Faster dynamic algorithms and sensitivity oracles for graph connectivity problems

Advisors: Thatchaphol Saranurak and Michael Zieve
Author: Jingyi Gao

July 2021

1 Acknowledgement

I want to thank my advisors Professor Saranurak and Professor Zieve for being very supportive and patient through the research and I also want to thank University of Michigan Math department for providing this opportunity and supporting the research.

2 Prelim

2.1 Motivation and abstract

Network is commonly used to display the relations between individuals, and the underlying structure of the network is a directed graph, where the nodes stand for individuals and edges represent the relations between people. Given a network, for any arbitrary pair of nodes, we might be interested in whether the pair is connected or not, and how much they are connected with each other. k-edge(respectively vertex) connectivity formally measures the connectedness of a pair of vertices in a graph. Specifically, the definitions are as follows

Definition 2.1 ($k-$edge(respectively $k-$vertex) connectivity)**.** *Given a directed graph $G = (V, E)$, a pair of vertices $u, v \in V$ are $k-$edge(resp. vertex) connected in G, if we need to remove at least k edges(resp. vertices) in G to make u and v disconnected in G.*

The project aims to find faster algorithms to check the $k-$edge(and $k-$vertex) connectivity between all pairs of vertices when given a graph. Moreover, we find algorithms which can quickly compute the $k-$edge (or $k-$vertex) connectivity in the new graph obtained from the initial graph by adding or deleting a few edges, by utilizing the computations already done for the initial graph.

2.2 Problem

The goal of the project is to find faster algorithms to check the $k-$vertex connectivity and $k-$ edge connectivity of all pairs of vertices given a graph G. We also seek to faster dynamic algorithms and sensitivity oracles that are able to maintain the connectivity when facing updates of the graph rather than compute everything from the scratch for the new graph. In our problems the update refers to adding or deleting edges in the original graph, and the definition of models are as follows:

Definition 2.2 (Sensitivity Oracle for k-vertex Connectivity)**.** *A sensitivity oracle for k-vertex connectivity is a data structure that works in three phases as follows:*

- ***Phase 1(preprocessing):*** *given a directed graph $G = (V, E)$ with n vertices and a connectivity parameter k, preprocess the graph.*

- ***Phase 2(update):*** *given sets of edges $F_i, F_d \subseteq E$ of size $f = |F_i| + |F_d|$, where F_i is the set of inserting edges and F_d is the set of deleting edges. Change E into $E \cup F_i \setminus F_d$.*

- ***Phase 3(query):*** *given two vertices $s, t \in V$, return $Conn(s,t)$, the vertex connectivity between s and t if they are k-vertex connected in $G_{new} = (V, E \cup F_i \setminus F_d)$.*

We define preprocessing time *of the oracle to be the time the oracle takes in the first phase and* space *of the oracle to be the space used by the oracle to store the data structure.* Update time *is the time used in the second phase, and the* query time *is the time to return the answer of the query.*

Theorem 2.3. *There exists a sensitivity oracle for k-vertex connectivity problems, that given a directed graph G with n vertices and connectivity parameter k, it takes $O((nk)^\omega)$ preprocessing time, $O(f^2 poly(k))$ query time, $O((fk^2)^\omega)$ update time, and $O((nk)^2)$ space, where ω is the matrix multiplication exponent.*

Definition 2.4 (Dynamic algorithm for k-vertex Connectivity)**.** *A dynamic algorithm for k-vertex connectivity is able to first preprocess the graph $G = (V, E)$, then receive a (infinite) sequence of updates one at a time and maintain the desired properties after each update. The model will support the following operations:*

- ***Update(insert or delete):*** *given two vertices $v_1, v_2 \in V$, change the current status of $\overrightarrow{v_1 v_2}$(connected into disconnected and vice versa) and change E into the new set E_{new} accordingly(deleting or adding one edge.)*

- ***Query:*** *given two vertices $s, t \in V$, return $Conn(s,t)$, the vertex connectivity between s and t if they are k-vertex connected in $G_{new} = (V, E_{new})$.*

Theorem 2.5. *There exists a dynamic algorithm for k-vertex connectivity problems, that given a directed graph G with n vertices and connectivity parameter k, it takes it takes $O((nk)^\omega)$ preprocessing time, $O((nk)^2)$ update time, $O(k^\omega)$ query time, and $O((nk)^2)$ space.*

2.3 Known Results

In this section, we will introduce some known results that our work will be based on. Cheung et.al [CLL11] presented an algorithm to transfer the problem of all pairs min-edge-cut in digraph into computing the rank of corresponding matrices in $O(m^\omega)$ time. Another result from Abboud et.al [AGI+18] proved an analogous algorithm to solve all pairs min-vertex-cut in digraph. More specifically, the key theorem stated as follows:

Lemma 2.6. *[AGI+18] Given a digraph $G = (V, E)$, for any vertex $s \in V$, denote the vertices that are adjacent to s with an edge has s as head as I_s, and the vertices that are adjacent to s with an edge has s as tail as O_s. We add two layers of vertices I'_s and O'_s in G, where $|I'_s| = |O'_s| = k + 1$. All vertices in I'_s has an edge attached to s where s is the head, and I_s and I'_s forms a complete directed bipartite graph, directed away from I_s. Similarly all vertices in O'_s has an edge attached to s where s is the tail, and O_s and O'_s forms a complete directed bipartite graph, directed away from O'_s. We call the new graph $G' = (V', E')$. Then construct an $n(2k+3) \times n(2k+3)$ matrix M as follows:*

$$M_{i,j} = \begin{cases} x_{i,j} & \text{if} \quad \overrightarrow{v_i v_j} \in E' \\ -1 & \text{if} \quad i{=}j \\ 0 & \text{otherwise} \end{cases}$$

then replace each indeterminate in M with a random element in field F, and the $s-t$ vertex connectivity is equal to the rank of the submatrix $M^{-1}_{N^{out}(s), N^{in}(t)}$ with high probability, where $N^{in}(u)$ and $N^{out}(u)$ are the set of vertices such that $v \in N^{in}(u), vu \in E'$ and $\forall v \in N^{out}(u), uv \in E'$; and the submatrix of A containing rows B and columns C is denoted as $A_{B,C}$.

this lemma turns the graph problem into computing the rank of a matrix. In order to use the current fastest algorithm for computing matrix rank, we would need to work with matrices with entries containing elements from a field. To guarantee the correctness of the algorithm when replacing indeterminates with specific elements in field, we introduce the following facts:

Theorem 2.7 (Sherman-Morrison-Woodbury[vdBS19]). *Let A be $n \times n$ matrix and P, Q be $n \times f$ matrices, such that* $\det(M)$, $\det(M + PQ^T) \neq 0$. *Define the $f \times f$ matrix $N := Id \cdot \det(M) + Q^T adj(M)P$. Then we have*

$$adj(M + PQ^T) = \frac{adj(M)\det(N) - (adj(M)P)adj(N)(Q^T adj(M))}{\det(M)^f}$$

3 Algorithms

We start with describing the structure of the algorithms.

3.1 Sensitivity Oracle for k-vertex Connectivity

Given a digraph $G = (V, E)$ with unit vertex capacity, the $k-$vertex connectivity sensitivity oracle is as follows:

- Preprocessing(G): we first construct $G' = (V', E')$ and M using Abboud et al's reduction as in Lemma 2.6, then compute $\text{adj}(M)$ and $\det(M)$. The preprocessing step will take total $O((nk)^\omega)$ time.
- Update($F \subseteq E$): we construct $|V'| \times fk^2$ matrices P and Q for which the multiplication PQ^T is the matrix encoding all updates. Then we let $N := Id \cdot \det(M) + Q^T \text{adj}(M)P$ and compute $\text{adj}(N)$ and $\det(N)$. The update will take total $O((fk^2)^\omega)$ time.
- Query(s,t): we compute the specific k^2 entries in $\text{adj}(M + PQ^T)$(namely the k^2 entries in $\text{adj}(M + PQ^T)_{N^{out}(s),N^{in}(t)}$ then compute and return the rank of the matrix consist of these k^2 elements, denoted as $\widetilde{Conn}(s,t)$, and return this value. The query step will take $O(f^2k^6 + k^\omega)$ time.

3.2 Dynamic algorithm for k-vertex Connectivity

The dynamic algorithm to check all pairs $k-$vertex connectivity has the same preprocessing step as the above sensitivity oracle, and is using a similar idea for the update except for the query step we would need to compute everything out instead of compute specific entries since we would need the whole matrix to get prepared for the possible upcoming updates:

- Update($F \subseteq E$) and Delete($F \subseteq E$): we construct $|V'| \times k^2$ matrices P and Q for which the multiplication PQ^T is the matrix encoding one update. We use Theorem 2.7 directly to compute $\text{adj}(M + PQ^T)$, which will take $O((nk)^2)$ time.
- Query(s,t): we compute and return the rank $\widetilde{Conn}(s,t)$ of $\text{adj}(M + PQ^T)_{N^{out}(s),N^{in}(t)}$ in $O(k^\omega)$.

4 Analysis of algorithms

In this section, we proved Theorem 2.3 and Theorem 2.5.

4.1 Sensitivity oracle for k-vertex Connectivity

The main structure of the algorithms based on the result in Lemma 2.6 where all-pairs $k-$vertex connectivity problem turns into getting the rank of a submatrix. The algorithm given by Abboud et al. is only able to deal with a static graph whereas our algorithm should be able to handle edge updates of the graph and maintain the $k-$vertex connectivity.

Lemma 4.1. $\widetilde{Conn}(s,t) = \min\{k, Conn(s,t)\}$ *with high probability, where $Conn(s,t)$ is the vertex connectivity between s and t.*

Proof. As in Lemma 2.6, the rank of the corresponding submatrix of the inverse of coefficient matrix M is the vertex connectivity between two vertices. We next articulate the details in the descriptions of sensitivity oracle for $k-$vertex connectivity for digraph.
If we want to add a batches of f updates, we denote the new graph to be G_{new} and do the follows. First of all, we let P and Q be $n(2k+3) \times fk^2$ matrices where P and Q have exactly fk^2 nonzero elements. When we treat f and k as constants that are much smaller than n, P and Q are very sparse.
Essentially we want to change fk^2 entries in M, denote the new matrix as $M' = M + PQ^T$, then get the inverse M'^{-1} in which encodes the vertex connectivity between all pairs of vertices in the new graph G_{new}. This is because when we add a new edge $\overrightarrow{v_1v_2}$ in G, according to the construction in Lemma 2.6, in G'_{new} we would need to construct a complete directed bipartite graph between O'_{v_1} and I'_{v_2} directed from O'_{v_1}. Therefore, adding one edge in G corresponds to adding k^2 edges in G', whence adding f edges in G corresponds to adding fk^2 edges in G', and this means that we would need to turn fk^2 zero entries in M into nonzero entries.
We use the Theorem 2.7(Sherman-Morrison-Woodbury) to make the update more efficient and take the best use of the previous results that had been computed. The idea is basically the same as in the paper of Brand and Saranurak [vdBS19]. To be concrete, since adjoint matrix is inverse matrix multiply by the determinant factor, and the rank of the matrix is invariant under multiplying a constant factor, it is suffices for us to look at matrix $\text{adj}(M + PQ^T)$ instead of $(M + PQ^T)^{-1}$. Let $N = Id \cdot \det(M) + Q^T\text{adj}(M)P$, then

$$\text{adj}(M') = \text{adj}(M + PQ^T) = \frac{\text{adj}(M)\det(N) - (\text{adj}(M)P)\text{adj}(N)(Q^T\text{adj}(M))}{\det(M)^f}$$

according to Theorem 2.7. When the vertex connectivity of a pair of vertices is queried, we want to compute the rank of a submatrix of M'. Since by the construction in Lemma 2.6, the inner degree and outer degree of all vertices are exactly k, when any pair of vertices are queried, we would need to compute k^2 entries of M' then compute the rank of the matrix consist of these entries. By the method in [vdBS19], we don't need to compute a complete matrix in order to get some specific parts of it. Indeed, since

$$\text{adj}(M')_{i,j} = \text{adj}(M + PQ^T)_{i,j} = \text{adj}(M)_{i,j}\frac{\det(N)}{\det(M)^f} - \frac{\left(\vec{e_i}^T\text{adj}(M)P\right)\text{adj}(N)\left(Q^T\text{adj}(M)\vec{e_j}\right)}{\det(M)^f} \tag{1}$$

we only need to do k^2 such computation to get all we want. And notice that $\text{adj}(M), \det(M)$, and $\text{adj}(N), \det(N)$ are required for computing any arbitrary entry in M', whence the computation of $\text{adj}(M)$ and $\det(M)$ can be done once the original graph is given, namely in the preprocessing phase, and $\text{adj}(N), \det(N)$ can be computed once the update is given, namely in the update phase. □

Note that it's nontrivial how the update can be done in constant time (if we treat f and k as constants.) The most straightforward way is to compute the whole inverse $(M + PQ^T)^{-1}$, which takes $O((nk)^\omega)$ time, and take the corresponding submatrix then compute the rank. We next analyze the time complexity and prove Theorem 2.3.

Proof of Theorem 2.3. The correctness of the algorithm is proven in Lemma 4.1. We next analyze the time complexity of each phase and the space complexity:

1. Preprocessing: for the prerocessing step, we compute $\det(M)$ and $\text{adj}(M)$. According to the construction in Lemma 2.6, M is a $n(2k+1) \times n(2k+1)$ matrix, and therefore it would take $O((nk)^\omega)$ time to preprocess the matrix M.

2. Update: after receiving P and Q, we compute $\text{adj}(N)$ and $\det(N)$ for the update step where recall that $N := Id \cdot \det(M) + Q^T\text{adj}(M)P$.

   (a) We first compute $Q^T\text{adj}(M)P$ in $O(f^2k^4)$ operations. This is because P and Q only has fk^2 nonzero entries respectively, whence left multiply $\text{adj}(M)$ by Q^T is simply taking the scalar multiple of fk^2 rows of $\text{adj}(M)$, and similarly right multiply $Q^T\text{adj}(M)$ by P is simply taking the scalar multiple of fk^2 columns of $\text{adj}(M)$. Therefore the matrix $Q^T\text{adj}(M)P$ is just f^2k^4 elements of $\text{adj}(M)$ each multiplied by a non-zero element of P and Q.

(b) Then we can compute $\det(N)$ and $\text{adj}(N)$ in $O((fk^2)^\omega)$ time.

3. Query: eventually we compute the desired k^2 entries in $\text{adj}(M')$ and the rank of the matrix consist of these elements in $O(f^2k^6 + k^\omega)$. To compute $\text{adj}(M')_{i,j}$, according to equation (1), we still need to compute $\vec{p} = \left(\vec{e_i}^T \text{adj}(M)P\right)$ and $\vec{q} = \left(Q^T \text{adj}(M)\vec{e_j}\right)$ and put everything together. It takes $O(fk^2)$ to compute both $\vec{p}$ and $\vec{q}$ since they are fk^2 elements in $\text{adj}(M)$ multiply by scalar in P and Q. Then, it takes $O(f^2k^4)$ to compute $\vec{p}\text{adj}(M)\vec{q}$. Hence, it takes $O(f^2k^6)$ to compute all k^2 elements and $O(k^\omega)$ to compute the rank.

For this algorithm, we only need the space to store the $n(2k+3) \times n(2k+3)$ coefficient matrix, which gives $O((nk)^2)$ space.
Notice that the update and query time and the space complexity is optimal if we treat f and k as constants. $O(n^2)$ is the optimal space complexity can be proven by contradiction. Assume the opposite. If we want to store the relation between all pairs within n vertices, and if we treat this as a function(where preimages are pairs of vertices and images are 0 or 1 indicaating connected or disconnected) we have 2^{n^2} possible such functions. If we have s spaces where $s < n^2$, then we can only store 2^s such functions. Since $2^{n^2} > 2^s$, we have a surjective map from space of size 2^{n^2} to space of size 2^s. This is to say that by pigeon hole principle, there must exists at least two functions that are mapped into a same spot, namely there exists two different graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, where $|V| = n$, such that when we query any two vertices, the algorithm will return an identical result, which is a contradiction, and the optimality is proved.

□

4.2 Dynamic algorithm for k-vertex Connectivity

Proof of Theorem 2.5. The dynamic algorithm for $k-$vertex connectivity for digraph works similarly as the sensitivity oracle. The only difference is that for the update step, instead of applying Theorem 2.7 and only compute the specific entries of $\text{adj}(M + PQ^T)$, in this case we need to compute the whole matrix since we are facing consecutive sequence of updates instead of a batches of updates, and hence we always need the whole matrix to be ready for the next update. □

References

[AGI+18] Amir Abboud, Loukas Georgiadis, Giuseppe F Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemysław Uznański, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. *arXiv preprint arXiv:1807.05803*, 2018.

[CLL11] Ho Yee Cheung, Lap Chi Lau, and Kai Man Leung. Graph connectivities, network coding, and expander graphs. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 190–199. IEEE, 2011.

[vdBS19] Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 424–435. IEEE, 2019.