Graph Algorithms On Distributed Networks and Databases

ShanghaiJiaotong University
JinShan Liu
5100309049

1 Summary:

Nowadays, people's increasing ability to generate large amount of information forces us to deal with massive data set, often with the size of terabytes, even petabytes. When dealing with such massive data, algorithms with polynomially computation are not efficient enough anymore. In this article, we study the powers and limits of fundamental graph and geometric algorithms when there are limited computational resources, such as time, memory, and communication, which are usually required to be sublinear in the input size. And what I am most interested in is how fast we can solve graph problems on data streams.

Here, we only focus on graph problems where the input is a sequence of edges. Traditionally, solving graph problems requires linear space in the number of nodes. However, using randomization, which allowing some possibility to give a wrong answer, we can check many graph properties in poly-logarithmic space. In this case, randomness is a crucial tool.

2 Introduction

2.1 Introduction of Distributed Algorithm

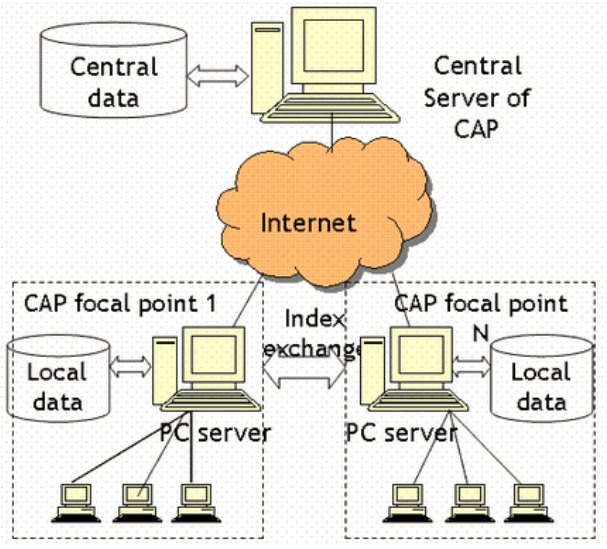

Figure 1: Distributed Network

Typically, we can solve a single problem in a single computer.But we know that the computation time will be very long since the some problems are really tough. However, if we can separate such a complicated problem into several independent parts, and all these computers compute altogether, the time will be definitely reduced. Then comes the idea of distributed algorithm.

A distributed algorithm is an algorithm designed to run on computer hardware constructed from interconnected processors. Distributed algorithms are used in many varied application areas of distributed computing, such as telecommunications, scientific computing, distributed information processing, and real-time process control. Distributed algorithms are sub-type of Parallel algorithm, typically executed concurrently, with separate parts of the algorithm being run simultaneously on independent processors, and having limited information about what the other parts of the algorithm are doing.

Just take a very easy example. Suppose given two n-bits strings x and y, we want to check whether x is equal to y. Using a single computer, we need to check bit by bit. However,if we have two computers, we can have the following method:

1. Send $\frac{n}{2}$ bits to Alice, and then Alice check whether first half n bits are the same. If

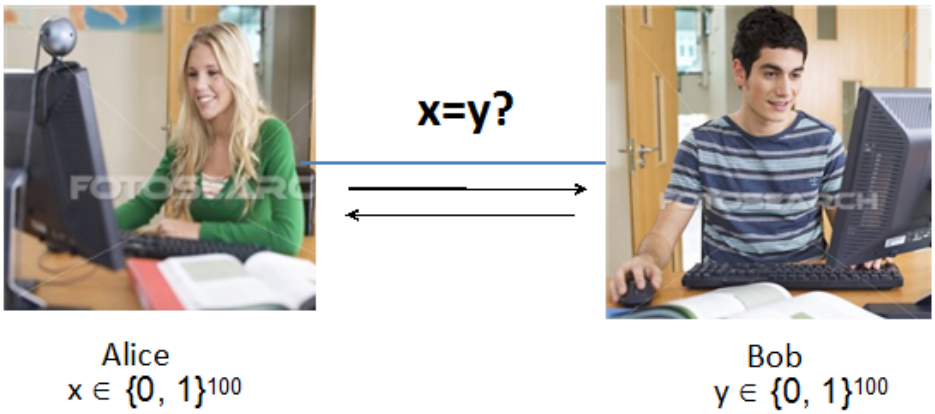

Figure 2: Check whether x=y

they are the same, send 1 to Bob, otherwise, send 0 to Bob.
2. Then Bob compare the left half n bits, and combine his result with Alice's result. Finally, Bob gives the final result.

In this case, the computation time is only half the original one. The efficiency is improved. This is the basic idea about distributed algorithm. In later sections, we will show how to use distributed network to solve graph problems.

2.2 Communication Complexity

The notion of communication complexity was introduced by Yao in 1979[2] who investigated the following problem involving two separated parties (Alice and Bob). Alice receives an n-bit string x and Bob another n-bit string y, and the goal is for one of them (say Bob) to compute a certain function f(x,y) with the least amount of communication between them. Note that here we are not concerned about the number of computational steps, or the size of the computer memory used. Communication complexity tries to quantify the amount of communication required for such distributed computations.

2.3 Deterministic Algorithm and Randomized Algorithm

Before we continue, I'd like to introduce some basic ideas about deterministic algorithm and randomized algorithm. A deterministic algorithm is an algorithm which, given a particular input, will always produce the same output. In other words, if the input data is fixed, the result will always be the same. A randomized algorithm is an algorithm which employs a degree of randomness as part of its logic. We will show how randomness is applied in our algorithms in later sections.

3 Communication Model

3.1 Introduction

Let X, Y, Z be arbitrary finite sets and let $f : X \times Y \to Z$ be an arbitrary function. There are two players, Alice and Bob, who wish to evaluate $f(x, y)$ for some inputs $x \in X$ and $y \in Y$. The difficulty is that Alice only knows x and Bob only knows y. Thus, to evaluate the function, they will need to communicate with each other. The communication will be carried out according to some fixed protocol P (which depends only on f) . The protocol consists of the players sending bits to each other until the value of f can be determined.

At each stage, the protocol P (for the function f) must determine whether the run terminates; if the run has terminated, the protocol must specify the answer given by the protocol (that is, $f(x, y)$); and if the run has not terminated, the protocol must specify which player sends a bit of communication next. This information must depend solely on the bits communicated so far during this run of the protocol, because this is the only knowledge common to both Alice and Bob. In addition, if it is Alice's turn to speak (that is, to communicate a bit), the protocol must specify what she sends; this depends on the communication so far as well as on x, the input visible to Alice. Similarly, if it is Bob's turn to speak, the protocol must specify what he sends; this depends on the communication so far and on y, his input.

We are only interested in the amount of communication between Alice and Bob, and we wish to ignore the question of the internal computations each of them makes. Thus, we allow Bob and Alice to have unlimited computational power. The cost of a protocol P on input (x, y) is the number of bits communicated by P on input (x, y). The cost of a protocol P is the worst case (that is, maximal) cost of P over all inputs (x, y). The complexity of f is the minimum cost of a protocol that computes f.

3.2 Best-Order Streaming Model

In this model, an input is in some order $e_1, e_2, ..., e_m$, where m is the size of the input. The input $e_1, e_2, ..., e_m$ could be numbers, edges, or any other items. In this article, we are interested in the case where they are edges. We will assume this implicitly throughout. Moreover, we assume that the input element is indivisible (e.g., vertices in e_i must appear consecutively). In the case of graph problems considered in this chapter, we also assume that the number of vertices is known to the algorithm before reading the stream.

Consider any function f that maps the input stream to $0, 1$. The goal of the typical one-pass streaming model is to develop an algorithm that uses small space to read the input in order $e_1, e_2, ..., e_m$ and calculate $f(e_1, e_2, ..., e_m)$.

In the best-order streaming model, we consider any function f that is order-independent. That is, for any permutation π

$$f(e_1, e_2, ..., e_m) = f(e_{\pi_1}, ..., e_{\pi_m})$$

Note that many graph properties (including those considered in this chapter) satisfy

the above property. Our main question is how much space a one-pass streaming algorithm needs in order to compute f if the input is provided in the best order possible. Formally, for any function $s(m)$ and any function f, we say that a language L,determined by f is in the $Stream - Proof(s(m))$ lass if there exists a streaming algorithm A with space $s(m)$ such that
1.if $f(e_1, e_2, ..., e_m) = 1$ then there exists a permutation π such that $A(e_{\pi_1}, ..., e_{\pi_m})$ answers 1;
2.otherwise, $A(e_{\pi_1}, ..., e_{\pi_m})$ answers 0 for every permutation π.

The other way to interpret this model is to consider the situation where there are two players in the setting, the prover and the verifier. The job of the prover is to provide the stream in some order so that the verifier can compute f using the smallest amount of memory possible. We assume that the prover has unlimited power but restrict the verifier to read the input in a streaming manner.The model above can be generalized to the following models.
1.$Stream(p, s)$: A class of problems that, when presented in the best-order, can be checked by a deterministic streaming algorithm A using p passes and $O(s)$ space.
2.$RStream(p, s)$: A class of problems that, when presented in the best-order, can be checked by a randomized streaming algorithm A using p passes and $O(s)$ space.

It is important to point out that when the input is presented in a specified order, we still need to check that the oracle is not cheating. That is, we indeed need a way to verify that we receive the input based on the rule we asked for. This often turns out to be the difficult step.

To contrast this model with the well-studied communication complexity models, we first define a new communication complexity model called magic-partition communication complexity. We later show a relationship between this model and the best-order streaming model.

3.3 Magic-Partition Communication Complexity

Recall the following standard 2-player communication complexity which we call worst-partition communication complexity. In this model, an input S, which is the set of elements, is partitioned into two sets X and Y , which are given to Alice and Bob, respectively. Alice and Bob want to together compute $f(S)$, for some order independent function f. In the worst-partition case, we consider the case when the input is partitioned in an adversarial way,i.e.we partition the input into X and Y in such a way that Alice and Bob have to communicate as many bits as possible.

The magic-partition communication complexity consists of three players, the oracle, Alice and Bob. An algorithm on this model consists of a function O(owned by the oracle) that partitions the input set $S = e_1, e_2, ..., e_m$ to two sets $X = e_{\pi(1)}, e_{\pi(2)}, ..., e_{\pi(m/2)}$ and $Y = e_{\pi(m/2+1)}, ..., e_{\pi(m)}$ for some permutation π and a protocol P used to communicate between Alice and Bob. We say that an algorithm consisting of O and P has communication complexity $c(m)$, for some function c, if
1.For an input S such that $f(S) = 1$, the protocol P uses $c(m)$ bits of communication and outputs 1 when it is run on the sets X and Y partitioned according to O
2.For an input S such that $f(S) = 0$, the protocol P uses $c(m)$ bits of communication and outputs 0 when it is run on any sets X and Y coming from any partition.

One way to think of this protocol is to imagine that there is an oracle who looks at the input and then decides how to divide the data between Alice and Bob so that they can compute f using the smallest number of communicated bits and Alice and Bob have to also check if the oracle is lying. We restrict that the input data must be divided equally

between Alice and Bob.

4 Checking for Distinctness

Consider the following problem: Given a stream of n numbers $a_1, a_2, a_3, ...$, where $a_i \in 1, 2, ..., n$. We want to check if every number appears exactly once. We are interested in solving the problem in the worst-order streaming model. Even though such a problem seems trivial, it is crucial in solving all the problems list in this article. The algorithm for this problem is an important ingredient of all algorithms in this article. We give the result:
1. Any deterministic algorithm for this problem needs $\Omega(n)$ space.
2. There exists a randomized algorithm that solves this problem in $O(\log n)$ space with an error probability at most $\frac{1}{n}$.

4.1 Space lower bound of the deterministic algorithm

Define a variation of DISTINCT, which is called full set disjointness problem, denoted by F-DISJ.
In this problem, a set $X \subseteq [n]$ is given to Alice and a set $Y \subseteq [n]$ is given to Bob where [n]=1,2,3,...,n and $|X|+|Y| = n$. Alice and Bob want together check whether $X \cap Y = \oslash$. Then we have the following theorem:
Theorem 1. The communication complexity of F-DISJ is $\Omega(n)$.
Before we prove this theorem, based on the communication model above, and then use this model to figure the communication complexity out.
Definition 2: A protocol V over domain $X \times Y$ with range Z is a binary tree where each internal node v is labeled either by a function a_v:$X \to 0, 1$ or by a function b_v: $Y \to 0, 1$, and each leaf is labeled with an element z∈Z.
Definition 3: For a function f: $X \times Y \to Z$, the (deterministic) communication complexity of f is the minimum cost of V, over all protocols V that compute f. We denote the (deterministic) communication complexity of f by $D(f)$.

The simplest way for Alice and Bob to evaluate a function f is for one of the players, say Alice, to send all her input to Bob (requiring $\log_2 |X|$ bits using an appropriate encoding), for Bob to compute $f(x, y)$ privately (with his unlimited computational power), and then for Bob to send the answer back to Alice ($\log_2 |Z|$ more bits). We thus have:
Proposition 4 For every function $X \times Y \to Z$,

$$D(f) \leq \log_2 |X| + \log_2 |Z|$$

Definition 5: Let $f : X \times Y \to 0, 1$. A set $S \subset X \times Y$ is called a fooling set (for f) if there exists a value $z \in 0, 1$ such that:
1. For every $(x, y) \in S$, $f(x, y) = z$
For every two distinct pairs (x_1, y_1) and (x_2, y_2) in S, either $f(x_1, y_2) \neq z$ or $f(x_2, y_1) \neq z$.
Lemma 6: If f has a fooling set S of size t, then $D(f) \geq log_2 t$.

Lemma 6 has already been proven in [3]. Here we only list the result.

Then come back to Theorem 1. Define a fooling set $F(A_1, B_1), (A_2, B_2), ..., (A_k, B_k)$ of size k such that $f(A_i, B_i) = 1$ for all i and $f(A_i, B_j) = 0$ for all $i \neq j$. Once this is shown, it will follow that the deterministic communication complexity is $\Omega(log|F|)$.
Now, consider the fooling set $F = (A, N/A) : \forall \subseteq N$. It is easy to check that the property above holds. Then, it is easy to show that $|F| = 2^n$, the number of bits needed to sent between Alice and Bob is at least $\log |F| = \Omega(n)$.

4.2 Space for Randomized Algorithm

In this subsection, we present a randomized one-pass worst-order streaming algorithm that solves DISTINCT using $O(logn)$ space. This algorithm is based on the Finger-printing Sets technique by Lipton.Roughly speaking, given a multi-set $x_1, x_2, ..., x_k$, its fingerprint is defined to be

$$\prod_{i=1}^{k}(x_i + r)$$

where p is a random prime and $r \in 0, 1, ..., p-1$. We use the following property of the fingerprints.

Theorem 7. Let $x_1, x_2, ..., x_k$ and $y_1, y_2, ..., y_l$ be two multi-sets. If the two sets are equal, then their fingerprints are always the same. Moreover, if they are unequal, the probability that they get the same fingerprints is at most

$$O(\frac{logb + logm}{bm} + \frac{1}{b^2m})$$

where all numbers are b bits numbers and $m = max(k, l)$ provided that the prime p is selected randomly from the interval

$$[(bm)^2, 2(bm)^2]$$

Before we prove theorem 3.6, we do some pre-work.

Definition 8 The height $H(\Phi)$ of a polynomial $\Phi(x) = \Phi x^d + ... + \Phi_0$ is the maximum of $|\Phi_d|, ..., |\Phi_0|$.

Lemma 9

(1) Let $\Phi_1(x)$ and $\Phi_2(x)$ be polynomials. Then, $H(\Phi) \leq H(\Phi_1) + H(\Phi_2)$.

(2) Let $\Phi(x) = (x - a_1)(x - a_2)...(x - a_m)$, then

$$H(\Phi) \leq (1 + |a_1|)...(1 + |a_m|)$$

The proof is obviously.

We now turn to the proof of theorem 6. Let $f(u) = (u - x_1)...(u - x_k)$ and $g(u) = (u - y_1)...(u - y_l)$ be polynomials in u. Also, let $\Phi(u) = f(u) - g(u)$. Since $x_1, x_2, ..., x_k$ and $y_1, y_2, .., y_l$ are unequal as multisets, the polynomial $\Phi(u)$ is not identically zero. Clearly, its degree is at most m. By the lemma its height is at most:

$$\prod_{i=1}^{k}(1 + |x_i|) + \prod_{j=1}^{l}(1 + |x_j|)$$

Since each number is at most n bit it follows that $H(\Phi)$ is at most $2^{mn} + 1$. Now let $t = (nm)^2$ and let p be a random prime with $t < p < 2t$ and let r be a random residue modulo p.

We must now show that it is unlikely that $\Phi(r) \equiv 0$ mod p since this implies that $x_1, x_2, ..., x_k$ and $y_1, y_2, ..., y_l$ have different fingerprints. Since Φ is not identically 0 it has at least one coefficient say c that is not 0. By the lemma, c is at most $2^{mn} + l$. Therefore, c has at most $nm + 1$ prime factors. Thus, the probability that $c = 0$ mod p is at most

$$O(\frac{nm}{\frac{t}{log(t)}})$$

since there are approximately $\frac{t}{log(t)}$ primes in the given interval. Since $t = (mn)^2$, it follows that this probability is

$$O(\frac{logb + logm}{bm})$$

This method of computing fingerprints has a number of advantages. First, it can be computed in at most the amount of space needed to store the fingerprints. Second, it can be computed in at most the time required to multiply numbers of this size. Just as important it can be computed "on-line", i.e. the fingerprint can be computed in a single pass over the multi-sets. This last property is critical for our applications. Thus, the prime p and hence the fingerprint has at most $O(log(b) + log(m))$ bits.

Next we need to compute the probability that $\Phi(x) \equiv 0$ mod p given that $c \neq 0$ mod p. Since Φ is not identically zero it follows that it has at most m roots. Thus, the probability that $\Phi(r) \equiv 0$ mod p is at most $\frac{m}{p}$ or $\frac{1}{n^2 m}$. Therefore, the total probability of the two sets have the same fingerprint is at most

$$O(\frac{logb + logm}{bm} + \frac{1}{b^2 m})$$

Now, to check if $a_1, a_2, ..., a_n$ are all distinct, we simply check if the fingerprints of $a_1, a_2, ..., a_n$ and $1, 2, ..., n$ are the same. Here, $b = logn$ and $m = n$. Therefore, the error probability is at most 1/n.

Remark We note that the fingerprinting sets technique can also be used in our motivating application of cloud computing above. That is, when the cloud sends back a graph as a proof, we have to check whether this proof graph is the same as the input graph we sent. This can be done by checking if the fingerprints of both graphs are the same. This enables us to concentrate on checking the stream without worrying about this issue in the rest of this essay.

5 Randomized Algorithm in Perfect Matching

5.1 Perfect Matching

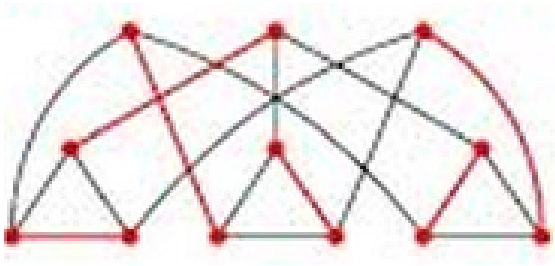

Figure 3: Perfect Matching

A perfect matching is a matching which matches all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching. Figure 3 above is an example of a perfect matching. We exhibit the ideas of developing algorithms and lower bounds in the best-order streaming model through the perfect matching problem.

Problem Let G be an input graph of n vertices where the vertices are labeled $1, 2, ..., n$. Given the edges of G in a streaming manner $e_1, e_2, ..., e_m$, we want to compute $f(e_1, ..., e_m)$ which is 1 if and only if G has a perfect matching. Let n be the number of vertices.

5.2 Upper Bound

Theorem 10. The problem of determining if there exists a perfect matching can be solved by a randomized best-order streaming algorithm using $O(log(n))$ space with a success probability at least $1 - \frac{1}{n}$.

Proof: We may use the following algorithm: The prover sends n/2 edges of a perfect matching to the verifier first and then send the rest of the edges. The verifier then check the followings:

1. Check if the first $\frac{n}{2}$ edges form a perfect matching. This can be done by checking

whether the fingerprint of the set $\bigcup_{i=1}^{n/2} e_i$(where $e_1, e_2, ..., e_{n/2}$) are the first half n edges in the stream) is equal to the fingerprint of the set 1,2,...,n.
2. Check if there are at most n vertices. This is done by checking that the maximum vertex label is at most n.
Finally, the verifier outputs 1 if the input passes all the above tests.
Then if the edges $e_1, e_2, ..., e_{n/2}$ form a perfect matching then $e_1, e_2, ..., e_{n/2}$ have no vertex in common and therefore, $\bigcup_{i=1}^{n/2} e_i = 1, 2, ..., n$. Their fingerprints are also the same. And for the case that the edges $e_1, e_2, ..., e_{n/2}$ do not form a perfect matching, observe that $\bigcup_{i=1}^{n/2} e_i \neq 1, 2, ..., n$ and therefore the fingerprints of the two sets will be different with probability at least $1 - \frac{1}{n}$. Consequently, the algorithm will successfully output 0 with probability at least $1 - \frac{1}{n}$.

5.3 Lower Bound

Theorem 11 If the input can be reordered only in an explicit way, then any deterministic algorithm solving the perfect matching problem needs $\Omega(n)$ space, where n is the number of vertices.
Proof: Let n be any even integer divisible by four. We show that the above theorem is true even when the input always contains exactly n/2 edges. In this case, checking whether these n/2 edges form a perfect matching is equivalent to checking whether every vertex appears as an end vertex of exactly one edge. We note that the input is allowed to contain multiple edges. We now show that the magic-partition (which means the input is partitioned in the best way possible) communication complexity if the perfect matching problem is $\Omega(n)$.

Consider any magic-partition communication complexity protocol which consists of a partition function O owned by an oracle and a communication protocol P between Alice and Bob. That is, a function O partitions the input into two sets of edges, A and B where $|A| = n/4$ and $|B| = n/4$. Then, A and B are sent to Alice and Bob, respectively. Alice and Bob, upon receiving A and B, communicate to each other using a protocol P and one of them outputs whether the input edges form a perfect matching or not (YES or NO). The main goal is to show that for any partition function O, there is some input that forces P to incur $\Omega(n)$ bits of communication. Recall that P has to deal with the following cases:
1) If the input is a perfect matching, P has to output YES when the input is partitioned according to O.
2) Otherwise, P has to output NO for any partition of the input. We now show the communication complexity of P.

First, let us consider the inputs that are perfect matchings. Let $g(n)$ denote the number of distinct perfect matchings in the complete graph K_n. Observe that:

$$g(n) = \frac{C_n^2 C_{n-2}^2 ... C_2^2}{(n/2)!} = \frac{n!}{(n/2)! 2^{n/2}}$$

Denote these matchings by $M_1, M_2, ..., M_{g(n)}$. For any integer i, let A_i and B_i be the partition of M_i according to O. We now partition $M_1, ..., M_{g(n)}$ into clusters in such a way that the matchings whose vertices are partitioned in the same way are in the same cluster. That is, any two inputs M_i and M_j are in the same cluster if and only if $\cup_{e \in M_i} e = \cup_{e \in M_j} e$.

We claim that there are at least $C_{n/4}^{n/2}$ clusters. To see this, observe that for any matching M_i, there are at most $g(n/2^2)$ matchings that vertices could be partitioned the same way as M_i. Therefore, the number of matchings such that the vertices are divided differ-

ently is at least

$$\frac{g(n)}{g(n/2)^2} = \frac{n!}{(n/2)!2^{n/2}}(\frac{(n/4)!2^{n/4}}{(n/2)!})^2 = \frac{C_n^{n/2}}{C_{n/2}^{n/4}} \geq C_{n/2}^{n/4}$$

Let t be the number of clusters (so $t \geq C_{n/2}^{n/4}$ and let $M_{i_1}, M_{i_2}, ..., M_{i_t}$ be the inputs from different clusters and let $(A_{i_1}, B_{i_1}), ..., (A_{i_t}, B_{i_t})$ be the corresponding partitions according to O. Observe that for any $t' \neq t''$, an input consisting of edges in $M_{i_t'}$ and $M_{i_t''}$ is not a perfect matching. Moreover, observe that for any t' and t'', any pair $(A_{i_t'}, B_{i_t''})$ could be an input to the protocol P (since the oracle can partition the input in anyway when the input is not a perfect matching). In other words, the communication complexity of P is the worst case (in term of communication bits) among all pairs $(A_{i_t'}, B_{i_t''})$.

Let $t' = \log t$ (Note that $t' = \Omega(n)$.) Consider the problem $EQ_{t'}$ where Alice and Bob each gets a t' bit vector x and y, respectively. They have to output YES if $x = y$ and NO otherwise. The deterministic worst-partition communication complexity of $EQ_{t'}$ is at least $t' + 1 = \Omega(n)$. The proof is shown below:
Proof: Alice and Bob each hold an n-bit string, $x, y \in 0, 1^n$. The equality function,$EQ(x, y)$,is defined to be 1 if $x = y$ and 0 otherwise. A fooling set of size 2^n is

$$S = (\alpha, \alpha)|\alpha \in (0, 1)^n$$

(because for every a, $EQ(\alpha, \alpha) = 1$, whereas for every $\alpha \neq \beta$, $EQ(\alpha, \beta) = 0$). It follows that $D(EQ) > n$. By also counting 0-rectangles, we conclude $D(EQ) > n + 1$. Finally, recall that $D(f) < n + 1$(By Proposition 4) for every function $f : (0, 1)^n \times (0, 1)^n \to 0, 1$ (by Proposition 4). Therefore, $D(EQ) = n + 1$.

Just let $n = \log t'$, the proof can be finished.

Now we reduce $EQ_{t'}$ to our problem using the following protocol P': Upon receiving x and y, Alice and Bob locally map x to A_{i_x} and y to B_{i_y} , respectively and then simulate P. Since x = y if and only if $A_{i_x} \bigcup B_{i_y}$ is a perfect matching, P' outputs YES if and only if x = y. Therefore, Alice and Bob can use the protocol P' to solve $EQ_{t'}$. Since, the deterministic worst-partition communication complexity of $EQ_{t'}$ is $\Omega(n)$, so is the communication complexity of P. This shows that the deterministic magic-partition communication complexity of the matching problem is $\Omega(n)$.

Note that the above lower bound is asymptotically tight since we can check if there is a perfect matching using O(n) space in the best-order streams: The oracle simply puts edges in the perfect matching first in the stream. Then, the algorithm checks whether the first n/2 edges in the stream form a matching by checking whether all vertices appear (using an array of n bits).

However, the problem of the above argument is that the protocol in Theorem 7.9 needs the worst-partition communication complexity of Distinct. In fact, Distinct can be easily solved in 1 bit using the following magic-partition communication complexity protocol: The oracle sends the first n/2 smallest numbers to Alice and sends the rest to Bob. Alice sends 1 to Bob if her numbers are $1, 2, ..., n/2$ and Bob outputs YES if he receives 1 from Alice and his numbers are $n/2 + 1, n/2 + 2..., n$.

6 Randomized in Graph Connectivity

Problem We consider a function where the input is a set of edges and $f(e_1, e_2, ..., e_m)$=1 if and only if G is connected. As usual, let n be the number of vertices of G. As before, we assume that vertices are labeled $1, 2, 3, ..., n$.

6.1 Upper Bound

Theorem 12. Graph connectivity can be solved by a randomized algorithm using $O((\log n)^2)$ space in the best-order streaming model.
Proof: We use the following lemma constructively.
Theorem 13. For any graph G of $n-1$ edges, where $n \leq 3$, G is connected if and only if there exists a vertex v and trees $T_1, T_2, ..., T_q$ such that for all i
1.$\cup_{i=1}^{q} V(T_i) = V(G)$ and for any $i \neq j$, $V(T_i) \cap V(T_j) = v$
2.there exists a unique vertex $u_i \in V(T_i)$ such that $u_i v \in E(T_i)$
3.$|V(T_i)| \leq [2n/3]$ for all i.
Proof: Notice that G is a spanning tree since it is connected and has exactly $n-1$ edges. Consider any vertex in the spanning tree, which on deleting, disconnects the graph in to two or more pieces such that each piece has at most 2n/3 vertices. The existence of such a vertex can be proven by induction. The base case where n = 3 can be proved simply by picking a vertex in the middle of the path of length 3.
Suppose such a vertex exists for all $n' < n$. Consider a tree on n vertices. Now, remove

Figure 4: Spanning tree with three nodes

a leaf node, say z, and, by induction, such a vertex v exists on the tree on remaining n vertices. Now add z back and let C be the component (on deleting v) that contains z.

If C has size at most $[2n/3] - 1$, the same v works on the larger tree. If C has size at

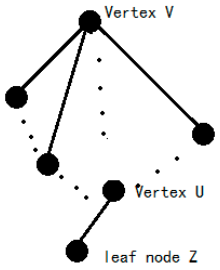

Figure 5: Spanning tree with n nodes

least $[2n/3]$ then consider the unique vertex in this component that connects to v, say u. Observed that u serves as the new vertex for the lemma. This is because the complement of C together with u has size at most $n - [2n/3] + 1 \leq 2n/3$. Since this can be done, u can be chosen as a vertex for the lemma. Whenever a vertex in a tree is disconnected, the new components also form trees. Call these $T_1, T_2, ...T_q$. Notice that there can be at most one vertex adjacent to v in each component T_i, since G has no cycles. Call this vertex in T_i by u_i. Therefore each $u_i \in T_i$ is adjacent to u and each T_i has at most $[2n/3]$ nodes.

It is sufficient to consider graphs of $n-1$ edges, as these $n-1$ edges that form a connected spanning component are sufficient to verify connectivity. Suppose that G is connected,i.e.G is a tree. Let v and $T_1, T_2, ..., T_q$ be as in the lemma. Define the order of G to be

$$Order(G) = vu_1, Order(T_1'), vu_2, Order(T_2'), ..., vu_q, Order(T_q')$$

where $T_i' = T_i / vu_i$. Note that T_i' is a connected tree and so we present edges of T_i' recursively. The recursion step ends when the eventual subtree is a star, i.e., edges presented

are $vu_1, vu_2, ...$ At this point, the verifier just checks that all consecutive edges are adjacent to the same vertex and form a star. This depth of recursion can be checked directly.

Now, when edges are presented in this order, the checker can check if the graph is connected as follows. First, the checker reads vu1. The checker remembers the vertex v, which takes $O(\log n)$ bits. Then the edges in T_i' are presented. He checks if T_i' is connected by running the algorithm recursively. Note that he stops checking T_i' once he sees vu_2. Notice that this step is consistent since the vertex v does not appear in any T_i' . Once an edge with a vertex v is received, the checker knows that the tree has been verified and the next tree is to be presented. So the checker repeats with vu_2 and T_2' and so on. Here again, v does not appear in T_2' but u_2 does. Therefore, the checker now again needs to check that T_2' is connected. Further, it is automatically checked that T_2' connects to v due to the edge vu_2. The checker proceeds in this manner checking the connectivity of each T_i' up to $i = q$. If each tree is connected (which is checked recursively), and all the edges vu_i appear separating the trees, then all the trees are connected to v. Therefore, the entire set of edges presented is connected. However, this does not guarantee that n distinct vertices, or n distinct edges have been received. Therefore, it only remains to be checked that n distinct edges have been presented.

If all n distinct vertices have appeared at least once, and the set of first n edges form a connected component, then G is a connected graph. Also note that if G is not connected then such ordering cannot be made and the algorithm above will detect this fact.

The space needed is for vu_i and for checking T_i'. I.e.

$$space(|G|) = space(max_i|T_i|) + O(\log n)$$

That is, $space(n) \leq space(2/3n) + O(\log n)$. By iteration, it is easy to check that the upper bound of space is $O((\log n)^2)$.

6.2 Lower Bound

Theorem 14. If the input can be ordered only in an explicit way, any deterministic algorithm solving the connectivity problem on the best-order stream needs $\Omega(n \log n)$space, where n is the number of vertices.

Proof. Let n be an odd number. We show that the theorem holds even when the input always consists of exactly $n - 1$ edges. (Therefore, the task is only to check whether the input edges form a spanning tree over n nodes.) We show this via the magic-partition communication complexity. The argument is essentially the same as that in the the proof of lower bound of perfect checking, here only give the essential parts.

Assume that n is an odd number more than two. Let $g(n)$ be the number of spanning trees of the complete graph K_n. Then comes Cayleys formula:

For any positive integer n, the number of trees on n labeled vertices is n^{n-2}

Therefore, $g(n) = n^{n-2}$. Let $T_1, T_2, ..., T_{g(n)}$ denote such trees. Consider any best-partition communication complexity protocol which consists of a partition function O owned by the oracle and a protocol P used by Alice and Bob. For $i = 1, 2, ..., g(n)$, let (Ai, Bi) be the partition of the input edges of T_i to Alice and Bob, respectively, according to O.

Now, draw a graph H consisting of $g(n)$ vertices, $v_1, v_2, ..., v_{g(n)}$. Draw an edge between vertices v_i and v_j if $A_i \cup B_j$ or $A_j \cup B_i$ is a spanning tree.

We claim that each vertex in H has degree at most $2g((n+1)/2)$. To see this, observe that for any set A of $(n-1)/2$ edges, there are at most $g((n+1)/2)$ sets B of $(n-1)/2$ edges such that $A \cup B$ is a spanning tree. This is because when we contract edges in A, there are $(n+1)/2$ vertices left and these vertices must form a spanning tree on the contracted graph. This observation is also true when we look at the set B. The claim thus

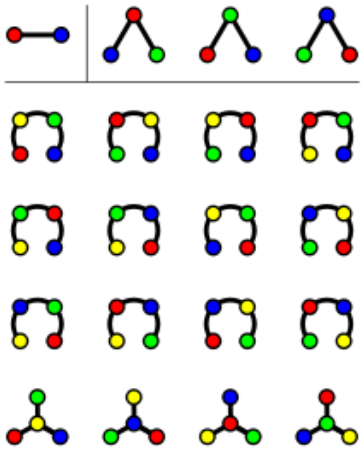

Figure 6: Trees on n labeled vertices

follows. Now pick an independent set from H using the following algorithm: Pick any vertex, delete such vertex and its neighbors and then repeat. Observe that this algorithm gives an independent set of size at least $\frac{g(n)}{2g((n+1)/2)+1}$ since there are $g(n)$ vertices in H and each vertex has degree at most $2g((n+1)/2)$. Let t be the size of the independent set. Note that $t = \frac{g(n)}{2g((n+1)/2)+1} = n^{\Omega(n)}$.Let $v_{i_1}, v_{i_2}, ..., v_{i_t}$ be the vertices in the independent set.

Consider the trees $T_{i_1}, T_{i_2}, ..., T_{i_t}$ corresponding to the independent set picked by the above algorithm. Since there is no edge between any $v_{i'_t}$ and $v_{i''_t}$, $(A_{i'_t}, B_{i''_t})$ and $(A_{i''_t}, B_{i'_t})$ do not form a spanning tree. As argued in the proof of Theorem 11, the protocol P must be able to receive any pair of the form $(A_{i'_t}, B_{i''_t})$ and answer YES if and only if $t' = t''$. By the fooling set argument or the reduction from $EQ_{\log t}$, it follows that P needs $\Omega(\log t) = \Omega(n \log n)$ bits, as mentioned above.

Note that the lower bound above is asymptotically tight since we can solve connectivity problem using the following $O(n \log n)$ space deterministic algorithm: The oracle present edges in a spanning tree first in the stream. Then the algorithm reads and checks whether these edges form a spanning tree using $O(n \log n)$ space.

7 Hamiltonian Cycle

The problem is to check whether the input graph has a Hamiltonian cycle.
Theorem 15. In randomized algorithm, the problem can be solved in $O(\log n)$ space.
Proof: The intuition is for the oracle to provide the Hamiltonian cycle first (everything else is ignored). The algorithm then checks if the first n edges indeed form a cycle; this requires two main facts. First that every two consecutive edges share a vertex, and the $n-th$ edge shares a specific vertex with the first. This fact can be easily checked. The second key step is to check that these edges indeed span all n vertices (and not go through same vertex more than once). This can be done by the fingerprinting technique similarly as mentioned above.
Theorem 16. There is an $\Omega(n)$ lower bound for the deterministic algorithms if the edges can be ordered only in an explicit way.
Proof: The proof is essentially similar to the proof of other lower bounds shown earlier and we only sketch it here. We consider the inputs that have exactly n edges. Let $g(n)$

be the number of the Hamiltonian cycles covering n vertices. Clearly $g(n) = (n-1)!$. Let $C_1, C_2, ..., C_{g(n)}$ be these circles and $(A_1, B_1), ..., (A_{g(n),B_{g(n)}})$ be the corresponding partitions. Since after we see n/2 edges, there could be only $g(n/2$ possible Hamiltonian cycles containing these edges, we can pick $\frac{g(n)}{2g(n/2)+1}$ that are independent. The communication complexity is thus

$$\Omega(\log(\frac{g(n)}{2g(n/2)+1})) = \Omega(n)$$

8 Conclusion

In this article, we describe a new kind of algorithm which reduce the space for communication effectively. Although such a algorithm may cause some mistakes, the possibility to make a mistake is very low, only about $\frac{1}{n}$. When the number of nodes is very large, such a low probability can be ignored. The motivation of this algorithm is based on the growing in the data size and the advent of the powerful cloud computation architectures and services.

Fingerprint is used as a crucial part in this algorithm. As mentioned above, applying such a randomized algorithm can definitely decrease communication complexity. In later fork, I may apply such an algorithm in some other graph problems, such as checking whether a graph is a planar graph, check whether a graph is a bipartiteness graph. What's more, this algorithm only allow two passes. It is also interesting to verify whether applying more passes can benefit more remains to be checked.

9 Reference

[1]Nanongkai D. Graph and geometric algorithms on distributed networks and databases[D]. Georgia Institute of Technology, 2011.
[2]Yao,A.C.-C.,"Some complexity questions related to distributive computing," in STOC, pp.209−213,1979.
[3]Kushilevitz, E. and Nisan, N., Communication complexity.New York,NY,USA:Cambridge University Press, 1997.
[4]Lipton,R.J.,"Efficient checking of computations," in STACS, pp.207-215,1990.
[5]Aigner,M.and Ziegler,G. M.,Proofs from THE BOOK. Springer, 3rd,November 2003.