Streaming Algorithm for Determining a Topological Ordering of a Digraph

Abhabongse Janthong
Advisor: Paul Valiant

April 29, 2014

Abstract

Finding a topological ordering for a directed graph is one of the fundamental problems in computer science. Several textbook-standard algorithms using linear memory have been discovered and utilized to solve several other problems for many decades, especially in resolving dependencies and solving other graph connectivity problems. However, these algorithms are becoming less practical nowadays as the size of the data we are working with is getting much larger because the data cannot fit into memory, prohibiting these algorithms from randomly accessing the input graph data.

For this thesis, our goal is to solve the topological ordering problem of a directed graph with a size that is substantially much larger than the size of available memory. Specifically, we study the problem in the context of the streaming settings in which the set of edges of the graph are provided through a data stream instead of being stored entirely on a random access memory. A recent lower bound result – rising from the analysis of communication complexity – shows that there is no one-pass streaming algorithm for the topological ordering problem that could achieve memory usage that is sublinear in the number of edges of a graph. Our work addresses on the remaining question of whether there is a multi-pass algorithm using sublinear working memory to solve this problem.

Our first result discusses a conjecture based the communication complexity. We study a specific game of communication between Alice and Bob in which both parties attempt to solve the s–t reachability problem of a 3-layer graph using small memory. The solution to this problem helps us construct a streaming algorithm for topological sorting. Unfortunately, using the notion of additive combinatorics, we are able to construct a counterexample that enforces a large lower bound on the required memory in order to solve the conjectured problem.

For the other result, we provide a surprisingly simple steaming algorithm for determining a topological ordering of an n-vertex directed graph; by storing up to k *arbitrary* incoming edges for each vertex in each pass of the stream, this algorithm achieves $\Theta(nk)$ working memory and uses $\lceil n/k \rceil$ passes on the input, where k is an integer parameter. This result is a nice tradeoff between the number of passes of the streaming and the amount of memory usage.

Contents

1 Introduction

In this section, we provide some backgrounds to a fundamental problem in which we are interested, we explain the goals we try to achieve in this thesis.

1.1 Topological ordering problem

Let us begin by revisiting the definition of a topological ordering (also known as a topological sort).

Definition 1 (Topological ordering)**.** Let G be a directed graph. A *topological order* of G is an ordering of vertices in $V(G)$ such that, for each edge $(u, v) \in E(G)$, u precedes v in the ordering.

Determining a topological ordering is critical in various problems such as in resolving dependencies. Furthermore, a topological sort can be used immediately to solve several simpler graph questions, such as acyclicity and problems related to s–t connectivity. There are several efficient algorithms for this problem. One simple algorithm involves modifying a depth-first search algorithm (see [1, p. 613]). This algorithm takes a linear time $\Theta(n + m)$ and uses $\Theta(n + m)$ working memory to compute a solution, where n is the

number of vertices and m is the number of edges. Note that all standard algorithms use $\Omega(n+m)$ working memory because they store the entire graph in memory. To circumvent the memory limitation, we turn to the notion of the streaming model, introduced in [2], and generalized to the multi-pass setting in [3].

1.2 Streaming model

In the streaming model, some or all parts of an input cannot be randomly accessed in memory. Rather, the input is divided into a sequence of smaller slices. Each input slice from the sequence is scanned and processed in order. By managing to obtain and store only the essential information about input slices being scanned up to each position in the input sequence, one might be able to solve the problem for the given input with relatively small memory.

For the sake of simplicity, we assume that the set of vertices $V(G)$ of an input graph G is fixed and labeled 0 through $n-1$, and they are hard-coded into all algorithms we are about to discuss here. Hence, the set of edges $E(G)$ is the only input of the algorithms. Specifically, the streaming algorithms will receive each edge through the input stream one after another in some arbitrary order.

1.3 Previous result

Based on the result by Bordaille, Mathieu, and Migler [4], it is impossible – even under the streaming model – to determine the acyclicity of a directed graph using sublinear working memory, if only one pass on the input is allowed (let alone finding a topological ordering which requires the graph to be acyclic). We will discuss this in more detail in section 2 of this thesis.

Due to the aforementioned result, we examine a streaming algorithm that allows mutiple passes on input allowed.

2 Communication complexity and theoretical lower bounds

2.1 Formal definition

Let us visit the notion of communication complexity, which was introduced by Yao [5].

Definition 2 (Communication complexity)**.** Let $X, Y \subseteq \{0,1\}^k$ be sets of bit strings of length k, and let $f : X \times Y \mapsto \{0,1\}$ be a boolean function to be computed.

In a *communication complexity* problem, two parties, commonly known as Alice and Bob, want to jointly compute the value of $f(x,y)$, given that initially only Alice knows $x \in X$ and only Bob knows $y \in Y$. The computational cost for this type of problem is the number of bits exchanged between the two parties.

The analysis of communication complexity is a useful technique for proving lower bounds on working memory for various problems. In fact, Bordaille, Mathieu, and Migler [4] use this concept to prove the lower bound of one-pass streaming algorithms for determining acylicity of a directed graph. We reconstruct this proof below in order to illustrate how communication complexity relates to the proofs of lower bounds.

2.2 Lower bound proof of one-pass streaming algorithm for acyclicity problem

This proof was given by Bordaille, Mathieu, and Migler [4].

First of all, let us introduce the bit-vector communication problem, which is in a variant of the above communication complexity model where all communication is one-sided, from Alice to Bob only.

Problem 3 ([4], Bit-vector problem)**.** Suppose Alice has a bit-vector x of length m and Bob has an index $i \in [m]$. Alice would like to send messages to Bob, comprising as few bits as possible, so that Bob can compute the value $x[i]$; Bob may not send any information to Alice (including the index value i).

Ablayev [6] has proven the following (simplified) theorem.

Theorem 4 ([6])**.** *For the bit-vector problem in which Alice has a bit-vector x of length m and Bob has an index i, it requires $\Omega(m)$ bits of communication from Alice to Bob using a randomized algorithm for Bob to correctly determine the value of $x[i]$ with constant probability.*

In other words, nothing Alice can do is substantially more effective than the obvious strategy: Alice sends the entire bit vector x to Bob so that Bob could obtain $x[i]$ directly. There is no better way to communicate with fewer bits that would achieve the same result.

Bordaille, et al. [4] use this result to prove a lower bound for the acyclicity problem in a graph by showing that, if we could detect acyclicity in a graph in the one-pass streaming setting using little memory, we could leverage this algorithm to solve the bit-vector problem, a contradiction.

1. Let $n = \lceil \sqrt{m} \rceil$. Construct a directed bipartite graph G with $V(G) = L \cup R$, where $|L| = |R| = n$, and vertices in each of L and R have labels in $[n]$, a set of integers 0 through $n - 1$.

2. Alice constructs a set of edges E based on the bit vector x. Specifically, E contains a directed edge $(u, v) \in L \times R$ if and only if $x[un + v] = 1$. Observe that each bit of the vector x determines whether an edge exists between distinct pairs of vertices in L and R respectively.

3. Alice simulates the streaming algorithm using edges in E as the input stream, and once she is done, she records the entire state of memory after the simulation and give it to Bob.

4. Bob resumes the simulation of the streaming algorithm by constructing a single directed edge $(v^*, u^*) \in R \times L$ where $u^*n + v^* = i$, and then provide this edge to the input stream as the last edge.

5. If the simulation outputs that the graph is cyclic, because G is bipartite, it must be the case that $(u^*, v^*) \in E$, which implies that $x[u^*n + v^*] = x[i] = 1$ by construction. So Bob should output 1 in this case. Otherwise, $(u^*, v^*) \notin E$, which implies that $x[u^*n + v^*] = x[i] = 0$, and Bob should then output 0.

The reduction is obviously correct from the last step of the simulation above. From theorem 4, we have that the state of memory of the simulation at the moment when Alice sent her memory contents to Bob must be $\Omega(m)$, and thus the streaming algorithm must be using $\Omega(m)$ memory. Therefore, we conclude that any one-pass streaming algorithm for the acyclicity problem has a linear working memory lower bound.

3 Conjecture

This section discusses a conjecture to the topological problem, which later turns out to be false.

3.1 An interesting problem

Let us consider the following special problem.

Problem 5. Let G be a tripartite graph with $V(G) = P \cup Q \cup R$, where each "layer" P, Q, and R, contains $\Theta(n)$ vertices, and all edges $E(G)$ belong to either $E_{PQ} \subseteq P \times Q$ or $E_{QR} \subseteq Q \times R$.

Our task is to help Alice choose a subset of edges $A \subseteq E_{PQ}$ and help Bob choose another subset of edges $B \subseteq E_{QR}$ such that, for any randomly chosen $u \in P$ and $v \in R$ that are connected in G with a path of length 2, then there exists a path of length 2 that contains an edge in A or B. Additionally, the chosen subset of edges A anb B should be relatively as small as possible.

We initially conjectured that for any input graph G of the topological ordering problem, G could be decomposed layer-by-layer into a decomposition tree with multiple tripartite graphs at the bottom of this tree. Then we could use an algorithm for Problem 5 to reconstruct the solution in a bottom-up fashion.

Unfortunately, our conjecture was wrong, as we how in the next part.

3.2 Disproving the conjecture[1]

We will give an example of the input graph $\tilde{G}$ for Problem 5, which has the $\Omega(n^2/2^{O(\sqrt{\log n})})$ lower bound on the total size of the chosen subsets A and B required to solve the problem.

[1]We are very grateful to Po-Shen Loh (personal communication, January 2014) for pointing out a counterexample to us.

Before we give a counterexample, let us look at the definition of a matching and an induced matching of a bipartite graph [7, lecture 20].

Definition 6. Let G be a bipartite graph with $V(G) = L \cup R$ and $E(G) \subseteq L \times R$.

A subset of edges $M \subseteq E(G)$ is a *matching* of G if every pair of edges in M do not share an endpoint.

A matching M of G is an *induced matching* if every pair of edges in M are not connected by an edge in $E(G)$.

Now let us consider an example of a bipartite graph $\hat{G}$ defined as follows.

Let $\hat{G}$ be a bipartite graph with $V(\hat{G}) = \hat{L} \cup \hat{R}$ such that $\hat{L} = [2n]$, $\hat{R} = [3n]$, and

$$E(\hat{G}) = E_0 \cup E_1 \cup \ldots \cup E_{n-1} \subseteq \hat{L} \times \hat{R} \tag{1}$$

$$E_x = \Big\{(x + a_i, x + 2a_i) : i \in \{1, 2, \ldots, r\}\Big\} \qquad \text{for } x = 0, 1, \ldots, n-1 \tag{2}$$

where $A = \{a_1, a_2, \ldots, a_r\}$ is a set of r distinct integers from $[n]$ that does not contain any three-term arithmetic progression.

We prove the following lemmas about the graph $\hat{G}$

Lemma 7. *Every pair of subset of edges E_x and E_y is disjoint.*

Proof. For every edge $(u_1, v_1) \in E_x$, we know that $2u_1 - v_1 = x$ by construction. Similarly, $2u_2 - v_2 = y$ for every edge $(u_2, v_2) \in E_y$. Hence, E_x and E_y do not share any edge since $x \neq y$. □

Lemma 8. *Each subset of edges E_x is an induced matching.*

Proof. Consider a pair of distinct edges $(x + a_i, x + 2a_i), (x + a_j, x + 2a_j) \in E_x$. If both edges shared an endpoint, then we get $x + a_i = x + a_j$ or $x + 2a_i = x + 2a_j$. In both cases, we had $a_i = a_j$ which suggests that both edges were actually the same. This contradiction leads to the conclusion that the set of edges E_x is indeed a matching of $\hat{G}$.

Next, consider a pair of distinct edges $(x + a_i, x + 2a_i), (x + a_j, x + 2a_j) \in E_x$ again. If both edges were connected by an edge, then without loss of generality, $\hat{G}$ must contain an edge $(x + a_i, x + 2a_j)$. By construction of $\hat{G}$, there exists a_k such that

$$(x + 2a_j) - (x + a_i) = a_k \qquad \Longrightarrow \qquad a_j - a_i = a_k - a_j \tag{3}$$

which suggest that A contains a three-term arithmetic progression a_i, a_j, a_k. This contradiction leads to the conclusion that the matching E_x is an induced matching. □

From both lemmas, it follows that the set of edges $E(\hat{G})$ can be partitioned into n induced matchings, each of which has exactly r edges. According to Behrend's theorem in additive combinatorics [7, lecture 12], there exists a subset A of $[n]$ that does not contain any three-term arithmetic progression, and whose cartinality, $r = |A|$, is at least $n/2^{O(\sqrt{\log n})}$. In other words, the number of total edges in the graph $\hat{G}$ could be at least as large as $n^2/2^{O(\sqrt{\log n})}$.

Based on the example graph $\hat{G}$, we construct the tripartite graph $\tilde{G}$ for problem 5 as follows:

> For each edge $(x + a_i, x + 2a_i) \in E_x \subseteq E(\hat{G})$, we insert two edges $(x + a_i) \to (x) \to (x + 2a_i)$ into the graph $\tilde{G}$, where vertices $(x + a_i)$, (x), and $(x + 2a_i)$ belongs to the layers P, Q, and R respectively.

It is obvious by the construction that the number of edges in $\tilde{G}$ we have just constructed is twice the number of edges in $\hat{G}$. Now we move onto the important lemma.

Lemma 9. *Let (u, v) be an edge in $\hat{G}$. Then there exists a unique path of length 2 connecting $u \in P$ and $v \in R$ within $\tilde{G}$.*

Proof. Suppose $(u, v) \in E_x$ for some subset E_x of $E(\hat{G})$. Obviously, there is a path of length 2 from u to v through $x \in Q$ in graph $\tilde{G}$. We need to show that such path is unique.

Assume for the sake of contradiction that there exists another path of length 2 from u to v through y in graph $\tilde{G}$. According to Lemma 7, it follows that (u, v) does not belong to E_y, which means that there exists $u' \in P$ and $v' \in Q$ such that both (u, v') and (u', v) belong to E_y. However, these two edges are connected by (u, v) in the graph $\hat{G}$. Hence, it contradicts the fact that E_y is an induced matching as shown in Lemma 8. Therefore, the path of length 2 from u to v is indeed unique. □

Corollary 10. *It immediately follows from this lemma that, for each edge $(u, v) \in V(\hat{G})$, either Alice needs to have $(u, x) \in A$ or Bob needs to have $(x, v) \in B$, where x is the intermediate vertex of the unique path between u and v. Therefore, the number of edges that need to be stored in A or B is at least as large as the number of edges of $\hat{G}$, which is at least $n^2/2^{O(\sqrt{\log n})}$.*

So the tripartite graph $\tilde{G}$ is an example of the input for Problem 5 where both Alice and Bob are inevitably required to store a significantly large number of edges in order to have at least one person solving the connectivity problem correctly. Therefore, our conjecture is disproven.

4 Proposed algorithm

In this section, we propose a streaming algorithm for the topological ordering problem using $\Theta(nk)$ working memory with $\lceil n/k \rceil$ passes on the input stream.

4.1 Main structure of the streaming algorithm

We begin by considering this main lemma.

Lemma 11. *There exists a streaming algorithm that, given a directed acyclic graph G with vertices labeled 0 through $n-1$, finds the first k vertices of some topological order of G, using $\Theta(nk)$ working memory in a single pass. To be specific, the algorithm returns a sequence L of distinct vertices such that*

(1) The length of the sequence is (at least) k;

(2) Each vertex $L[i]$ does not *succeed all vertices in $\{L[j] : j > i\} \cup (V(G) - L)$.*

Once we have this lemma, we can express a solution to this problem with the following recursion.

$$\textsc{Topological-Sort}(G) = \begin{cases} \textsc{empty-list} & \text{if } G \text{ is empty} \\ L_G \circ \textsc{Topological-Sort}(G - L_G) & \text{otherwise} \end{cases} \tag{4}$$

where L_G is the sequence of the first k vertices of some topological order of G obtained by applying the algorithm from Lemma 11.

In other words, starting from an original graph G, we apply Lemma 11 to obtain the first k vertices. Once we remove these vertices from G we then apply the same Lemma again on the remaining graph to obtain the next k vertices of the ordering. By repeating this same process for at most $\lceil n/k \rceil$ passes, we will eventually exhaust all vertices of the graph. The solution is obtained by concatenating all sequences obtained during the repeated process. Therefore, the number of passes performed by this algorithm is at most $\lceil n/k \rceil$. Additionally, the lemma subroutine obviously dominates the utilization of working memory, so this algorithm has the $\Theta(nk)$ working memory.

We have just established the following theorem showing a nice tradeoff between working memory and the number of passes.

Theorem 12. *There is a streaming algorithm for the topological ordering problem using $\Theta(nk)$ working memory that makes $t = \lceil n/k \rceil$ passes on the input stream.*

Alternatively speaking, there is a streaming algorithm for the topological ordering problem using $\Theta(n^2/t)$ working memory in t passes.

Although the solution of the topological ordering problem described above is written in terms of recursion, in fact this algorithm could easily be implemented in an iterative fashion. The key idea is to keep track of whether or not each vertex is already removed from the graph. The algorithm is described below.

4.2 Proof of Main Lemma

We begin the proof of Lemma 11 by describing an algorithm for the problem, followed by the analysis of the working memory and the proof of correctness.

Algorithm. Let G be a directed graph. Here is the high level explanation of the algorithm.

1. Scan the entire stream of edges $E(G)$ and selectively store some of them in memory.

   Specifically, for each vertex v, we store up to k edges directed into v. If there are more than k of such edges, we store only the first k encountered in the stream and throw away the rest, and set the variable $r[v] = \textsc{true}$ to indicate that some edges directed into v were thrown away. Otherwise, $r[v]$ is FALSE.

   Let H denote the subgraph of G induced by edges currently in the memory.

2. Initialize the empty list $L = [\ ]$, which will later be updated with vertices in some topological order.

3. While there exists a vertex $u \in H$ such that $\text{indeg}_H(u) = 0$ (i.e. u has no incoming edges) and $r[u] = \textsc{false}$, do

   3.1. Append u to the list L; and

   3.2. Delete u and all of its incident (outgoing) edges from H.

4. Return L

Proof of Lemma 11. Clearly, the above algorithm uses $\Theta(nk)$ working memory as each vertex has at most k incoming edges as constructed by the algorithm. We are left to prove that the output of this algorithm indeed satisfies Lemma 11.

> *Proof of property (2) from Lemma 11.* Consider each vertex u chosen at step 3 of this algorithm. By construction (see step 3.2), we know that $\text{indeg}_H(u) = 0$ and $r[u] = \textsc{false}$, which implies that u definitely does *not* succeed the remaining vertices in H. These remaining vertices $V(H) - \{u\}$ would either end up located in L after the ith position or would not end up in L at all. Hence, L satisfies the property (2) of the lemma. □

For property (1), we use a proof by contradiction.

> *Proof of property (1) from Lemma 11.* Assume that the algorithm terminates with $|L| < k$. That is, before reaching step 4, $\text{indeg}_H(u) > 0$ or $r[u] = \textsc{true}$ for all $u \in H$. Consider each of the following cases.

Case 1. If $\text{indeg}_H(u) > 0$ for all vertices $u \in H$, then subgraph H contains a cycle, contradicting the fact that G is acyclic.

Case 2. Otherwise, there exists $u \in H$ such that $\text{indeg}_H(u) = 0$ and $r[u] = \text{TRUE}$. By construction of the algorithm, the vertex u must have had exactly k incoming edges in the *original* subgraph H at the end of step 1. However, since then, number of vertices removed from H is less than k, which guarantees that less than k edges directed to u were removed, and thus u must still have some incoming edges left (i.e. $\text{indeg}_H(u) > 0$). Hence, a contradiction occurs.

Therefore, we conclude that the output list L of the algorithm is the first k vertices of a topological ordering of the input graph G. □

Hence, we may now conclude that the output of the algorithm is correct, and the working memory of this algorithm is indeed $O(nk)$.

□

4.3 Details of the algorithm

Here is one example of how the streaming algorithm could be implemented.

```
TOPOLOGICAL-SORT(G, k)
 1  unselected = V(G)     // a set of vertices remaining in the graph
 2  L = EMPTY-LIST        // a topological order of graph (to be filled)
 3  while unselected ≠ ∅
 4      H = (unselected, ∅)     // a subgraph of G
 5      r[v] = FALSE for each vertex v ∈ unselected
 6      for each edge uv ∈ E(G)     // this is one pass on the input stream
 7          if u ∉ unselected or v ∉ unselected
 8              Skip this edge.
 9          elseif indeg_H(v) == k
10              r[v] = TRUE
11          else
12              Add edge uv into the subgraph H
13      while ∃u ∈ G, indeg_H(u) == 0 and r[u] is FALSE
14          Append u to L
15          Remove u from both subgraph H and the set unselected
16  return L
```

For the efficiency of this algorithm, here are the necessary implementation details:

- To represent the set of remaining vertices *unselected*, we use a bit-vector of length n to represent the set of vertices *unselected*, which would allow us to check or modify the set membership of any vertex without taking significant space and time.

- For the edges structure subgraph H, we only need to maintain an adjacentcy list of edges going out from each vertex of H. For the number of incoming edges of each vertex, the integer counters are sufficient. Such adjacency lists of edges are necessary when a vertex is removed from the subgraph H since all incident (outgoing) edges must also be removed. When an edge uv is removed, we also decrease the counter of incoming edges of v by 1. Additionally, if this counter $\text{indeg}_H(v)$ hits 0, and $r[v] = \text{FALSE}$, we push this vertex v to a queue which will be useful for obtaining the next vertex to be removed from the graph.

By the way, this algorithm can be slightly modified to support detection of cycles in the input graph. Particularly, if during the scan of the input we find an edge $uv \in E(G)$ such that $u \in$ *unselected* and $v \notin$ *unselected*, then the graph must contain a cycle. At this point, the algorithm may terminate with an error message indicating that the graph is not acyclic – the property required for a topologicla ordering to exist.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

[2] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the twenty-eighth annual ACM symposium on theory of computing*, STOC '96, (New York, NY, USA), pp. 20–29, ACM, 1996.

[3] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, "On graph problems in a semi-streaming model," *Theor. Comput. Sci.*, vol. 348, pp. 207–216, Dec. 2005.

[4] G. Borradaile, C. Mathieu, and T. Migler, "Lower bounds for testing digraph connectivity with one-pass streaming algorithms," *ArXiv e-prints*, Apr. 2014.

[5] A. C.-C. Yao, "Some complexity questions related to distributive computing (preliminary report)," in *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, (New York, NY, USA), pp. 209–213, ACM, 1979.

[6] F. Ablayev, "Lower counds for ooe-way ppobabilistic communication complexity and their application to space complexity," *Theoretical Computer Science*, vol. 157, pp. 139–159, 1996.

[7] S. Raskhodnikova, "Lecture scribes on sublinear algorithms." Sublinear Algorithms course, Spring 2012, Pennsylvania State University.