA 1. *With probability at least $1 - O\left((n \ln n)^{-1/2}\right)$, there exists at least one path between S and T at the end of the execution of the algorithm, and the algorithm described succeeds in finding it.*

PROOF. First note that since $m = \Omega(n \ln n)$, by a Chernoff bound, at a given time point of the algorithm every node will have degree at least $c \ln n$ with probability at least $1 - n^{-4}$ for a sufficiently large constant c. By applying a union bound, we obtain that with probability at least $1 - n^{-3}$ each node will have degree at least $c \ln n$ during the entire algorithm execution. Similarly, the degree of each node is at most $O(m/n) = O(\sqrt{n})$ whp.

Fix a time point when the algorithm has constructed a set B_S. Since it stops after $R = \Theta(\sqrt{n/\ln n})$ steps, and since each degree is $O(\sqrt{n})$ whp., we have that whp.

$$|B_S| = O(R\sqrt{n}) = O(n/\sqrt{\ln n}).$$

In the next step a node $v \in B_S$ is selected and its neighbors are added into B_S. Whp. v has at least $c \ln n$ neighbors, distributed uniformly at random in V, and the expected number of neighbors that are not currently in B_S is at least

$$c \ln n \cdot \frac{n - O(n/\sqrt{\ln n})}{n}.$$

[3]Intuitively, this is because the MST can be computed using only pairwise comparisons of the weights of the edges.

[4]An alternative probing model is *edge probing*: given a pair of nodes u and v, the algorithm learns whether there is an edge between u and v or not. However, this model seems more appropriate in a regime where the graph G_t has a positive density (i.e., $m = \Theta(n^2)$). Many of our results can be adapted to this probing model for dense graphs.

[5]In particular we assume $m \geq cn \ln n$ for a constant $c > 1$.

By a Chernoff bound, with probability at least $1 - n^{-3}$, the number of neighbors not in B_S is at least $c' \ln n$, for some constant c'. Therefore, the set B_S will grow by $c' \ln n$ nodes. Hence, when the process finishes after R steps we have that $|B_S| \geq c'' R \ln n$, for some constant c''.

We apply the same analysis for the set B_T, with the only difference that we ignore the nodes in both B_S and B_T when we study the growth of B_T. The same analysis gives that the size of B_T is also whp. at least $\Omega(R \ln n)$. Note that the number of node probes made by the algorithm is $2R = O(\sqrt{n/\ln n})$.

We analyze the probability that the algorithm fails to find a path as follows: this probability is bounded by the probability that there is no edge between nodes in B_S and visited nodes in B_T. Whp., each visited node v in B_T has at least $c' \ln n$ neighbors, distributed uniformly, thus the probability that no edge exists between v and B_S is upper bounded by

$$\left(1 - \frac{c'' R \ln n}{n}\right)^{c' \ln n} \leq e^{-c'c'' R \ln^2 n / n}.$$

Hence, the probability that no edge exists between a visited node in B_T and B_S is at most

$$e^{-c'c'' R^2 \ln^2 n / n} = e^{-c'c'' \ln n}.$$

Therefore, with high probability the algorithm succeeds in finding a plausible path (we call it plausible since some of its edges might have been removed during the execution of the algorithm, an event that as we show next is highly improbable).

The probability that the path computed at the end of the execution (call this time t) is still valid is at least the probability that none of the edges of the path is removed at any time step after it was observed by the algorithm until time t. There are at most $2R$ such time steps. Also, notice that B_S and B_T have both diameter $O(\ln n)$, thus the length of the path is at most $O(\ln n)$. Therefore, the probability of getting an invalid path at time t is at most the probability that at least one of the $O(2R \ln n) = O(\sqrt{n \ln n})$ bad events of the form "edge e is removed at time x" occurs. The probability of each such event is $1/m$, and hence by the union bound, the probability of getting an invalid path at time t is at most $O(\sqrt{n \ln n}/m) = O((n \ln n)^{-1/2})$. $\square$

THEOREM 2. *There is an algorithm for ST-path connectivity on evolving graphs that guarantees that for every t, the probability that the path output by the algorithm at time t is invalid is at most $O((n \ln n)^{-1/2})$.*

PROOF. We use the following algorithm: the time is divided into *phases* of length $2R$. The ith phase starts at time $t = 2Ri+1$ and ends at time $2R(i+1)$. In each phase, we run the algorithm that we described previously to find a path between S and T, and at any time step $t \in [2Ri+1, 2R(i+1)]$, we output the path computed in the last phase, that is, at time $2Ri$. This completes the description of the algorithm.

Now, consider a time step $t \in [2Ri+1, 2R(i+1)]$. By Lemma 1, at time $2Ri+1$ the algorithm succeeds in outputting a valid path whp. We now analyze the probability that at this time, the path output by the algorithm is invalid. We can apply the exact same analysis that we applied in the end of the proof of Lemma 1 and deduce that the probability that the path found at time step $2Ri+1$ is valid at time t is at least $1 - O((n \ln n)^{-1/2})$. $\square$

3.2 The Two-Path Algorithm

We now give a more sophisticated algorithm that achieves a guarantee better than the one in the previous section. The idea is to always keep track of two disjoint paths between S and T instead of one, using the first path as the primary solution and the second one as an emergency solution. We also need to monitor the primary path to discover when it becomes infeasible. We do this using a time-sharing trick: in even time steps, we run the algorithm for finding paths for the next round, while in odd steps, we probe the nodes of the current primary solution in a round-robin fashion to discover possible failures. We start by proving a statement analogous to Lemma 1.

LEMMA 3. *With probability at least $1 - O\left((n \ln n)^{-1/2}\right)$ there exists at least one path between S and T at the end of the execution and the algorithm described succeeds in finding it.*

PROOF. The algorithm is similar to the ball-growing algorithm of Lemma 1, except here to ensure finding two disjoint paths, we first pick two neighbors u_1 and u_2 of S and two neighbors v_1 and v_2 of T. Then, we grow four balls around each of the four nodes u_1, u_2, v_1, v_2. Each ball is grown for R steps (i.e., we have R visited nodes in each ball), and we keep the balls around u_2 and v_2 disjoint from the balls around u_1 and v_1 by rejecting any neighbor that is already included in those balls.[6] Since whp. the degree of each node is $\sqrt{n}$, there at most $O(R\sqrt{n}) = O(n/\sqrt{\ln n})$ nodes that if seen would be rejected. Therefore, arguing as in the proof of Lemma 1, at the end of the ball-growing process, each ball contains $\Omega(R \ln n)$ nodes. Therefore, the argument in the proof of Lemma 1 shows that with high probability, the ball B_{u_1} intersects with the ball B_{v_1} and the ball B_{u_2} intersects with the ball B_{v_2}. These intersections define two edge-disjoint (in fact, internally node-disjoint) paths between S and T. The number of probes this algorithm performs is precisely $4R = O(\sqrt{n/\ln n})$. $\square$

THEOREM 4. *There is an algorithm for ST-path connectivity on evolving graphs that guarantees that for every t, the probability that the path output by the algorithm at time t is invalid is at most $O(\frac{\ln n}{n})$.*

PROOF. As in the proof of Theorem 4, we divide the time into *phases*, in this case of length $8R$. The ith phase starts at time $t = 8Ri+1$ and ends at time $8R(i+1)$. In the even time steps during each phase, we run the algorithm from Lemma 3 to find two disjoint paths between S and T (pretending that the graph does not change while the algorithm is operating). We denote the two paths computed at the end of phase i by P_i (the primary path, e.g. the one going through u_1 and v_1) and P'_i (the emergency path, e.g. the one going through u_2 and v_2). In the odd time steps during phase i, the algorithm probes the nodes on the path P_{i-1} in a round-robin fashion to discover failures. The output during phase i, before any possible failure is detected is P_{i-1}, and after such a failure is detected, is P'_{i-1}. This completes the description of the algorithm.

Next, we analyze the probability that the path output by the algorithm at a fixed time step $t \in [8Ri+1, 8R(i+1)]$ is

[6]Here we need to initially add the nodes u_1, u_2, v_1, v_2 to the corresponding balls to avoid situations where one balls blocks the center of another ball.

invalid. This probability is the sum of the probability that P_{i-1} is output at time t and it is invalid, and the probability that P'_{i-1} is output at time t and it is invalid. Since P_{i-1} has length $O(\ln n)$ and is probed during the odd time steps of phase i, the only way it is invalid but still is chosen as the output is if one of its edges is removed during the last $O(\ln n)$ steps. Therefore, by the union bound, the probability that P_{i-1} is output at time t and it is invalid is bounded by

$$O\left(\frac{\ln^2 n}{m}\right) = O\left(\frac{\ln n}{n}\right).$$

The event that P'_{i-1} is output at time t and it is invalid occurs only if for both paths P_{i-1} and P'_{i-1} at least one edge of the path is removed in one of the time steps after this edge is observed by the path-finding algorithm of phase $i-1$ and before t. There are at most $16R$ such time steps, and each path has $O(\ln n)$ edges. Therefore, the probability of this event for each of the two paths is at most $O(16R\ln n/m) = O((n\ln n)^{-1/2})$. Since the two paths are disjoint, these two events are negatively correlated (in fact, conditioning on the edges of one path having been removed only slightly decreases the probability of the similar event for the second path). Therefore, the probability of the event that P'_{i-1} is output at time t and it is invalid is at most $O(((n\ln n)^{-1/2})^2)$.

Putting these two bounds together, we obtain that the probability that the algorithm outputs an invalid path at time t is at most

$$O\left(\frac{\ln n}{n} + \frac{1}{n\ln n}\right) = O\left(\frac{\ln n}{n}\right).$$

□

3.3 Lower Bound

One might wonder if a three-path algorithm would result in a similar improvement over the two-path algorithm. As we show, this is not the case, at least up to loglog factors. Intuitively, the reason for this is that after two paths, the time it takes to discover a new failure becomes the dominant factor in the overall probability of outputting an invalid path, and keeping track of more paths cannot help with this. We give a sketch of the lower bound proof in the following that contains the main ideas, although a complete formal proof requires careful use of the principle of deferred decisions.

THEOREM 5. *Assume $m = \Theta(n\ln n)$. For any ST-path connectivity algorithm on evolving graphs and any fixed t, the probability that the algorithm outputs an invalid path at time t is at least $\Omega\left(\frac{\ln n}{n(\ln\ln n)^2}\right)$.*

PROOF SKETCH. Given our assumptions, the graph G_t at time t is distributed according to $\mathcal{G}(n,m)$. Therefore, it is not hard to show that, with high probability, the length of the shortest path between S and T is at least $L = \Omega(\ln n/\ln\ln n)$. In particular, the length of the path P output by the algorithm at time t is at least L. Consider the time interval $[t-L/4, t]$. The algorithm can probe at least $L/4$ nodes in this time interval. Therefore, there are at least $L/2$ edges on P neither of whose endpoints is probed by the algorithm in this interval. For each such edge e and each time step x in $[t-L/4,t]$, there is a probability of $1/m$ that e is removed from the graph at time x. The probability that at least one of these $L^2/8$ events happen is at least

$$\Omega\left(\frac{L^2}{m} - \frac{L^4}{m^2}\right) = \Omega\left(\frac{\ln n}{n(\ln\ln n)^2}\right),$$

and if this event happens, the path output at time t will be invalid. □

3.4 Node-Disjoint Paths Between $O(\sqrt{n})$ Node Pairs

The ST-connectivity algorithm accesses $O(\sqrt{n})$ nodes in each phase. By constructing in parallel a number of node-disjoint balls we can extend the technique to maintaining $h = O(\sqrt{n})$ node-disjoint paths between a given collection of h disjoint pairs of nodes $\{(S_1,T_1),\ldots,(S_h,T_h)\}$, with an $O(h\frac{\ln n}{n})$ guarantee on the probability of producing an invalid answer. The details are presented in Appendix B.

4. MINIMUM SPANNING TREE

In this section we study the minimum spanning tree (MST) problem on evolving graphs. The problem is defined on a weighted complete graph in which the weights are changing as described in Section 2. In particular, this means that at time t the input G_t is the complete graph with n nodes and $m = \binom{n}{2}$ edges, and the weights of the edges induce a strict ordering π^t on the set E of edges $e_1 <_{\pi^t} \cdots <_{\pi^t} e_m$, where e_1 is the lowest weight edge. We assume that the ordering of edges changes gradually and we model this by assuming that for every $t > 1$, π^{t+1} is obtained from π^t by swapping α random pairs of consecutive elements. The analysis we present in this section can handle α as large as $O(n)$. As we will see this ordinal weighted model is rich enough to capture the MST problem on evolving graphs.

At each time step t we are given limited access to the real ordering π^t. In particular, at each time step t we can compare a pair of edges and obtain their relative ordering in π^t. We denote the rank of an edge e in an ordering π by $\pi(e)$ (e.g., the lowest weight edge has $\pi(e) = 1$). Finally, we assume that the initial order of the edges π^1 is random and that the algorithm knows π^1; otherwise our results hold for sufficiently large t (in particular, for $t = \Omega(m\ln m)$, the time required to sort the edges).

4.1 Measure of Approximation

Since we do not rely on the actual weights of edges, we need to define the notion of approximation for MST. Suppose that the MST is given by choosing edges $e_{i_1} <_{\pi^t} \cdots <_{\pi^t} e_{i_{n-1}}$. If our algorithm chooses edges $e_{i'_1} <_{\pi^t} \cdots <_{\pi^t} e_{i'_{n-1}}$, a natural metric would be to look at the correspondence between the indices $I = \{i_1,\ldots,i_{n-1}\}$ and $I' = \{i'_1,\ldots,i'_{n-1}\}$. A measure like the Jaccard distance[7] could be bad because $I \cap I'$ might be empty, but each index might be off just by 1. Because of the properties of spanning trees, we have the following proposition. Here recall that the edges of the graph are indexed such that $e_1 <_{\pi^t} \cdots <_{\pi^t} e_m$.

PROPOSITION 6. *The indices in I and I' are such that for every $1 \le \ell \le n-1$, we have $i_\ell \le i'_\ell$.*

PROOF. The proof is by contradiction. Assume that there exists a spanning tree of indices $I' = \{i'_1,\ldots,i'_{n-1}\}$ and a

[7]Given two sets A, B, their Jaccard distance is $\frac{|A\Delta B|}{|A\cup B|}$.

subset $S \subset \{1, \dots, n-1\}, S \neq \emptyset$ such that $\forall s \in S : i_s > i'_s$. In particular let r be the minimum of S. Note that $i'_r > i_{r-1}$, otherwise $i'_{r-1} < i'_r \leq i_{r-1}$ and r would not be the minimum of S. Now, consider edges $e_1 <_{\pi^t} \cdots <_{\pi^t} e_{i'_r - 1}$: $e_{i'_1} <_{\pi^t} \cdots <_{\pi^t} e_{i'_{r-1}}$ is a spanning forest for these edges (since the MST has $r-1$ edges in the same set). Since $i'_r < i_r$, the MST does not contain $e_{i'_r}$, therefore i'_r creates a cycle when added to $e_{i'_1} <_{\pi^t} \cdots <_{\pi^t} e_{i'_{r-1}}$, that is a contradiction. $\square$

This suggests $D = \sum_{\ell=1}^{n-1} (i'_\ell - i_\ell)$ as a natural ordinal measure to compare the solution I' against the optimal solution. This value ranges from 0 (when $I' = I$) to $O(nm) = O(n^3)$. Note that if L is the maximum difference between the weights of two edges that are adjacent according to π^t, then by Proposition 6, the total weight of the spanning tree produced by the algorithm is at most the weight of the MST plus DL. Therefore, a bound on the measure D will result in a bound on the total weight when the cardinal values of the weights are considered.

4.2 The Algorithm and Analysis

Our first algorithm is as follows: we use the simple sorting algorithm of [2] to keep an approximation $\tilde{\pi}^t$ of the real ordering π^t of all the $O(n^2)$ edges. At each time step, our first algorithm returns the tree built using Kruskal's greedy algorithm on $\tilde{\pi}^t$ (i.e., the MST for ordering $\tilde{\pi}^t$).

Before presenting the analysis, we need some notation: let T and T' denote the MST and the tree produced by the algorithm (i.e., the MST with respect to $\tilde{\pi}^t$), respectively. Recall that the indices of the edges in T and T' are denoted by I and I', respectively. Let $E_k = \{e_{i_1}, \dots, e_{i_k}\}$ be the first k edges in T, the MST according to order π^t (i.e., the first k edges chosen by Kruskal's algorithm), and $T_k = (V, E_k)$ denote the subgraph induced by these edges. Similarly, let $E'_k \triangleq \{e_{i'_1}, \dots, e_{i'_k}\}$ and $T'_k \triangleq (V, E'_k)$.

The following is a generalization of [2, Lemma 3], and its proof is similar to the one in [2].

Lemma 7. *For every edge e we have $|\pi^t(e) - \tilde{\pi}^t(e)| < c_1 \alpha \ln n$, in expectation and with high probability for some constant c_1.*

Let $\mathcal{E}(e)$ be the event "$|\pi^t(e) - \tilde{\pi}^t(e)| < c_1 \alpha \ln n$", and $\mathcal{E} = \cap_{e \in E} \mathcal{E}(e)$. By choosing the constant c_1 large enough, we have that the event $\mathcal{E}$ holds with high probability. Now, since the maximum error cannot be more than n^3, choosing c_1 large enough we have that the event $\neg\mathcal{E}$ only contributes a negligible amount to the expectation of measure D. Therefore, for ease of exposition, we condition on the event $\mathcal{E}$ in the analysis below.

A warmup bound. For any t, the permutation π^t corresponds to a random order of edges. Thus, with high probability the edges of the MST T are among the lightest $O(n \ln n)$ edges. The tree T' built by our algorithm will use some edges not in the T. Since the choice of an edge depends upon the choice of edges that precede it in the ordering, is it possible that our algorithm builds a tree of measure $D = \Theta(n^2)$? (Note that since $i_{n-1} = O(n \ln n)$, it is sufficient that $i'_{n-1} = \Theta(n^2)$ for this to happen.) In the next lemma we show that this is not the case:

Proposition 8. *At any time step t, we have that $D \leq 2c_1 \alpha n \ln n$ with high probability.*

Proof. For an edge e_{i_j} in the MST T, consider its position $\tilde{\pi}^t(e_{i_j})$ in $\tilde{\pi}^t$. From Lemma 7 we have

$$\tilde{\pi}^t(e_{i_j}) \leq i_j + c_1 \alpha \ln n$$

w.h.p for every edge e_{i_j}. Since T is a spanning tree, and T' is the MST for $\tilde{\pi}^t$, by Proposition 6 (which we apply after exchanging permutations π^t and $\tilde{\pi}^t$) we have $\tilde{\pi}^t(e_{i'_j}) \leq \tilde{\pi}^t(e_{i_j})$. Also, by Lemma 7, we have

$$\tilde{\pi}^t(e_{i'_j}) \geq \pi^t(e_{i'_j}) - c_1 \alpha \ln n = i'_j - c_1 \alpha \ln n.$$

Putting these three inequalities together, we obtain:

$$i'_j - c_1 \alpha \ln n \leq \tilde{\pi}^t(e_{i'_j}) \leq \tilde{\pi}^t(e_{i_j}) \leq i_j + c_1 \alpha \ln n.$$

Thus, with high probability,

$$D = \sum_{\ell=1}^{n-1} (i'_\ell - i_\ell) \leq 2c_1 \alpha n \ln n.$$

$\square$

An improved analysis. In the rest of this section we present a more careful analysis, showing that the error of the algorithm is much smaller than the one provided by Proposition 8, namely $O(\alpha^2 \ln^2 n)$. Since the proof is rather involved technically, with a few subtle points, we first present a high-level overview. We want to compare the tree T' that the algorithm creates (the MST according to the order $\tilde{\pi}^t$) with the MST T (the one according to π^t). To do that we show that most of the edges of the two spanning trees are the same, and if the algorithm did not to use an edge e that exists in the MST T it replaced e with another edge that it is not ranked much worse (according to π^t). It turns out that associating the sets of edges that are not common in the two trees is tricky as these edges connect components with each other in the spanning trees and the component structure in the two trees is different. For this reason we define two mappings between the edges of the two trees. The first is the one used in Lemma 10 and maps the edges that exist only in the tree T' produced by the algorithm to the set of (all) edges of the MST T. Lemma 10 is used subsequently in the proof of Theorem 13 to show that the number of edges existing only in T' is $O(\alpha \ln n)$. However, since this mapping is not a one-to-one mapping, it cannot be used to bound the cost of the two solutions. For this reason, in Proposition 12 we prove the existence of a second mapping, a bijection from the set of edges of one tree to the set of edges of the other, with the property that most of the edges are mapped to themselves, while the edges that are mapped to different edges are at most $\alpha \ln n$ far from each other. This proposition is also used in Theorem 13 to show that the difference in the cost of the two solutions is $O(\alpha^2 \ln^2 n)$.

We start with the following lemma which follows easily from Kruskal's MST algorithm.

Lemma 9. *Consider the forest F built by Kruskal's algorithm on $\tilde{\pi}^t$ after all the edges in E_k are processed by the algorithm. If nodes u, v are connected in T_k, then they are connected in F.*

Consider an edge $e = (u, v)$ in T' (the MST of $\tilde{\pi}^t$) that is not in T (the MST of π^t). Let e_{i_k} be the first edge chosen by Kruskal's algorithm on π^t such that u and v are connected in T_k (i.e., in T_k nodes u and v are connected but in T_{k-1} they are not connected).

Lemma 10. *We have*

$$0 < \pi^t(e) - \pi^t(e_{i_k}) \leq 2c_1\alpha \ln n$$

in expectation and with high probability.

Proof. First we prove the first inequality, that is, that edge e_{i_k} is ranked before edge e according to π^t. Recall that $e = (u, v)$ does not belong to E_k, and that in $T_k = (V, E_k)$ nodes u, v are connected. Therefore, adding e to T_k creates a cycle, which means that if it were the case that $e <_{\pi^t} e_{i_k}$, then Kruskal's algorithm would have selected edge e before selecting e_{i_k} when creating the MST T, a contradiction.

For the second inequality assume, for the sake of contradiction, that $\pi^t(e) - \pi^t(e_{i_k}) > 2c_1\alpha \ln n$. Then, for all $j \leq k$ we have with high probability

$$\begin{aligned}\tilde{\pi}^t(e) - \tilde{\pi}^t(e_{i_j}) &\overset{(a)}{>} \pi^t(e) - c_1\alpha \ln n - \pi^t(e_{i_j}) - c_1\alpha \ln n \\ &\overset{(b)}{\geq} \pi^t(e) - \pi^t(e_{i_k}) - 2c_1\alpha \ln n \\ &\overset{(c)}{\geq} 0,\end{aligned}$$

where (a) follows by applying twice Lemma 7, (b) from the fact that for $j < k$ we have $e_{i_j} <_{\pi^t} e_{i_k}$, and (c) from our assumption. But this would mean that for all the edges e_{i_j} for $j \leq k$, we have that $e_{i_j} <_{\tilde{\pi}^t} e$, which by Lemma 9 in turn implies that Kruskal's algorithm could not have selected edge e while building T' (the MST according to $\tilde{\pi}^t$), a contradiction. □

To bound the value D, we note that we can rewrite $D = \sum_{j=1}^{n-1}(i'_j - i_j)$ as $D = \sum_{j=1}^{n-1}(\pi^t(f(e_{i_j})) - i_j)$, where f is any bijection from the set E_{n-1} into the set E'_{n-1}. In what follows we will prove that there exists a mapping f such that for each edge e in $E_{n-1} \cap E'_{n-1}$ we have $f(e) = e$, and for each e in $E_{n-1} \setminus E'_{n-1}$ the corresponding edge $f(e) \in E'_{n-1}$ is such that

$$\pi^t(f(e)) - \pi^t(e) \leq c_1\alpha \ln n.$$

To obtain an upper bound on D we then only need to obtain an upper bound on the number of edges in $E_{n-1} \setminus E'_{n-1}$. The following lemma is instrumental in building the bijection f.

Lemma 11. *For each j, if e_{i_j} is the ℓth edge in $E_{n-1} \setminus E'_{n-1}$ in the order π^t, then:*

(i) among $e_1, \ldots, e_{i_j+2c_1\alpha \ln n}$ there are at least ℓ edges in $E'_{n-1} \setminus E_{n-1}$;

(ii) at least one edge of $E'_{n-1} \setminus E_{n-1}$ is in the set $e_{i_j}, \ldots, e_{i_j+2c_1\alpha \ln n}$.

Proof. We first prove (i) $\implies$ (ii). Suppose not. Then there are at least ℓ edges of $E'_{n-1} \setminus E_{n-1}$ among $e_1, \ldots, e_{i_j-1}$. This implies that $\pi^t(e_{i'_j}) \leq i_j - 1 < i_j = \pi^t(e_{i_j})$, contradicting Proposition 6.

We now prove (i) by contradiction. Assume there are fewer than ℓ edges of $E'_{n-1} \setminus E_{n-1}$ among $e_1, \ldots, e_{i_j+2c_1\alpha \ln n}$. Let S denote the set of all edges except those that come after all edges of E_j in the ordering $\tilde{\pi}^t$. Using Lemma 7 twice, it follows that S is a subset of $e_1, \ldots, e_{i_j+2c_1\alpha \ln n}$, and by our assumption, there are fewer than ℓ edges of $E'_{n-1} \setminus E_{n-1}$ in S. Thus, denoting $r \triangleq |S \cap E_{n-1} \cap E'_{n-1}|$, the forest built by Kruskal's algorithm on the set S in order $\tilde{\pi}^t$ has size less than $\ell + r$. On the other hand, the set S contains a forest of size $r + \ell$, namely T_j. Since Kruskal's algorithm always builds a forest of maximal size, this is a contradiction. □

We are now ready to prove the existence of $f(\cdot)$ suitable for our purpose.

Proposition 12. *There exists a bijection $f : E_{n-1} \mapsto E'_{n-1}$ such that*

1. *for each edge e in $E_{n-1} \cap E'_{n-1}$: $f(e) = e$;*
2. *for each e in $E_{n-1} \setminus E'_{n-1}$ we have: $\pi^t(f(e)) - \pi^t(e) \leq 2c_1\alpha \ln n$.*

Proof. We build the mapping f as follows. We consider the edges in E_{n-1} one by one following their rank in π^t. Let e_{i_j} be the current edge considered. If $e_{i_j} \in E'_{n-1}$, then we set $f(e_{i_j}) = e_{i_j}$. Otherwise, we choose the edge $e \in E'_{n-1} \setminus E_{n-1}$ of minimum rank in π^t that has not been considered in building $f(\cdot)$ yet. Lemma 11 ensures that $\pi^t(e) - \pi^t(e_{i_j}) \leq 2c_1\alpha \ln n$. We then set $f(e_{i_j}) = e$. □

We are now ready to give a bound on D.

Theorem 13. *There is an MST algorithm on evolving graphs such that for every time step t, the solution output by the algorithm at time t satisfies $D = O\left(\alpha^2 \ln^2 n\right)$ in expectation and with high probability.*

Proof. To prove the theorem we will use Lemma 10 to count the number of edges in which the optimal solution and the algorithm's solution disagree, and Proposition 12 to bound the cost. This is the reason that we need to define two mappings (one defined in Lemma 10 and the bijection given by Proposition 12).

By Lemma 10, an edge $e = (u, v)$ that is in T' (the MST according to order $\tilde{\pi}^t$) and not in T (the MST according to order π^t) is found in the $2c_1\alpha \ln n$ positions of π^t following the edge e_{i_k} (that belongs to T), that is the first edge in the order π^t to connect components C_1 and C_2 of T_{k-1}, with C_1 containing u and C_2 containing v. We call edge e a *bad* edge associated to e_{i_k}. To compute the total error, we will count the number of bad edges (associated to any edge in T), and given the existence of the bijection f we can apply Proposition 12 and conclude that the error is at most $2c_1\alpha \ln n$ times the number of bad edges.

Now we compute the number of bad edges. For edge e_{i_k} in the T, which connects components C_1 and C_2 of T_{k-1}, let B_{i_k} be the set of edges between component C_1 and C_2. Since the order π^t is a random permutation of the edges, and since edge e_{i_k} is in T (the MST according to π^t), e_{i_k} has to be among the first $2n \ln n$ ranked edges of π^t with high probability. (This follows from the fact that in the $G(n, m)$ model with $m = 2n \ln n$ edges, a spanning tree exists with high probability.) Now let us consider a potential bad edge e that will be associated with edge e_{i_k}. Since $\pi^t(e_{i_k}) < 2n \ln n$, and since e is in a random position after edge e_{i_k} (here again we use the fact that π^t defines a random permutation of the edges), the probability that e is a bad edge is at most the probability that edge e is one of the $2c_1\alpha \ln n$ edges that follow edge e_{i_k}, which is at most

$$\frac{2c_1\alpha \ln n}{m - 2n \ln n} = O\left(\frac{\alpha \ln n}{n^2}\right).$$

Therefore, the expected number of bad edges associated with edge e_{i_k} is

$$|B_{i_k}| \cdot O\left(\frac{\alpha \ln n}{n^2}\right).$$

However, the sets B_{i_k} associated with all the edges e_{i_k} of the MST T form a partition of all edges. Therefore, the expected number of bad edges is

$$\sum_{i_k : e_{i_k} \in T} |B_{i_k}| \cdot O\left(\frac{\alpha \ln n}{n^2}\right) = O\left(\alpha \ln n\right).$$

Now we apply Proposition 12 and we obtain that the expected total error is $O\left(\alpha^2 \ln^2 n\right)$. Since the number of bad edges is the sum of independent Bernoulli random variables, we also have (by an application of a Chernoff bound) that the total error is $O\left(\alpha^2 \ln^2 n\right)$ with high probability. $\square$

It is sometimes possible to do better than Theorem 13. The idea is to run two sorting algorithms in parallel, the first one is to keep track of the cheapest $O(n \log n)$ edges in the graph, and the other one only on this set, to keep track of the ordering of these edges (which are the only edges that are important in the construction of the MST) more accurately. This can result in a constant bound when α is sublinear in n. See Appendix C for the sketch of this algorithm and its proof.

5. MATROIDS

In this section we study how the results on MST can be generalized to the more general problem of finding a basis of minimum weight in a matroid. Recall the definition of a matroid:

DEFINITION 14. *A* matroid $\mathcal{M}$ *is a pair* (E, S) *where* E *is a finite set of size* m *and* S *is a non-empty collection of subsets of* E, $S \subseteq 2^E$, *satisfying the following properties:*

– hereditary property: *if* $X \in S$ *then* $X' \in S$ *for all* $X' \subseteq X$;

– exchange property: *if* $X, Y \in S$ *and* $|X| > |Y|$, *then there is an element* $e \in X \setminus Y$ *such that* $Y \cup \{e\}$ *is in* S.

Any set in S *is called an* independent set *and any maximal independent set is called a* basis *of* $\mathcal{M}$.

Let $n = |M|$ denote the *rank* of the matroid $\mathcal{M}$, where M is the minimum weight basis. The model for change is similar as before: at any time t, the weights of the elements in E induces an order π^t on E. For every $t > 1$, π^t is obtained from π^{t-1} by swapping α random pairs of consecutive elements. The algorithm can compare the weights of two elements of E in each time step t, and will get the result of this comparison according to π^t. We assume that the initial ordering π^1 is random and that the algorithm knows π^1; otherwise our results hold for sufficiently large t. Our goal is to find a basis of $\mathcal{M}$ of minimum total weight.

As before, we use an ordinal measure of approximation defined as follows: assume the elements in E are indexed such that $e_1 <_{\pi^t} \cdots <_{\pi^t} e_m$, and let $i_1 < \cdots < i_n$ and $i'_1 < \cdots < i'_n$ denote the index of the elements of the minimum weight basis M, and the basis M' computed by the algorithm, respectively. We use the quantity $D = \sum_{\ell=1}^{n} (i'_\ell - i_\ell)$ as a measure of the quality of the solution produced by the algorithm. It is not hard to generalize Proposition 6 for matroids, justifying that D is indeed a reasonable measure of approximation.

As before, we use the sorting algorithm of [2] to keep an approximation $\tilde{\pi}^t$ of the real ordering π^t of all the m elements, and in each step return the minimum weight basis built using the greedy algorithm on $\tilde{\pi}^t$.

An argument essentially identical to the proof of Proposition 8 shows that the solution found by the above algorithm satisfies $D = O(n \log m)$ with high probability. To get a better bound, we need an additional assumption about the matroid $\mathcal{M}$. Namely, we need to assume that the number of disjoint bases of the matroid is not too small. Let k denote the maximum number of disjoint bases in $\mathcal{M}$. In what follows, we assume that there exists a constant $\varepsilon > 0$ such that $k = \Omega\left(m^\varepsilon \ln n\right)$. For example, in the MST problem on a complete graph with N nodes, we have $k = \Theta\left(N\right)$ (Polesskii theorem [4]), $n = N - 1$, and $m = \Theta\left(N^2\right)$. Therefore, this assumption comes for free in this case. Using this condition, we are able to prove the following bound, which generalizes Theorem 13. The proof of this statement is deferred to Appendix D.

THEOREM 15. *For any matroid satisfying the condition* $k = \Omega\left(m^\varepsilon \ln n\right)$, *there is an algorithm that at any time step* t, *outputs a basis such that* $D = O\left(\alpha^2 \ln^2 m\right)$ *in expectation and with high probability.*

It is sometimes possible to obtain a better error, as reported in Appendix E.

6. DISCUSSION AND FUTURE WORK

In this paper we applied the evolving-data model [2] for sorting and selection on sequences to graph problems. The model captures the fact that in many real-life problems the graph changes continually yet for these changes to be observed a protocol must keep querying the graph. We studied weighted and unweighted problems by analyzing the problems of maintaining a minimum spanning tree and of maintaining connectivity between two nodes. We also extended our spanning-tree analysis to general matroids.

Natural extensions include the application of the model to more graph problems such as maintaining a minimum-weight matching, shortest paths, and minimum-flow. In addition, in this paper we based our problem on ordinal weights leading to a simple and clean model. For problems such as matching, shortest path, and flow, it is worth exploring richer models in which edge have actual weights that change gradually. Finally, it is of interest to develop protocols and obtain lower bounds on the performance achievable by distributed algorithms with local knowledge and limited query power.

7. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the EU FP7 Project N. 255403 – SNAPS. Eli Upfal was also supported in part by an award from Yahoo! Research Faculty Research and Engagement Program.

8. REFERENCES

[1] P. K. Agarwal, D. Eppstein, L. J. Guibas, and M. R. Henzinger. Parametric and kinetic minimum spanning trees. In *FOCS*, pages 596–605, 1998.

[2] A. Anagnostopoulos, R. Kumar, M. Mahdian, and E. Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011.

[3] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.

[4] C. J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, New York, 1987.

[5] D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In M. J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, 1999.

[6] D. R. Karger. Random sampling in matroids, with applications to graph connectivity and minimum spanning trees. In *Proc. 34th FOCS*, pages 84–93, 1993.

[7] R. Kleinberg. *Online Decision Problems with large Strategy Sets*. PhD thesis, MIT, 2005.

[8] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers Inc., 2005.

[9] J. G. Oxley. *Matroid Theory*. Oxford University Press, New York, 1992.

[10] D. Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.

[11] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009.

[12] E. Shamir and E. Upfal. A fast construction of disjoint paths in communication networks. In M. Karpinski, editor, *Proc. FCT*, volume 158 of *Lecture Notes in Computer Science*, pages 428–438. Springer, 1983.

[13] A. Slivkins and E. Upfal. Adapting to a changing environment: The Brownian restless bandits. In *Proc. 21st COLT*, pages 343–354, 2008.

APPENDIX

A. ARBITRARY STARTING CONFIGURATION FOR THE CONNECTIVITY PROBLEM

In Section 3 we have analyzed the case where we start the graph in the stationary regime, arguing that as $t \to \infty$ the graph is distributed as a $\mathcal{G}(n, m)$ random graph. In this section we make this statement more quantitative by using the coupling method.

Let $\{G_t = (V, E_t);\ t = 0, 1, 2, \dots\}$ denote the evolution of the graph and let $\{G'_t = (V, E'_t);\ t = 0, 1, 2, \dots\}$ be the evolution of a graph under the same rules of edge insertion and deletions and with G'_0 being distributed as a $\mathcal{G}(n, m)$ random graph. It is easy to see that each G'_t is distributed as a $\mathcal{G}(n, m)$ random graph (i.e., $\mathcal{G}(n, m)$ is the stationary distribution). We define the following *coupling* between the two processes.

Let $U = \{(u, v);\ u, v \in V\}$ be the set of all potential edges in the graph. At time t let (u'_o, v'_o) be a random edge from E'_t and (u'_n, v'_n) be a random potential edge from $U \setminus E'_t \cup \{u'_o, v'_o\}$. Then we let $E'_{t+1} = (E'_t \setminus \{(u'_o, v'_o)\}) \cup \{(u'_n, v'_n)\}$. That is, we follow the process described in Section 2. This describes the evolution of G'_t.

Now we describe the evolution of G_t, which we will couple with that of G'_t. At time t, an edge (u_o, v_o) will be removed and an edge (u_n, v_n) will be added. These edges are selected as follows. We define $E''_t = E'_t \setminus (u'_o, v'_o)$ (useful since (u'_o, v'_o) is a valid edge to be (re)inserted).

- If $(u'_o, v'_o) \in E_t \cap E'_t$ then let $(u_o, v_o) = (u'_o, v'_o)$, otherwise let (u_o, v_o) be distributed uniformly at random in $E_t \setminus E'_t$.
- If $(u'_n, v'_n) \in U \setminus (E_t \cup E''_t)$ let $(u_n, v_n) = (u'_n, v'_n)$, otherwise let (u_n, v_n) be distributed uniformly at random in $E''_t \setminus E_t$.

One can notice that the edge (u_o, v_o) is a uniformly random edge of G_t and (u_n, v_n) is a uniformly random nonedge, so G_t indeed evolves according to the model of Section 2. In addition, notice that if at some time t_0 we have that $E_{t_0} = E'_{t_0}$, then we have that $E_t = E'_t$ for all $t \geq t_0$. We therefore define the *coupling time* t_c as

$$t_c = \inf\{t \geq 0;\ E_t = E'_t\}.$$

Since the distribution of G'_t is according to $\mathcal{G}(n, m)$ then notice that also the graph of interest G_t is distributed as $\mathcal{G}(n, m)$ for all $t \geq t_c$. Therefore, we want to bound the coupling time t_c. We do that in the following theorem.

THEOREM 16. *For any initial state of the graph G_0 the coupling time is $O(m \ln m)$ whp.*

PROOF. For $E, E' \subset U$ with $|E| = |E'|$ define $d(E, E') = |E \setminus E'|$ to be the number of edges by which E and E' differ; we call $d(E, E')$ the *distance* between E and E'. We will show that at every step, $d(E_t, E'_t)$ decreases with sufficiently high probability. At a given step t, notice that if $(u'_o, v'_o) \in E_t \setminus E'_t$, and $(u'_n, v'_n) \in (u'_n, v'_n) \in U \setminus (E_t \cup E''_t)$, we have that $d(E_{t+1}, E'_{t+1}) = d(E_t, E'_t) - 1$, otherwise $d(E_{t+1}, E'_{t+1}) = d(E_t, E'_t)$. Therefore, the probability that the distance decreases equals

$$\frac{|E_t \setminus E'_t|}{|E_t|} \cdot \frac{|U \setminus (E_t \cup E''_t)|}{|U \setminus E''_t)|} = \frac{d(E_t, E'_t)}{m} \cdot \frac{\binom{n}{2} - m - d(E_t, E'_t) + 1}{\binom{n}{2} - m + 1}.$$

Since we have assumed that $m = O(n\sqrt{n})$, and since $d(E_t, E'_t) \leq m$, we have that the second fraction is $\Theta(1)$. Therefore we have that the probability that the distance will decrease is at least $\gamma \frac{d(E_t, E'_t)}{m}$ for some constant γ.

Therefore, the coupling time is stochastically bounded by a random variable, which is the sum of m geometric random variables, with expectation

$$\sum_{d=1}^{m} \frac{m}{\gamma d} = \Theta(m \ln m).$$

Therefore, the expected coupling time is $O(m \ln m)$ and (by applying Chernoff bounds) this holds whp. $\square$

B. NODE-DISJOINT PATHS BETWEEN $O(\sqrt{N})$ NODE PAIRS

THEOREM 17. *There is an algorithm that maintains $h = O(\sqrt{n})$ node-disjoint paths between a given collection of h disjoint pairs of nodes in an evolving graph. The algorithm guarantees that for every t and for each pair (S_i, T_i), the probability that the path output by the algorithm at time t is invalid is at most $O(h \frac{\log n}{n})$.*

Proof Sketch. We sketch the proof for $m = \Theta(n \log n)$ (see [12] for a tighter analysis of this procedure). Let $\{(S_1, T_1), \ldots, (S_h, T_h)\}$ be the set of pairs. We execute the two-path ST connectivity algorithm for each pair (S_i, T_i). The disjoint path algorithm alternates between the h copies of the two-path ST connectivity algorithm in a round-robin fashion with an additional constraint that the $4h$ balls built by the algorithms are disjoint, and no ball contains more than $c_1\sqrt{n}$ nodes for some $c_1 < 1$. When the algorithm visits its node v at a given ball, it adds to the ball the neighbors of v that are not yet in any ball, until the ball has size $c_1\sqrt{n}$. Let $p = c_2 \log n/n$ and $h = c_3\sqrt{n}$. If $c_2 \geq 24/(1 - 4c_1c_3)$ then with high probability each visited node adds at least $\log n$ new nodes to its ball. Thus, after visiting $\sqrt{n}/\log n$ nodes in each ball, all balls are of size $c_1\sqrt{n}$. The probability that any pair is not connected by two edge disjoint path is bounded by $2h(1-p)^{c_1^2 n} \leq 1/n$ for $c_2 \geq 2/c_1^2$ (considering only unvisited nodes).

The execution time of a phase of the disjoint path algorithm is h times the execution time of a phase of the two-path ST connectivity algorithm. Following the proof of Theorem 4, the probability that the two path algorithm outputs an invalid path is dominated by the probability that the path was changed in the last $O(\log n)$ steps of that algorithm, which is $O(h \log n/n)$ in our case because the last $O(\log n)$ steps of each copy of the two-path algorithm are executed during a time interval of $O(h \log n)$ steps. $\square$

C. A BETTER ALGORITHM FOR MST

We now develop an algorithm resulting in a smaller error for the MST on a complete graph, given some restriction on α.

Since the error of the sorting depends on the number of edges to be ordered, if we could restrict the number of edges considered by the sorting algorithm, the total error would become much smaller. In particular, if there exists an $\varepsilon > 0$ such that $\alpha = O\left(n^{1-\varepsilon}\right)$, we can obtain a better bound on the error since the following applies.

Lemma 18. *Consider a subset $E' \subseteq E$ of the edges with $|E'| = O(n \ln n)$, and run the sorting algorithm only on E'. Let $\tilde{\pi}'^t$ be the order obtained from this sorting, and let π'^t be π^t restricted to the edges of E'. If there exists a constant $\varepsilon > 0$ such that $\alpha = O\left(n^{1-\varepsilon}\right)$, for every element e we have that*

$$\left|\pi'^t(e) - \tilde{\pi}'^t(e)\right| = O(1)$$

with high probability for n large enough.

(The proof of Lemma 18 is similar to proof of Lemma 3 in [2], with the difference that we are sorting $O(n \ln n)$ edges and each pair of consecutive edges in π'^t has probability at most $2\alpha/n^2$ to swap at each time step.)

Since $\alpha = O\left(n^{1-\varepsilon}\right)$, we have that (i) Lemma 18 holds, and (ii) the MST is contained in the $O(n \ln n)$ first ranked edges of $\tilde{\pi}^t$. We can then use two sorting algorithms in parallel: one sorts all the n^2 edges in the odd steps, while the other one takes the first $O(n \ln n)$ edges produced by the sorting of all edges, and sort them in the even steps. If we can build an MST using only the edges considered by the second sorting algorithm, we produce that MST in output, otherwise we return the MST built on the order produced by sorting all edges. With the same analysis of Section 4.2, we obtain the following:

Theorem 19. *Let $\alpha = O\left(n^{1-\varepsilon}\right)$ for a constant $\varepsilon > 0$. The error D is $O(1)$ with high probability.*

D. ANALYSIS OF THE EVOLVING MATROID ALGORITHM

In what follows we let M_k be the first k elements in M according to order π^t as chosen by the greedy algorithm: $M_k = \{e_{i_1} <_{\pi^t} e_{i_2} <_{\pi^t} \ldots <_{\pi^t} e_{i_k}\}$. We let M'_k be the first k elements in the approximate basis according to order $\tilde{\pi}^t$: $M'_k = \left\{e_{i'_1} <_{\pi^t} e_{i'_2} <_{\pi^t} \ldots <_{\pi^t} e_{i'_k}\right\}$.

Recall that we assumed that $k = \Omega(m^\varepsilon \ln n)$ for a constant $\varepsilon > 0$, where n denotes the *rank* of the matroid $\mathcal{M}$ and k denotes the maximum number of disjoint bases in $\mathcal{M}$. Then the minimum basis in π^t is found in the lightest $o(m)$ elements, as ensured by the following corollary of Theorem 5.2 in [6].

Corollary 20. *Suppose $\mathcal{M}$ contains k disjoint bases. Then a basis is found in the lightest $O\left(\frac{m \ln n}{k}\right)$ elements of π^t.*

Thus, when $k = \Omega(m^\varepsilon \ln n)$, the minimum weight basis is in the lightest $O\left(\frac{m \ln n}{k}\right) = O\left(m^{1-\varepsilon}\right)$ elements of π^t.

Note that it is not easy to compare M and M', since the elements chosen by the greedy algorithm are not independent. However, as done in Section 4 we can rewrite $D = \sum_{\ell=1}^n (j_\ell - i_\ell)$ as $D = \sum_{j=1}^n (\pi^t(f(e_{i_j})) - i_j)$, where $f(\cdot)$ is any bijection from the set $M = M_n$ into the set $M' = M'_n$. In what follows we prove that if the analogous of Lemma 7 for matroids holds there exists a bijection $f(\cdot)$ such that for each element e in $M_n \cap M'_n$ we have $f(e) = e$, and for each e in $M_n \setminus M'_n$ the corresponding element $f(e) \in M'_n$ is such that

$$\tilde{\pi}^t(f(e)) - \tilde{\pi}^t(e) \leq 2c_1\alpha \ln m.$$

To obtain an upper bound on D we then only need to obtain an upper bound on $|M'_n \setminus M_n| = |M_n \setminus M'_n|$.

The following lemma is instrumental in building the bijection $f(\cdot)$ with the above properties. Its proof is similar to the proof of Lemma 11

Lemma 21. *For each j, if e_{i_j} is the ℓth element in $M_n \setminus M'_n$ in the order π^t, then:*

1. *in $\left\{e_1, e_2, \ldots, e_{i_j + 2c_1\alpha \ln m}\right\}$ there are at least ℓ elements in $M'_{n-1} \setminus M_{n-1}$;*
2. *at least one element of $M'_n \setminus M_n$ is in the set $\left\{e_{i_j+1}, e_{i_j+2}, \ldots, e_{i_j+2c_1\alpha \ln m}\right\}$.*

Proof. We first prove that if property 1 above holds, then 2 must hold. Let assume for the sake of contradiction that 1 holds but 2 does not. Then there are no elements of $M'_n \setminus M_n$ in $\left\{e_{i_j+1}, e_{i_j+2}, \ldots, e_{i_j+2c_1\alpha \ln m}\right\}$. Thus in $\left\{e_1, e_2, \ldots, e_{i_j-1}\right\}$ there are $\geq \ell$ elements of $M'_n \setminus M_n$, and we have that $\pi^t(e_{i'_j}) < \pi^t(e_{i_j})$ (i.e., $i'_j < i_j$), that contradicts Proposition 6 for matroids.

We now prove 1. The proof is by contradiction. Let assume that in $e_1 <_{\pi^t} e_2 <_{\pi^t} \ldots <_{\pi^t} e_{i_j+2c_1\alpha \ln m}$ there are $< \ell$ elements of $M'_n \setminus M_n$. Let F be the elements in order $\tilde{\pi}^t$ before seeing all elements in M_j. Note that F is a subset of elements $e_1 <_{\pi^t} e_2 <_{\pi^t} \ldots <_{\pi^t} e_{i_j+2c_1\alpha\ln m}$, since we assume that the analogous of Lemma 7 for matroids holds.

Let $r = |F \cap M_n \cap M'_n|$ (i.e., r is the number of elements in both M_n and M'_n seen in $\tilde{\pi}^t$ before observing all elements in M_j). Thus $M_j \subseteq F$, and $j = |M_j| = r + \ell$. Now, since there are $< \ell$ elements of $M'_n \setminus M_n$ in $e_1 <_{\pi^t} e_2 <_{\pi^t} \ldots <_{\pi^t} e_{i_j+2c_1\alpha\ln m}$, there are $< \ell$ elements of $M'_n \setminus M_n$ in F, i.e. $|(M'_n \setminus M_n) \cap F| < j$. By the exchange property of matroids there exists $e' \in M_j \setminus ((M'_n \setminus M_n) \cap F)$ such that $(M'_n \setminus M_n) \cap F \cup \{e'\} \in S$. Let $\tilde{\pi}^t(e')$ be the rank of e' at time t in order $\tilde{\pi}^t$. By the hereditary property of matroids $(M'_n \setminus M_n) \cap \{e_1 <_{\tilde{\pi}^t} e_2 <_{\tilde{\pi}^t} \ldots <_{\tilde{\pi}^t} e_{\tilde{\pi}^t(e')-1}\} \cup \{e'\} \in S$, thus e' should have chosen by the greedy algorithm on $\tilde{\pi}^t$, thus $e' \in M'_n$, that is a contradiction. ☐

The following guarantees the existence of $f(\cdot)$ suitable to our purpose. Its proof is the generalization of the proof of Proposition 12 to matroids.

Proposition 22. *There exists a bijection $f : M_n \to M'_n$ such that*

1. *for each element e in $M_n \cap M'_n$: $f(e) = e$;*
2. *for each e in $M_n \setminus M'_n$ the corresponding element $f(e) \in M'_n$ is such that $\pi^t(f(e)) - \pi^t(e) \leq 2c_1\alpha\ln m$.*

Proof. We build the bijection $f(\cdot)$ as follows. We consider the elements in M_n one after the other following their rank in π^t. Let e_{i_j} the current element considered. If $e_{i_j} \in M'_n$, then we set $f(e_{i_j}) = e_{i_j}$. Otherwise, let e be the element in $M'_n \setminus M_n$ of minimum rank in π^t that has not been considered in building $f(\cdot)$ yet. Lemma 21 ensures that $\pi^t(e) - \pi^t(e_{i_j}) \leq 2c_1\alpha\ln m$. We then set $f(e_{i_j}) = e$. ☐

To obtain a bound on D, we now need a bound on $|M'_n \setminus M_n| = |M_n \setminus M'_n|$. To derive this bound we restrict the positions where an element $e \notin M$ must be found in π^t in order to be chosen for M'. Since π^t is a random permutation, this would give us a bound on the probability that e is chosen in M'.

To this end, recall the definition of cycle for a matroid.

Definition 23. *Given a matroid (E, S), a cycle is a minimal dependent set, i.e., $C \subseteq E$ is a cycle if $\forall e \in C, C \setminus \{e\} \in S$.*

The following is a known property of cycles in matroid (e.g., Proposition 1.4.11 of [9]).

Proposition 24. *Let $C_1, C2$ be two different cycles such that $x \in C1 \cap C2$. Then $C1 \cup C2 \setminus \{x\}$ contains a cycle.*

Using the proposition above, we can prove the following.

Lemma 25. *Given $e \notin M$, let $\ell = \min\{j : M_j \cup \{e\} \notin S\}$. Then*

1. *$\pi^t(e) >_{\pi^t} \pi^t(e_{i_\ell}) = i_\ell$;*
2. *if e comes after all elements of M_ℓ in $\tilde{\pi}^t$, then $e \notin M'$.*

Proof. First note that $\ell > 0$ for all $e \notin M$, otherwise e would have been chosen by the greedy algorithm on π^t, i.e. $e \in M$ that is a contradiction. The same argument shows that the first property holds.

Now consider the second property. Since $M_\ell \cup \{e\} \notin S$, there is a cycle $C \subseteq M_\ell \cup \{e\}$. Since $M_\ell \in S$, then $e \in C$. Let $\bar{C} = C \setminus M' \setminus \{e\}$. If $\bar{C} = \emptyset$ (i.e., all the elements of M_ℓ that constitute C are in M'), then e cannot be chosen by the greedy algorithm on $\tilde{\pi}^t$, that is $e \notin M'$.

Otherwise, let $\bar{C} = \{e_{r_1}, e_{r_2}, \ldots, e_{r_k}\}$. For each $e_{r_q} \in \bar{C}$ there must be $C_q \subseteq M'$ such that $C_r \cup \{e_{r_q}\}$ is a cycle, since $e_{r_q} \notin M'$. Consider now the following sequence $\{B_i\}$ of subsets of E: $B_0 = C$; $B_i = B_{i-1} \cup C_i \setminus \{e_{r_q}\}$, for $1 \leq i \leq k$. It is easy to show that (i) B_i contains a cycle for all $0 \leq i \leq k$ (by Proposition 24), and (ii) $B_k \subseteq M' \cup \{e\}$. Thus $M' \cup \{e\} \notin S$, that is $e \notin M'$. ☐

From the lemma above, we can derive the following bound.

Lemma 26. *Given $e \in M' \setminus M$, let $\ell = \min\{j : M_j \cup \{e\} \notin S\}$. We have that*

$$0 < \pi^t(e) - \pi^t(e_{i_k}) \leq 2c_1\alpha\ln m.$$

Proof. The first inequality, that is, that element e_{i_ℓ} is ranked before element e according to π^t follows from Property 1 of Lemma 25.

For the second inequality assume, for the sake of contradiction, that $\pi^t(e) - \pi^t(e_{i_\ell}) > 2c_1\alpha\ln m$. Then, for all $j \leq \ell$ we have that

$$\begin{aligned}\tilde{\pi}^t(e) - \tilde{\pi}^t(e_{i_j}) &\overset{(a)}{>} \pi^t(e) - c_1\alpha\ln m - \pi^t(e_{i_j}) - c_1\alpha\ln m \\ &\overset{(b)}{\geq} \pi^t(e) - \pi^t(e_{i_k}) - 2c_1\alpha\ln m \\ &\overset{(c)}{>} 2c_1\alpha\ln m - 2c_1\alpha\ln m = 0,\end{aligned}$$

where (a) follows by applying twice the analogous of Lemma 7 for matroids, (b) from the fact that for $j < \ell$ we have that $e_{i_j} <_{\pi^t} e_{i_k}$, and (c) from our assumption. But this would mean that for all the elements e_{i_j} with $j \leq \ell$, we have that $e_{i_j} <_{\tilde{\pi}^t} e$, which by Lemma 25 implies that $e \notin M'$, a contradiction. ☐

From Lemma 26, we can derive a bound on $|M' \setminus M|$ that will result in a better bound for D.

Theorem 27. *$|M' \setminus M| \in O(\alpha\ln m)$ in expectation and with high probability, for sufficiently high m.*

Proof. Consider the order π^t at time t, and let M be the minimum weight basis for π^t. Given an element $e \notin M$, $\ell(e) = \min\{j : M_j \cup \{e\} \notin S\}$.

Now we have:

$$\begin{aligned}E[|M' \setminus M|] &= \sum_{e \in E \setminus M} \Pr[e \text{ is in } M'] \\ &\overset{(a)}{\leq} \sum_{e \in E \setminus M} \Pr[\pi^t(e) - \pi^t(e_{i_{\ell(e)}}) \leq 2c_1\alpha\ln m] \\ &\overset{(b)}{\leq} \sum_{e \in E \setminus M} O\left(\frac{\alpha\ln m}{m}\right) \\ &= O(\alpha\ln m)\end{aligned}$$

where (a) follows from Lemma 26 and (b) from the fact that π^t is a random permutation and all the edges in M are among the $o(m)$ lightest edges in π^t.

The result with high probability easily follows from a Chernoff bound. ☐

Combining Proposition 22 and Theorem 27, we can prove Theorem 15.

Proof Proof of Theorem 15.. By Theorem 27, $|M' \setminus M| = O(\alpha \ln m)$ in expectation and with high probability. By Proposition 22, each element in $M' \setminus M$ contributes no more than a factor $O(\alpha \ln m)$ to D. Thus $D = O\left(\alpha^2 \ln^2 m\right)$ in expectation and with high probability. $\square$

E. A BETTER ALGORITHM FOR MATROIDS

Similarly to what we have done for the MST problem, we now provide a better algorithm for general matroids with some restrictions on α.

As before, note that if we could restrict the number of elements to be ordered, the total error would become much smaller. Remember that we impose $k = \Omega(m^{\varepsilon} \ln n)$ for some constant $\varepsilon > 0$, so the minimum weight basis is in the lightest $O\left(\frac{m \ln n}{k}\right)$ elements of π^t. Let assume that there exists ε_1 such that $\alpha = O\left(m^{\varepsilon - \varepsilon_1}\right)$. Moreover, assume that $\alpha = O\left(\frac{m \ln n}{k \ln m}\right)$. (This last condition ensures that the minimum basis M in π^t is found in the lightest $O\left(\frac{m \ln n}{k}\right)$ elements of $\tilde{\pi}^t$ too.) Then we can obtain a better bound on the error since the following applies.

Lemma 28. *Consider a subset $E' \subseteq E$ of the elements with $|E'| = O(m \ln n/k)$, and run the sorting algorithm only on E'. Let $\tilde{\pi}'^t$ be the order obtained from this sorting, and let π'^t be π^t restricted to the elements of E'. If $\alpha = O(m^{\varepsilon_1})$ for a constant $\varepsilon_1 < \varepsilon$, for every element e we have that*

$$\left|\pi'^t(e) - \tilde{\pi}'^t(e)\right| = O(1)$$

with high probability for n large enough.

(The proof of Lemma 28 is similar to the proof of Lemma 3 in [2], considering that we are sorting $O(m \ln n/k)$ elements and each pair of consecutive elements in π'^t has probability at most $2\alpha/m$ to swap at each time step.)

We can then use two sorting algorithm in parallel: one sorts all the m elements in the odd steps, while the other one takes the first $O\left(\frac{m \ln n}{k}\right)$ elements produced by the sorting of all elements, and sort them in the even steps. If we can build a basis using only the elements considered by the second sorting algorithm, we produce that basis in output, otherwise we return the basis built on the order produced by sorting all elements. Using the same analysis of previous section, we obtain the following:

Theorem 29. *If $\alpha = O(m^{\varepsilon_1})$ for a constant $\varepsilon_1 < \varepsilon$, the error D is $O(1)$ with high probability.*