Streaming and Online algorithms

Michel de Rougemont
IRIF-CNRS

Abstract

We describe Streaming and Online algorithms as a new model of computation. We present algorithms for the moments of a stream of values, for graph properties and for properties of balanced words. The competitive ration is the important measure for the quality of Online algorithms. We present Online Bipartite matching as an application.

Contents

1 Introduction 3
2 Reservoir Sampling 3
3 Moments in a stream of values 4
  3.1 Morris Algorithm for estimating $F_1$ 4
  3.2 Estimating $F_0$ 4
  3.3 Basic estimator for $F_2$ 5
    3.3.1 Analysis 5
    3.3.2 Implementation with 4-wise independent Hashing functions 7
  3.4 Lower bound for $F_\infty$ 7
4 Graph properties from a stream of edges 8
  4.1 Graph properties by sampling [16] 8
  4.2 Graph properties in a stream 12
    4.2.1 Size of the graph 12
    4.2.2 Communities and dense subgraphs 12
    4.2.3 Uniform Spanning trees 16
5 Regular properties of a stream of Balanced words [14] 17
  5.1 Testing is hard, i.e. it requires $O(n^\alpha)$ queries 18
  5.2 Streaming Property Testing in $O(poly(\log n))$ 19
    5.2.1 Final result 20
6 Online algorithms

6.1 Bipartite matching ............................................................... 21

6.1.1 Competitive ratio of the Greedy’s algorithm ....................... 21

6.1.2 A better expected Competitive ratio ................................. 22
1 Introduction

Streaming algorithms have become very important, as data from many fields can be accessed as a stream: astronomy, medicine, social networks, ... We are interested in algorithms which read data step by step and maintain a small memory, if possible constant or $poly(\log)$ in the size of the stream.

Streaming algorithms read an input stream of length $n$ and give an answer at the end. Online algorithms must provide an answer at each stage $i$. In some cases a prefix of a stream is not a valid input. It is the case for balanced words or trees. In this case we only consider streaming algorithms.

We first consider a stream $s = x_1, x_2, ..., x_n$ of numerical values $x_i \in \{1, 2, ..., m\}$, then a stream of edges $e = (v_i, v_j)$ in a graph $G = (V, E)$ and finally balanced words. For online algorithms, we consider a stream of edges and consider the Bipartite matching problem. In this case all the edges adjacent to a node $u$ arrive contiguously in the stream.

2 Reservoir Sampling

A classical technique, introduced in [23] is to sample each new element $x_n$ of a stream $s$ with some probability $p$ and to keep it in a set $S$ called the reservoir which holds $k$ tuples. It also applies to values with a weight. Let $s = x_1, x_2, ..., x_n$ be the stream and let $\hat{S}_n$ be the reservoir at stage $n$. We write $\hat{S}$ to denote that $S$ is a random variable.

$k$-Reservoir sampling: $A(s)$

- Initialize $S_k = \{x_1, x_2, ..., x_k\}$,
- For $j = k + 1, ..., n$, select $x_j$ with probability $k/j$. If it is selected, replace a random element of the reservoir (with probability $1/k$) by $x_j$.

Lemma 1 Let $\hat{S}_n$ be the reservoir at stage $n$. Then for all $n > k$ and $1 \leq i \leq n$:

$$\text{Prob}[x_i \in \hat{S}_n] = k/n$$

Proof: Let us prove by induction on $n$. The probability at stage $n+1$ that $x_i$ is in the reservoir $\text{Prob}[x_i \in \hat{S}_{n+1}]$ is composed of two events: either the tuple $x_{n+1}$ does not enter the reservoir, with probability $(1 - k/(n+1))$ or the tuple $x_{n+1}$ enters the reservoir with probability $k/(n+1)$ and the tuple $x_i$ is maintained with probability $(k-1)/k$. Hence:

$$\text{Prob}[x_i \in \hat{S}_{n+1}] = k/n((1 - k/(n+1)) + k/(n+1) .(k-1)/k)$$

$$\text{Prob}[x_i \in \hat{S}_{n+1}] = k/n(n+1-k)/(n+1) + (k-1)/(n+1)$$

$$\text{Prob}[x_i \in \hat{S}_{n+1}] = k/(n+1)$$
Interestingly, imagine we are interested in the most recent data. At every step some
values are outdated and leave the reservoir and new values appear. How do we generalize
the reservoir into a Window Reservoir? Some other techniques based on hashing are possible,
but the strict generalization of the classical reservoir is an interesting open problem.

3 Moments in a stream of values

Let $U = \{1, 2, \ldots, m\}$ be a set of elements and a data stream $s_n = x_1, \ldots, x_n$ of elements of $U$. Consider the frequency $f_i \in \{0, 1, 2, \ldots, m\}$ the number of occurrences of the element $i$ in the stream.

The $k$-th frequency moment $F_k = \sum_{j \in U} f_j^k$. Notice that $F_0$ is the number of distinct elements in the stream, $F_1 = n$, and as $k$ increases we give more weight to the most frequent element. It is then natural to define $F_\infty = \max_j f_j$.

3.1 Morris Algorithm for estimating $F_1$

A counter needs $O(\log n)$ space. Could we use less space? Morris answered this question in 1977 [20], with a surprising technique.

Assume $n = 2^i$, and let us store $i$. As $n$ increases we increase $i$ with probability $1/2^i$.

Given $i$ (which requires $O(\log \log n)$ space to store), we estimate $n$ as $Z = 2^i$. If $n = 2^i$, then $E[Z] = n$.

3.2 Estimating $F_0$

Let $h$ be a random function $h : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, M\}$. Typically, $M$ is small.

Algorithm $A_2(s_n)$:

- Let $h$ be a random function.
- $Min = h(x_1)$
- At every step, reading $x_i$, let $Min = \min\{Min, h(x_i)\}$

Return $X = M/Min$

Intuitively, the expected value if $n = 1$ of $Min$ is $M/2$ hence $X = 2$. If $n = 2$, then the expected value of $Min$ is $M/3$ hence $X = 3$. As $n$ increases the estimator is essentially
unbiased.

MinHash is a variation to estimate the Jaccard Similarity between two streams. Assume $A$ is the domain of the first stream and $B$ is the domain of the second stream. Let $J(A, B) = |A \cap B|/|A \cup B|$ the Jaccard Similarity. It is the probability that $X(A) = X(B)$. It can be approximated by the rate of common minimum values. Keep the 100 smallest values of $h(x_i)$ in both streams $A$ and $B$. If there are 10 values which are in common, then $J(A, B)$ is approximately 10%.

Count-MinSketch [9] is a generalization to approximate the different frequencies. Consider a matrix $A(i, j)$ with $w$ columns and $d$ rows. Each row $i$ is associated with hash function $h_i : U = \{1, 2, ...m\} \rightarrow \{1, 2, ...M\}$. Every time we read $x_k$, we compute

$$h_1(x_k), h_2(x_k), ..., h_w(x_k)$$

and we increase each value $A(i, h_i(x_k))$ by 1. If we want the frequency of $x_k$, we approximate it by the $\text{Min}_i\{A(i, h_i(x_k))\}$. If we take $w = 3/\varepsilon$ and $M = \log 1/\delta$, we approximate the frequency within an additive factor $\varepsilon$ with probability $1 - \delta$.

3.3 Basic estimator for $F_2$

The approach is to find an estimator whose expectation is $F_2$ and whose variance is bounded.

- Let $h : U \rightarrow \{-1, +1\}$ be a function which assigns a random sign to a an element of $U$.
- $Z = 0$. For each $j$ in the stream, $Z = Z + h(j)$.
- Return $X = Z^2$

Notice that $Z = \sum_j f_j.h(j)$. In order to estimate $F_2$, we first show that $X$ is unbiased and that the variance is bounded.

3.3.1 Analysis

Let us first analyse the expectation of $X$ and then the variance.

Lemma 2 $\mathbb{E}[X] = F_2$

Proof:

$$\mathbb{E}[X] = \mathbb{E}[Z^2]$$

$$\mathbb{E}[X] = \mathbb{E}[(\sum_j f_j.h(j))^2]$$

$$\mathbb{E}[X] = \mathbb{E}[\sum_j f_j^2 + 2\sum_{j<l} f_j.f_l.h(j).h(l)]$$
\[ E[X] = F_2 + 2 \sum_{j<l} E[\sum_{j<l} f_j.f_l.h(j).h(l)] \]

As \( E[h(j).h(l)] = 0 \):
\[ E[X] = F_2 \]

**Lemma 3** \( Var[X] \leq 2.F_2^2 \)

**Proof:** Recall that \( Var[X] = E[X^2] - (E[X])^2 \).
\[ E[X^2] = E[(\sum_{j} f_j.h(j))^4] \]

The terms \( h(j_1).h(j_2).h(j_3).h(j_4) \) have a 0 expectation, as all the terms where \( h(j_1) \) appears not squared. Hence:
\[ E[X^2] = E[(\sum_{j} f_j.h(j))^4] = E[\sum_{j} f_j^4.h(j)^4 + 6 \sum_{j<l} f_j^2.f_l^2.h(j)^2.h(l)^2] \]
\[ E[X^2] = \sum_{j} f_j^4 + 6 \sum_{j<l} f_j^2.f_l^2 \]
\[ F_2^2 = \sum_{j} f_j^4 + 2 \sum_{j<l} f_j^2.f_l^2 \]

Hence \( E[X^2] \leq 3.F_2^2 \) and \( Var[X] = E[X^2] - (E[X])^2 \leq 2.F_2^2 \).

In order to reduce the variance, we can make independent trials and average. Let \( Y = \frac{\sum_t X_t}{t} \), i.e. the average of \( t \) independent trials. Then \( E[Y] = E[X] \).

**Lemma 4** \( Var[Y] = \frac{Var[X]}{t} \)

**Proof:**
\[ Var[Y] = Var[\frac{\sum_t X_t}{t}] = \frac{\sum_t Var[X_t]}{t^2} = \frac{Var[X]}{t} \]

We can then guarantee the quality of the approximation of \( Y \).

**Lemma 5** If \( t = \frac{2}{\varepsilon.\delta} \), then \( Prob[Y \in [(1 - \varepsilon).F_2, (1 + \varepsilon).F_2]] \geq 1 - \delta \).

**Proof:** Recall Chebyshev’s inequality: \( Prob[|Y - F_2| \geq c] \leq \frac{Var(Y)}{c^2} \).

In our case \( c = \varepsilon.F_2 \), and using lemma 3, we get:
\[ Prob[|Y - F_2| \geq \varepsilon.F_2] \leq \frac{Var(Y)}{(\varepsilon.F_2)^2} \leq \frac{2.F_2^2}{t.(\varepsilon.F_2)^2} \]

Hence \( t = \frac{2}{\varepsilon.\delta} \).
3.3.2 Implementation with 4-wise independent Hashing functions

Suppose we try to implement the previous estimator. We need to store the values of \( h \) and it requires \( O(n) \) values. Can we do better?

We only need to have \( \mathbb{E}[h(j_1).h(j_2).h(j_3).h(j_4)] = 0 \), i.e. when we consider the 16 possibilities of 4-sequences of +1 and −1. It is the case if the distribution of the 16 values is uniform.

A family \( H \) of function \( h \) from \( U \to [1, 2, \ldots, m] \) is \( k \)-wise independent if for all \( k \) distinct \( x_1, \ldots, x_k \), \( \text{Prob}_{h \in H}[h(x_1) = y_1, h(x_2) = y_2, h(x_3) = y_3 \ldots h(x_k) = y_k] = 1/m^k \), i.e. uniform.

If \( m \) is prime, we may take a polynomial of degree 3 with 4 random values in \([1, 2, \ldots, m]\) as coefficients and map to the result modulo \( m \). Let us explain the construction on a 2-wise independent function:

\[
h(j) = a.j + b \pmod{m}
\]

where \( a, b \) are random on \([1, 2, \ldots, m]\). It generalizes to \( k \)-wise independence by taking a polynomial of degree \( k - 1 \).

**Lemma 6** If \( m \) is a prime number, \( h(j) = a.j + b \pmod{m} \) is 2-wise independent.

**Proof:** We need to evaluate for some fixed \( s, s' \):

\[
\text{Prob}_{h \in H}[h(j) = s \land h(j') = s'] = \frac{1}{m^2}
\]

If \( m \) is a prime number, then \( Z_m \) is a finite field. This system has unique solutions: \( a^* = \frac{s - s'}{j - j'} \) and \( b^* = s - \frac{a(s - s')}{j - j'} \).

If we take a random \( a \in [1, 2, \ldots, m] \) it has probability \( 1/m \) to be \( a^* \), and similarly for \( b \).

\[
\text{Prob}_{h \in H}[h(j) = s \land h(j') = s'] = \text{Prob}_{a, b}[a = a^* \land b = b^*] = \frac{1}{m^2}
\]

because if either \( a \neq a^* \) or \( b \neq b^* \) then either \( h(j) \neq s \) or \( h(j') \neq s' \). Hence:

\[
\text{Prob}_{h \in H}[h(j) = s \land h(j') = s'] = 1/m^2
\]

In the end, we need \( O\left(\frac{2}{\epsilon^2 \delta} \log n + \log m\right) \) space.

3.4 Lower bound for \( F_\infty \)

Contrary to \( F_0 \) and \( F_1 \), some moments such as \( F_\infty \) can’t be approximated.

**Lemma 7** \( F_\infty \) requires \( \Omega(n) \) space to be approximated by a randomized algorithm.
Proof: Assume Alice holds $x = x_1...x_n$, Bob holds $y = y_1...y_n$ and they try to compute DISJOINTNESS($x, y$). Assume a stream $s$ of values $i$ if $x_i = 1$ followed by a stream $s$ of values $i$ for $y_i = 1$. If $x_i = 0$ we do not choose $i$ in $x$, and similarly if $y_i = 0$, we do not take $i$ in $y$. The possible values for $F_\infty$ are 1 and 2, as soon as the stream is non empty.

If $F_\infty$ could be approximated in $O(s)$ space, the Communication Complexity of DISJOINTNESS would also be $O(s)$ space and we know that it is $\Omega n$. 

\[\square\]

4 Graph properties from a stream of edges

Let $G(V, E)$ be a symmetric graph with $n$ vertices and $m$ edges. Let $d_i$ the degree of node $i$.

4.1 Graph properties by sampling [16]

Consider a connected graph and an ergodic random walk. It has a stationary distribution $\pi$. For this distribution $\text{Prob}[x = v_i] = d_i/D = p_i$ where $d_i$ is the degree of the node $v_i$ and $D = \sum d_i$. Suppose we take $r$ nodes with the distribution $\pi$: \{${x_1, x_2,...x_r}$\}. Let $Y_{j,j'} = Y_{j,j'} = 1$ if $x_j = x_{j'}$ i.e. if there is a collision and 0 otherwise.

Consider the following variables which only depend on the samples. Let:

$$\psi_1 = \sum_{i=1,...r} d_i$$
$$\psi_{-1} = \sum_{i=1,...r} 1/d_i$$
$$C = \sum_{j \neq j'} Y_{j,j'}$$

The variable $C$ measures the number of collisions. Let $R = \psi_1 \psi_{-1} - r$.

A classical estimator based on the Birthday paradox is $r^2/2.C$ if the nodes are sampled uniformly. In [16], the authors propose a better estimator.

Algorithm to estimate $n$. Maintain $R$ and $C$ from $r$ independent samples taken with probability $d_i/D$. Output $\hat{n} = R/C$.

Lemma 8 $\mathbb{E}[R]/\mathbb{E}[C] = n$. 

8
Proof: Let us estimate $\mathbb{E}[C]$ and $\mathbb{E}[R]$:

$$\mathbb{E}[R] = \mathbb{E}[\psi_1 \cdot \psi_{-1} - r] = \mathbb{E}[\sum_{i=1,\ldots,r} d_i \cdot \sum_{i=1,\ldots,r} \frac{1}{d_i} - r] = \mathbb{E}[\sum_{i \neq j} \frac{d_i}{d_j}]$$

$$\mathbb{E}[R] = 2 \cdot \binom{r}{2} \cdot \mathbb{E}[d_i] \cdot \mathbb{E}[\frac{1}{d_j}] = 2 \cdot \binom{r}{2} \cdot n \cdot \sum_{i=1,\ldots,n} p^2_i$$

$$\mathbb{E}[C] = \mathbb{E}[\sum_{j \neq j'} Y_{j,j'}] = 2 \cdot \binom{r}{2} \cdot \mathbb{E}[Y_{j,j'}] = 2 \cdot \binom{r}{2} \cdot \sum_{i=1,\ldots,n} p^2_i$$

Hence, $\mathbb{E}[R]/\mathbb{E}[C] = n$. □

Let $P_i = \sum_{i} p^j_i$

Let $a = \frac{1}{\sqrt{\sum p^2_i}} = \frac{D}{\sqrt{\sum d^2_i}} \leq \sqrt{n}$, $b = \sum_{i} \frac{1}{p_i} / n^2 \leq D/n$.

Lemma 9 \(\frac{\text{Var}(R)}{\mathbb{E}[R]^2} \leq \frac{1}{r.(r-1)} + \frac{a\cdot b}{r.(r-1)} + \frac{a}{r} + \frac{b}{r} + \frac{2}{r}\).

Proof: Let us estimate $\mathbb{E}[R^2]$:

$$\mathbb{E}[R^2] = \sum_{i \neq j} \mathbb{E}[\frac{d_i}{d_j}]$$

$$\mathbb{E}[R^2] = \sum_{i \neq j} \sum_{i' \neq j'} \mathbb{E}[\frac{d_i}{d_j} \cdot \frac{d_{i'}}{d_{j'}}]$$

We divide the analysis in 6 cases:

1. Case $i = i', j = j'$. There are $2! \cdot \binom{r}{2}$ cases:

$$\mathbb{E}[\frac{d_i}{d_j} \cdot \frac{d_i}{d_j}] = \mathbb{E}[\frac{d^2_i}{d^2_j}] = P_3 \cdot P_{-1}$$

2. Case $i = j', j = i'$. There are $2! \cdot \binom{r}{2}$ cases:

$$\mathbb{E}[\frac{d_i}{d_j} \cdot \frac{d_j}{d_i}] = 1$$

3. Case $i = i', j \neq j'$. There are $3! \cdot \binom{r}{3}$ cases:

$$\mathbb{E}[\frac{d_i}{d_j} \cdot \frac{d_i}{d_{j'}}] = \mathbb{E}[\frac{d^2_i}{d_j \cdot d_{j'}}] = n^2 \cdot P_3$$

4. Case $i = j, j' \neq j'$. There are $3! \cdot \binom{r}{3}$ cases:

$$\mathbb{E}[\frac{d_i}{d_j} \cdot \frac{d_{j'}}{d_{j'}}] = \mathbb{E}[\frac{d^2_i}{d_{j'} \cdot d_{j'}}] = n^2 \cdot P_3$$

5. Case $i = j', j \neq j'$. There are $3! \cdot \binom{r}{3}$ cases:

$$\mathbb{E}[\frac{d_i}{d_j} \cdot \frac{d_{j'}}{d_i}] = \mathbb{E}[\frac{d^2_i}{d_{j'} \cdot d_i}] = n^2 \cdot P_3$$

6. Case $i = j, j' \neq j'$. There are $3! \cdot \binom{r}{3}$ cases:

$$\mathbb{E}[\frac{d_i}{d_j} \cdot \frac{d_{j'}}{d_{j'}}] = \mathbb{E}[\frac{d^2_i}{d_{j'} \cdot d_{j'}}] = n^2 \cdot P_3$$

Hence, $\mathbb{E}[R^2] = \sum_{i \neq j} \mathbb{E}[\frac{d_i}{d_j}]$. □
4. Case $i \neq i', j = j'$. There are $3!(^3r)$ cases:

$$\mathbb{E}\left[ \frac{d_i}{d_j} \cdot \frac{d_i}{d_j} \right] = \mathbb{E}\left[ \frac{d_i}{d_j} \right] = P_2^2 \cdot P_{-1}$$

5. Case $i = j', j \neq i'$ or $i' = j, j' \neq i$. There are $2.3!(^3r)$ cases:

$$\mathbb{E}\left[ \frac{d_i}{d_j} \cdot \frac{d_i}{d_j} \right] = \mathbb{E}\left[ \frac{d_i}{d_j} \right] = n \cdot P_2$$

6. Case different $i, j, i', j'$. There are $4!(^4r)$ cases:

$$\mathbb{E}\left[ \frac{d_i}{d_j} \cdot \frac{d_i}{d_j} \right] = n^2 \cdot P_2^2$$

We can then decompose the 6 cases of $\frac{Var(R)}{\mathbb{E}[R]^2}$ using

$$n \cdot P_2 \geq 1, \quad (r - 1)/(r - 1) \leq 1, \quad P_3/P_2^2 \leq 1/P_2^{1/2}, \quad :$$

$$\frac{Var(R)}{\mathbb{E}[R]^2} \leq \frac{Var(R)}{r(r - 1)n^2P_2^2} \leq \frac{P_3 \cdot P_{-1}}{r(r - 1)n^2P_2^2} + \frac{1}{r(r - 1)n^2P_2^2} + \frac{(r - 2) \cdot P_3}{r(r - 1)n^2P_2^2} + \frac{(r - 2) \cdot P_{-1}}{r(r - 1)n^2P_2}$$

$$\frac{Var(R)}{\mathbb{E}[R]^2} \leq \frac{1}{r(r - 1)} + \frac{ab}{r(r - 1)} + \frac{a}{r} + \frac{b}{r} + \frac{2}{r}$$

The heavy term is $\frac{ab}{r(r-1)}$. If $a > b$ then $\frac{ab}{r(r-1)} < \frac{a^2}{r^2}$.

**Lemma 10** $\frac{Var(C)}{\mathbb{E}[C]^2} \leq \frac{n^2}{r(r-1)} + \frac{2a}{r}$.

**Proof:** Let us estimate $\mathbb{E}[C^2]$:

$$E[C] = \mathbb{E}[\sum_{j \neq j'} Y_{j,j'}] = 2 \cdot \binom{r}{2} \cdot \mathbb{E}[Y_{j,j'}] = 2 \cdot \binom{r}{2} \cdot \sum_{i=1, \ldots, n} p_i^2 = r \cdot (r - 1) \cdot P_2$$

$$E[C^2] = \mathbb{E}[\left( \sum_{i \neq j} Y_{i,j} \right)^2] = \sum_{i \neq j} Y_{i,j} \cdot \sum_{i' \neq j'} Y_{i',j'} \cdot \mathbb{E}[Y_{i,j} \cdot Y_{i',j'}]$$

We divide the analysis in 3 cases:

1. Case $i = i', j = j'$. There are $2!(^2r)$ cases:

$$\mathbb{E}[Y_{i,j} \cdot Y_{i',j'}] = \mathbb{E}[Y_{i,j}] = P_2$$
2. Case \( i \neq i', j = j' \) or \( i' = i, j' \neq j \). There are 2.3!\((\binom{r}{3})\) cases:

\[
\mathbb{E}[Y_{i,j}Y_{i',j'}] = \mathbb{E}[Y_{i,j}Y_{i',j}] + \mathbb{E}[Y_{i,j}Y_{i,j}] = P_3
\]

3. Case different \( i, j, i', j' \). There are 4!\((\binom{r}{4})\) cases:

\[
\mathbb{E}[Y_{i,j}Y_{i',j'}] = (\mathbb{E}[Y_{i,j}])^2 = P_2
\]

This term is at most \( \mathbb{E}[C]^2 \).

We can then decompose the 3 cases of \( \frac{\text{Var}(C)}{\mathbb{E}[C]^2} \) in:

\[
\frac{\text{Var}(C)}{\mathbb{E}[C]^2} = \frac{\text{Var}(C)}{r(r-1)P_2^2} \leq \frac{1}{r(r-1)P_2} + \frac{2(r-2).P_3}{r(r-1)P_2^2}
\]

\[
\text{Var}(C) \leq \frac{a^2}{r(r-1)} + \frac{2a}{r}
\]

The heavy term is \( \frac{a^2}{r(r-1)} \approx \frac{a^2}{r^2} \). Let \( f(\varepsilon, \delta) = \frac{c}{\varepsilon^2}\delta \cdot \frac{D}{\min\{\sum d^2/n\}} \).

**Theorem 1** If \( r \geq f(\varepsilon, \delta) \), then \( \text{Prob}[\hat{n} \in [(1-\varepsilon)n, (1+\varepsilon)n]] \geq 1 - \delta \).

**Proof:** Consider now \( \text{Var}(R) \) to bound the error on \( \hat{n} \), using Chebyshev’s inequality:

\[
\text{Prob}[|R - \mathbb{E}[R]| \geq \varepsilon] \leq \frac{\text{Var}(R)}{\varepsilon^2}. \text{ If } |R - \mathbb{E}[R]| \leq \varepsilon. \mathbb{E}[R]/3 \text{ and } |C - \mathbb{E}[C]| \leq \varepsilon. \mathbb{E}[C]/3 \text{ then: }
\]

\[
(1-\varepsilon)n \leq \frac{(1-\varepsilon).\mathbb{E}[R]/3}{(1+\varepsilon).\mathbb{E}[C]/3} \leq \frac{R}{C} \leq \frac{(1+\varepsilon).\mathbb{E}[R]/3}{(1-\varepsilon).\mathbb{E}[C]/3} \leq (1+\varepsilon)n
\]

\[
\text{Prob}[|R - \mathbb{E}[R]| \geq \mathbb{E}[R]/3] + \text{Prob}[|R - \mathbb{E}[R]| \geq \mathbb{E}[R]/3] \leq \delta
\]

It is achieved if:

\[
\frac{\text{Var}(R)}{\mathbb{E}[R]^2} + \frac{\text{Var}(C)}{\mathbb{E}[C]^2} \leq \frac{\varepsilon^2.\delta}{9}
\]

Using the lemma 10 and 9, it is easy that if \( r - 1 \geq 7c.\text{Max}\{a, b\} \) then

\[
\frac{\text{Var}(R)}{\mathbb{E}[R]^2} + \frac{\text{Var}(C)}{\mathbb{E}[C]^2} \leq \frac{1}{c} = \frac{\varepsilon^2.\delta}{9}
\]

Notice that if we only take the heavy terms \( \frac{2a^2}{r^2} \leq \frac{1}{c} \) if we take \( r > 7c.a \). In the worst case, \( f(\varepsilon, \delta) = O(\sqrt{n}) \) but is much smaller on graphs given by a Zipfian degree distribution as the ones encountered in social networks.
4.2 Graph properties in a stream

A stream of edges defines at any time $t$ a graph $G_t$. Notice that we can sample edges uniformly, with a reservoir sampling and obtain a distribution of nodes where each node has a probability to be chosen proportional to its degree.

Consider the following procedure, given a reservoir $S$ of size $k$. Take a uniform edge of $S$, with probability $1/k$ and then the origin (or the extremity) of the edge with probability $1/2$. The nodes will be taken with probability $d_i/D$. Assume it is true for $m$ edges, and a new edge arrives. It may connect a new node to an old node. To select the new node of degree 1, we need to select the new edge with probability $k/(m+1)$, then select it in the reservoir with probability $1/k$ and finally select the node with probability $1/2$. Hence the node is selected with probability $k/2k(m+1) = 1/D$. To select the old node, it was selected with probability $d_i/D$. If we keep the new edge it is also selected with probability $k/2k(m+1) = 1/D$ so the total probability is $d_i/D$.

4.2.1 Size of the graph

In this case, we can use the previous estimator, as nodes can be selected with the correct distribution. On the other hand we don’t have $d_i$ but only $d_{i,R}$ i.e. the degree of the node $i$ in the reservoir $R$.

We could maintain $\hat{R}$ and $\hat{C}$ from $r$ independent samples taken with a reservoir sampling, estimate the degrees and the collisions and output $\hat{n} = R/C$. If we take $d_{i,R}$ in the expression:

$$E[R] = E[\psi_1.\psi_{-1} - r] = E[\sum_{i=1,...,r} d_i. \sum_{i=1,...,r} \frac{1}{d_i} - r] = E[\sum_{i\neq j} \frac{d_i}{d_j}]$$

then $E[d_{i,R}]$ has to consider all possibilities from 1 to $k-1$ and use $c_1, c_2, \ldots$, the number of nodes of degree 1, 2, ... in the graph $G$. It is is different from $E[d_i]$. In fact we will not obtain $n$ as in the previous case. If we take a graph $G$ as a union of isolated edges, then the Reservoir will be $k$ isolated edges. We will not distinguish a graph of size $2n$ from a graph of size $4n$.

What is more interesting is to interpret the relative sizes of the connected components of the Reservoir. It reflects the relative sizes of the dense subgraphs, i.e. the Communities of the graph.

4.2.2 Communities and dense subgraphs

Community detection in a social graph is a well established subject. Social networks such as Twitter evolve dynamically, and dense subgraphs appear and disappear over time as interest in particular events grows and wanes. How can we detect large dense subgraphs efficiently?
Recall that a subgraph \((S, E(S))\) is \(\rho\)-dense if \(|E[S]|/|S| \geq \rho\).

In the case of a stream of edges, the approximation of dense subgraphs is well studied in [6, 12, 13, 18] and an \(\Omega(n)\) space lower bound is known [5]. If we assume that the dense subgraphs are large, of size \(\Omega(\sqrt{n})\), we show that the space lower bound is reduced to \(\Omega(\sqrt{n})\). Social graphs define a specific regime for which we propose a streaming algorithm which uses \(\Omega(\sqrt{n} \log n)\) space.

**Definition 1** The large dense subgraph problem, parameterized by \(\gamma\) and \(\delta\), takes as input a graph \(G = (V, E)\) and decides whether there exists an induced subgraph \(S \subseteq V\) such that \(|S| > \delta \sqrt{n}\) and \(|E[S]| > \gamma|S|(|S| - 1)/2\).

**Social networks and power laws.** A scale-free network is a network whose degree distribution asymptotically follows a power law: the fraction of nodes with degree \(d\) is proportional to \(d^{-\delta}\) for \(d\) tending to infinity. Many real-world networks are thought to be scale-free. The most widely known generative model appears in work by Barabási and Albert [8]. Another generative model is the copy model studied by Kumar et al. [17], which also generates a power law. Other examples can be found in [3, 22, 2, 21]. Networks generated by preferential attachment tend to place the high-degree nodes in the “middle” of the network, connecting them together to form a core. The configuration models, are defined as uniform distributions \(\mu\) over a space of graphs with a fixed degree sequence, is a standard null model used already in sociology (in a directed version), ecology, biology, statistics, etc. in 1938 by Moreno and Jennings in a directed version [19], and also for modeling the World Wide Web [11], social contagion [1], public opinion formation [24], etc. We focus on social network models with a fixed degree sequence following a power law with parameter \(\delta = 2\), such that the total number of edges is \(m = O(n \log n)\).

**Dense subgraph detection algorithm** Let \(C\) be the largest connected component of the Reservoir of size \(k = \Theta(\alpha\sqrt{n} \log n)\). In order to decide the graph property \(P:\) is there is a \(\gamma\)-clique of size greater then \(\delta \sqrt{n}\)? Consider this simple algorithm.

**Algorithm** \(\text{Detect}(\gamma, \delta)\). Input: a stream of edges of a graph \(G\). Let \(\alpha = 1/\gamma \delta\).

- Use Reservoir sampling to maintain a reservoir \(R\) of size \(k = \Theta(\alpha\sqrt{n} \log n)\). Let \(C\) denote the vertices of the largest connected component of \(R\).
- Accept iff \(|C| \geq \Theta(n^{1/8} \log^2 n)\).

The sampling rate is \(\frac{k}{m}\), i.e. \(\frac{k}{m} = \frac{\alpha\sqrt{n} \log n}{\alpha \sqrt{n}} = \frac{\alpha}{\sqrt{n}}\). We show the correctness of this algorithm in two steps. We can show that Algorithm \(\text{Detect}(\gamma, \delta)\) rejects almost surely in the uniform case. We show in Theorem 2 that Algorithm \(\text{Detect}(\gamma, \delta)\) accepts almost surely if there is a \(\gamma\)-clique.
Analysis of Algorithm Detect(\(\gamma,\delta\)) : Detection of a large dense subgraph \(S\).

We analyze the size of the largest connected component \(C\) of the Reservoir used for the detection of a dense subgraph (Algorithm Detect(\(\gamma,\delta\))). The following theorem formalizes the fact that Algorithm Detect(\(\gamma,\delta\)) is correct with high probability on any graph that contains a large \(\gamma\)-clique.

**Theorem 2** Assume that \(G\) contains a \(\gamma\)-clique on \(S\) where \(|S| > \delta,\sqrt{n}\). If \(\alpha > \frac{(1+\varepsilon)\delta}{\gamma,|S|}\), then Algorithm Detect(\(\gamma,\delta\)) Accepts almost surely.

**Proof:** \(S\) is a \(\gamma\)-clique,

\[
\frac{k}{m} = \alpha/\sqrt{n} \gg \frac{(1+\varepsilon)}{\gamma,|S|}
\]

In a \(\gamma\)-clique, where each edge is taken with probability \(p\), there is a giant component if

\[
p > \frac{(1+\varepsilon)}{\gamma,|S|}
\]

Hence \(C\) is large and Algorithm Detect(\(\gamma,\delta\)) accepts almost surely. \(\square\)

The above theorem shows that on positive instances, Algorithm Detect(\(\gamma,\delta\)) is almost surely correct. What about negative instances? We observe that there exists an input graph \(G\) that does not have a \(\gamma\)-clique of size strictly greater than \(\varepsilon,\sqrt{n}\), yet which Algorithm Detect(\(\gamma,\delta\)) (incorrectly) accepts. \(G\) consists of a clique \(K\) of size \(\varepsilon,\sqrt{n}\) and of a path of size \(n - |K|\). With high probability, the Reservoir contains a component of size at least 90% of \(K\), and will therefore accept, incorrectly.

However this input is somewhat pathological. We can prove that, assuming that \(G\) is drawn from the following stochastic power law model, the algorithm is correct on \(G\) with high probability.

**Space lower bounds.** The restricted multiparty disjointness problem in the 1-way communication model is defined as follows. There are \(q\) players and for each \(j = 1,\ldots, q\), player \(j\) has an \(n\)-bit vector \(x_j = x_{j,1} \ldots x_{j,n}\). In the restricted problem:

- either all vectors \(x_j\) are pairwise distinct (i.e. there are no \(j, j', i\) such that \(x_{j,i} = x_{j',i} = 1\)),

- or there exists a unique \(i^*\) such that \(\bigwedge_j x_{j,i^*} = 1\), and all vectors \(x_j\) are otherwise pairwise distinct (i.e. there is no \(j, j', i \neq i^*\) such that \(x_{j,i} = x_{j',i} = 1\)).

In the 1-way communication model, information is only sent from a player \(j\) to a player \(j'\) such that \(j' > j\), and the last player, player \(q\), must decide whether there is an \(i\) such that \(\bigwedge_j x_{j,i} = 1\).
Lemma 11 [?]: The restricted multiparty disjointness problem with 1-way communication and \( q \) players requires communication complexity \( \Omega(n/q) \).

Lemma 12 [5]: A \( \sqrt{q}/2 \)-approximation streaming algorithm for the maximum density ratio requires space \( \Omega(n/q) \).

Proof: The reduction is from the restricted multiparty disjointness problem with 1-way communication, shown in figure 1. Take an instance with \( q \) players. Let \( q = \sqrt{n} \) and consider an instance of the restricted \( q \)-party disjointness problem with 1-way communication. Player \( j \) holds boolean variables \( x_{j,1}, x_{j,2}, \ldots, x_{j,n} \), for each \( j = 1, 2, \ldots, q \). We construct a graph \( G \) as a union of \( n \) disjoint graphs \( G_1, \ldots, G_n \), each over \( q \) vertices. For \( i = 1, 2, \ldots, n \), the nodes of \( G_i \) are denoted \( u_{1,i}, u_{2,i}, \ldots, u_{q,i} \). To define the edges of \( G \), if \( x_{j,i} = 1 \) then we add the \((q-1)\) edges from node \( u_{j,i} \) to all the other nodes of \( G_i \), \( u_{j',i} \) for \( j' \neq j \).

For a Yes instance, such that \( \bigwedge_j x_{j,i} = 1 \), graph \( G_i \) is a clique and the maximum density ratio is therefore \( \rho^* = (q-1)/2 \). For a No instance, \( G \) is a forest and the maximum density ratio is \( \rho^* = (q-1)/q = 1 - 1/q \).

The input stream has the edges of player 1, then of player 2, etc., with the edges of player \( q \) coming last. A streaming algorithm for \( \rho^* \) with approximation less than \( \sqrt{q}/2 \) and space \( o(n/q) \) could decide between a Yes and a No instance of the \( q \)-multiparty disjointness problem.

The input stream has the edges of player 1, then of player 2, etc., with the edges of player \( q \) coming last. A streaming algorithm for \( \rho^* \) with approximation less than \( \sqrt{q}/2 \) and space \( o(n/q) \) could decide between a Yes and a No instance of the \( q \)-multiparty disjointness problem.

\[ \begin{bmatrix} x_{1,1} = 0 \\ x_{1,2} = 1 \\ \vdots \\ x_{n,n} \end{bmatrix} \begin{bmatrix} x_{j,1} \\ x_{j,2} \\ \vdots \\ x_{j,n} \end{bmatrix} \begin{bmatrix} x_{q,1} \\ x_{q,2} \\ \vdots \\ x_{q,n} \end{bmatrix} \Rightarrow \begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_q \end{array} \]

Yes instance of q-DISJ \( \Rightarrow \) some clique \( G_i \)
No instance \( \Rightarrow \rho < 1 - 1/q \)

Figure 1: The reduction from multiparty disjointness to dense graphs

If we assume that \( S \) must be large, of size at least \( 2\sqrt{n} \), we obtain:

Corollary 1 A \( n^{1/4} \)-approximation streaming algorithm for the maximum density ratio for \( S \) of size at least \( 2\sqrt{n} \) requires space \( \Omega(\sqrt{n}) \).
Proof: Let \( q = 2\sqrt{n} \) in the proof of the previous Lemma. \( \Box \)

4.2.3 Uniform Spanning trees

The problem of generating uniform Spanning Trees has been studied by Aldous and Broder [4] and recent studies on the approximation of the effective resistance of edges [10] have renewed some interest in the problem. Given an edge \( e \in E(G) \) the graph \( G/e \) is obtained by contracting the edge \( e \) while \( G\setminus e \) is obtained by deleting the edge \( e \).

The output of a randomized algorithm \( A \) is denoted by \( A(G) \). Given a spanning tree \( T \) and a cycle \( C \), the cycle obtained by adding the edge \( e \) to \( T \) is denoted by \( C(e,T) \). The length of the cycle is denoted as \( |C(e,T)| \). The number of spanning trees in \( G \) is denoted by \( Z(G) \). The effective resistance for edge \( e \in G \) is denoted as \( R(e) \). Interpret each edge as a resistance and Ohm’s law \( V = RI \). For two nodes \( a, b \), \( I(a,b) \) is the global current between \( a \) and \( b \): then \( V(a) - V(b) = R(a,b).I_1 \). If we set \( I(a,b) = 1 \), then the effective resistance is \( V(a) - V(b) = R(a,b).I_1 \) where \( I_1 \) is the fraction of the current in edge \( (a,b) \).

Consider a triangle \( a, b, c \). Let \( V(b) = 0 \) and the current \( I \) from \( a \) has two components \( I_1 \) along \( (a,b) \) and \( I_2 \) along \( (a,c), (c,b) \). By symmetry all the resistances are identical to \( R \).

We write \( V(a) = R.I_1 \), \( V(a) = 2RI_2 \), \( I_1 + I_2 = 1 \).

Hence \( 2V(a) = 3R \) and \( R = 2/3 \).

If \( L \) is the Laplacian of the Matrix, let \( i \) be the vector of dimension \( n \) such that \( i[a] = 1 \) and \( i[b] = -1 \) and \( i[c] = 0 \) otherwise. Then

\[
i = L.v \\
v = L^+.i \\
v[a] - v[b] = i^+.v = i^+.L^+.i
\]

It is well known that if \( T \) is a uniform spanning tree for \( G \) then \( Pr[e \in T] = R(e) \) [10]. A possible algorithm is to choose each edge with probability \( R(e) \) and then reduce the graph and recompute the resistances. In [10], the authors show how to approximate these computations. The sum of effective resistances \( \sum_e R(e) = n - 1 \). We use this well known identity:

\[
Z(G). (1 - R_e) = Z(G\setminus e)
\]

**Streaming Algorithm to construct a spanning tree** \( T \). Keep a spanning tree \( T \) at any time. When a new edge \( e \) appears in the stream, detect if there is a cycle \( C \) in \( T + e \). If there is no cycle, add \( e \) to \( T \), otherwise break the cycle by removing a uniform edge of \( C \), i.e. an edge of \( C \) with probability \( 1/|C| \).
Let $\pi$ a permutation of the edges. It is easy to see that the distribution of the spanning trees $T$ may not be uniform for a given $\pi$. We conjecture that it is close to the uniform distribution if we choose a uniform $\pi$, i.e.

$$Prob_{\pi,r}[T] \simeq 1/Z(G)$$

when $r$ are the random choices of the Streaming Algorithm.

5 Regular properties of a stream of Balanced words [14]

Balanced words or Dyck languages or Parentheses languages assume an alphabet of complementary letters: $\Sigma = \{a, b, \ldots, \overline{a}, \overline{b}, \ldots\}$. A word is well formed or balanced if complementary letters match the nested structure of the word. For example $ab\overline{b}a$ is balanced but $ab\overline{a}b$ is not balanced. Alternatively, we can consider a labeled unranked ordered tree and there is a one to one correspondence between balanced words and trees.

Balanced words encode semi-structured databases (XML or Json), runs of call/return in a program or in a system. Regular trees generalize regular expressions on words and are represented as schemas. As an example, let (1) be the Schema:

$$c : c + b^*$$
$$b : b + a$$
$$a : a + \overline{a}$$

It defines a family of labeled trees of Figure 2. A VPA (Visual Pushdown Automata) accept precisely the regular balanced words. It is a special class of Tree Automata.

The central question is to decide given a balanced word or a tree $T$ and a schema $S$ whether $T$ is valid or $\varepsilon$-far from valid. Can we use finitely many samples?

The (standard) edit distance $\text{dist}(u, v)$ between two weighted words $u$ and $v$ is defined as the minimum total cost of a sequence of edit operations changing $u$ to $v$. All letters that have not been inserted nor deleted must keep the same weight. For a restricted set of letters $\Sigma'$, define $\text{dist}_{\Sigma'}(u, v)$ when insertions (but not deletions) are restricted to letters in $\Sigma'$ (this makes $\text{dist}_{\Sigma'}$ not symmetric). A weight function on a word $u$ with $n$ letters is a function $\lambda : [n] \rightarrow \mathbb{N}^*$ on the letters of $u$, whose value $\lambda(i)$ is called the weight of $u(i)$. A weighted word over $\Sigma$ is a pair $(u, \lambda)$ where $u \in \Sigma^*$ and $\lambda$ is a weight function on $u$. We define $|u(i)| = \lambda(i)$ and $|u[i, j]| = \lambda(i) + \lambda(i + 1) + \ldots + \lambda(j)$. The length of $(u, \lambda)$ is the length of $u$. For simplicity, we will denote by $u$ the weighted word $(u, \lambda)$. 

17
Testing a regular tree

![Diagram of a valid tree for the schema (1)]

Figure 2: A valid tree for the schema (1)

Similarly, balanced-edit operations can be deletions or insertions of letters, but each deletion of a push symbol (resp. pop symbol) requires the deletion of the matching pop symbol (resp. push symbol). Similarly for insertions: if a push (resp. pop) symbol is inserted, then a matching pop (resp. push) symbol must also be inserted simultaneously. The cost of these operations is the weight of the affected letters, as with the edit operations. We define the balanced-edit distance \( \text{bdist}(u,v) \) between two balanced words as the total cost of a sequence of balanced-edit operations changing \( u \) to \( v \). Similarly to \( \text{dist}_{\Sigma'}(u,v) \) we define \( \text{bdist}_{\Sigma'}(u,v) \).

When dealing with a visibly pushdown language, we will always use the balanced-edit distance, whereas we will use the standard-edit distance for regular languages. Note that since balanced-edit distance is larger than the standard edit distance, our testers will also be valid for that distance.

A sample is a subtree of size \( k \) from a random node, or equivalently a factor and its complement in a balanced word.

5.1 Testing is hard, i.e. it requires \( O(n^{\alpha}) \) queries

**Lemma 13** Testing \( T \) requires \( O(n^{1/4}) \) queries.

**Proof:** In the previous example, using the Birthday paradox, we need \( O(n^{1/4}) \) queries to expect a collision, i.e. 2 samples on the same branch, in one of the \( O(n^{1/2}) \) branches. Testing \( b^*a^* \) requires 2 samples. \( \square \)
5.2 Streaming Property Testing in $O(poly(\log n))$

**Definition 2** Let $\varepsilon > 0$ and let $L$ be a language. A streaming $\varepsilon$-tester for $L$ with one-sided error $\eta$ and memory $s(n)$ is a randomized algorithm $A$ such that, for any input $u$ of length $n$ given as a data stream:

1. If $u \in L$, then $A$ accepts with probability 1;
2. If $u$ is $\varepsilon$-far from $L$, then $A$ rejects with probability at least $1 - \eta$;

We will need the following generalizations:

1. Extend the alphabet to neutral symbols $R_i$
2. Consider letters with a weight
3. Generalize the Tester on regular words to a Tester on weighted regular words

Consider the case when we have a simple peak. For example the first peak is a balanced word

$$w: \quad b\ b...b\ b.\ a\ a\ a....a\ a\ a\ \bar{a}\ \bar{a}...\bar{a}\ \bar{a}\ \bar{b}\ \bar{b}...\bar{b}\ \bar{b}$$

The slice word is the word $w_s: \quad (b, \bar{b}).(b, \bar{b})...(b, \bar{b}).(b, \bar{b}).(a, \bar{a}).(a, \bar{a}).(a, \bar{a})...(a, \bar{a}).(a, \bar{a}).(a, \bar{a})$.

A classical observation is:

**Lemma 14** For any Schema $S$, there is a finite automaton $A$ such that $w$ is valid for $S$ if and only if $w_s$ is accepted by $A$.

We then need to introduce new neutral symbols $R$ which will replace peaks, after they are tested, as in Figure 3 which represents the stack of a VPA automaton.

![Figure 3: The Stack of a VPA and the first compression step](image-url)
The key definition is that of a neutral new symbol $R$ which approximates a balanced word $w$, given the slice-automaton $A$ with states $Q$. It defines the Compression operation.

**Definition 3** A new symbol $R$ such that $R \subseteq Q.Q \varepsilon$-approximates $w$ if:

1. If $q \xrightarrow{w} q'$ then $(q, q') \in R$
2. If $(q, q') \in R$ then $\exists v$ such that $\text{dist}(w, v) \leq \varepsilon$ and $q \xrightarrow{v} q'$

Each such $R$ will replace a peak of size $m$, but $R$ has a weight of $m$. We then need to generalise a tester for regular words to consider weighted words. In this case, we sample a position according to the distribution of weights. This situation is similar for Timed words and Timed Automata.

![Iterated Compression: final slicing word](image)

**Figure 4** Final compression

We also need to define a composition of $R$, as we iterate the compression. In the case of Figure 4 the size of $R$ increases at each step.

5.2.1 Final result

**Theorem 3** For any regular Schema $S$, there is a streaming $\varepsilon$-tester for the language of valid balanced words $L_S$ with one-sided error $\eta$ and memory space $O(c.(\log n)^6(\log 1/\eta)/\varepsilon^4)$.
6 Online algorithms

Online algorithms read their inputs as a stream $s$, process step by step and propose an output at every stage. Consider an optimization problem on an input $x$, solved offline by an agent with unlimited resources, and let $\text{cost}(\text{OPT}(x))$ be the cost of the optimum $\text{OPT}(x)$. Consider now a stream $s$ which describes $x$ and an algorithm $A$ which computes a solution $A(s)$ and approximates the optimization problem with $\text{cost}(A(s))$.

Suppose we have a minimization problem, the competitive ratio is:

$$\max_s \frac{\text{cost}(A(s))}{\text{cost}(\text{OPT}(s))}$$

It is the worst case sequence $s$ for which the ratio $\frac{\text{cost}(A(s))}{\text{cost}(\text{OPT}(s))}$ is maximum. For a maximization problem, the competitive ratio is:

$$\min_s \frac{\text{cost}(A(s))}{\text{cost}(\text{OPT}(s))}$$

It is the worst case sequence $s$ for which the same ratio is minimum.

Notice that the stream $s$ describes an input $x$, but there may be several possible $s$. For example, $G$ is a graph and there are at least two models for $s$:

- the stream $s$ describes the nodes $u$ and all its edges $\{(u, v_i)\}$,
- the stream $s$ describes the edges $(u, v)$.

It may be convenient to imagine an adversary who knows the entire input and the algorithm $A$ and constructs the worst-case $s$. If $A$ is randomized, an oblivious adversary knows $A$ and $x$ but does not know the random choices of $A$.

6.1 Bipartite matching

Let $G_n = (U \cup V, E)$ a bipartite graph where edges link $U$ to $V$, both with $n$ elements. Assume there is a perfect matching $m^*$ between $U$ and $V$, with $n$ edges. How well can an online algorithm $A$ perform? We just compare the size of the matching $|m|$ obtained by $A$ in the worst case, i.e. for a sequence of edges. The problem was first analysed in [15] and in [7].

Let $\sigma$ be an ordering on $V$ of size $n$ and consider a model where nodes of $U$ arrive online with all their edges. For example $(u, v_2), (u, v_7), (u, v_8)$ if the degree of $u$ is 3. Consider the greedy algorithm $A$, which takes inputs $u$ and match the new node with an unmatched $v$ of minimum rank.

It is easy to see that there are cases where $|m| = n/2$. Just consider the input $s : (u_1, v_1), (u_1, v_2), (u_2, v_1)$ for which $A(s) = n/2$. In contrast for $s' : (u_2, v_1), (u_1, v_1), (u_1, v_2)$, $A(s') = n$. Let us prove that the competitive ratio is $1/2$. We then ask if it can be improved.

6.1.1 Competitive ratio of the Greedy’s algorithm

Let $m_g$ be the matching of the greedy algorithm, and $\text{opt}$ the optimal matching.
Lemma 15 For a stream where nodes arrive with all their edges, $|m_g| \geq |opt|/2$.

Proof: If $(u, v) \in m_g$ let $\alpha(u) = 1$ and $\beta(v) = 1$. If $u$ or $v$ is unmatched, then $\alpha(u) = 0$ or $\beta(v) = 0$. Observe that:

$$\sum_{u,v} \alpha(u) + \beta(v) = 2|m_g|$$

We need to show that $|opt| \leq \sum_{u,v} \alpha(u) + \beta(v)$.

To this end, observe that for any $(u, v) \in E$, $\alpha(u) + \beta(v) \geq 1$. Otherwise suppose $\alpha(u) = 0$ and $\beta(v) = 0$, it contradicts the greedy. Then

$$|opt| \leq \sum_{(u,v) \in opt} \alpha(u) + \beta(v) \leq \sum_u \alpha(u) + \sum_v \beta(v)$$

\[\square\]

6.1.2 A better expected Competitive ratio

Assume $G$ has a perfect matching $m^*$, i.e. $|m^*| = n$. Consider the following randomized algorithm.

Ranking $(G, \pi, \sigma)$:

- Choose a random $\sigma$,

- Read the next $u$, i.e. all the edges $\{(u, v_i)\}$. If there an unmatched $v_i$, match $u$ with the $v_i$ of minimum rank for $\sigma$.

The output of Ranking $(G, \pi, \sigma)$ is a matching $m$. We want to show that the expectation of $|m|$ is in the worst case $1 - 1/e \simeq 0.632$. The goal is to show that if a node $u$ is not matched, the Ranking $(G, \pi, \sigma)$ algorithm has matched in expectation most of the nodes before. We follow the following structure:

- we analyse what happens when we remove a vertex $v$, in terms of alternate paths,

- and when we reintroduce the node $v$ in position $i$,

- we prove a main lemma on the probability $x_t$ that a node $v$ of rank $t$ is matched.

Consider $x \in V$ and $H = G - x$ i.e. the graph where we remove the node $x$ and its adjacent edges. Let $\pi_H, \sigma_H$ be the corresponding $\pi$ where some edges are removed and the corresponding $\sigma$ where $x$ is removed. An alternating path is a path with edges of $(H, \pi_H, \sigma_H)$ alternating with edges of $(G, \pi, \sigma)$.

Lemma 16 If the matching of $(G, \pi, \sigma)$ differs from the matching of $(H, \pi_H, \sigma_H)$ then they differ by one alternating path.
Proof: If there is a different matching, some edge \((u, x)\) disappears. If \(u\) is matched in \((H, \pi_H, \sigma_H)\), there is another edge \((u, v)\) where \(v\) has a higher rank than \(x\). We repeat the argument and produce an alternating path. \(\square\)

Lemma 17 Let \(m^*(u) = v\) of rank \(t\). If \(v\) is not matched in Ranking \((G, \pi, \sigma)\), then \(u\) is matched to \(v'\) of rank \(\leq t\).

Proof: If \(v\) is not matched by \(u\), then \(u\) is matched with some \(v'\) of rank \(\leq t\). Otherwise \(v\) would be matched by \(u\). \(\square\)

Lemma 18 Let \(m^*(u) = v\) of rank \(t\). Let \(\sigma_i(v)\) be the order where the node \(v\) is removed and reintroduced at rank \(i\). If \(v\) is not matched in Ranking \((G, \pi, \sigma)\), then for all \(i\), the node \(u\) is matched in Ranking \((G, \pi, \sigma_i(v))\) to \(v_i\) of rank \(\leq t = \sigma(v)\).

Proof: If \(v\) is not matched in Ranking \((G, \pi, \sigma)\), then we can remove \(v\) and obtain the same matching on Ranking \((G, \pi, \sigma - \{v\})\). The ordering \(\sigma - \{v\}\) is the same as \(\sigma_i(v) - \{v\}\).
Therefore the matching obtained by \( (G, \pi, \sigma) \), and by \( (G, \pi, \sigma_i(v) - \{v\}) \) differ by lemma 16 by an alternating path. By lemma 17, \( u \) is matched to some \( v_i \) of rank less than \( t \) the rank of \( v \) in \( \sigma \).

Let \( x_t \) be the probability that a node \( v \) of rank \( t \) is matched by Ranking \((G, \pi, \sigma)\). The fundamental property used in the following lemma.

**Lemma 19**  
For all \( t \):

\[
1 - x_t \leq \sum_{1 \leq s \leq t} x_s / n
\]

Notice that \( 1 - x_t \) is the probability that the node of rank \( t \) is not matched. \( \sum_{1 \leq s \leq t} x_s / n \) is the expected number of matched nodes of rank less than \( t \).

**Proof:** Consider a random \( v \) in \( \sigma \), and place it at rank \( t \). Let \( u \) such that \( m^*(u) = v \). Let \( R_{t-1} \subset U \) be the nodes of \( U \) matched by Ranking \((G, \pi, \sigma - \{v\})\) with a rank \( \leq t - 1 \). Notice that:

\[
\mathbb{E}[|R_{t-1}|] = \sum_{1 \leq s \leq t-1} x_s
\]

By lemma 17, if \( v \) is not matched then \( u \in R_{t-1} \); \( u \) and \( R_{t-1} \) are independent, hence:

\[
\text{Prob}[u \in R_{t-1}] = |R_{t-1}| / n = \sum_{1 \leq s \leq t-1} x_s / n
\]

Hence the lemma. \( \square \)

Let \( S_t = \sum_{1 \leq s \leq t} x_s \). An efficient algorithms tries to maximize \( S_n \) and we want to analyse the situation when \( S_n \) is minimum.

**Theorem 4**  
The expected competitive ratio of Ranking \((G, \pi, \sigma)\) is \( 1 - 1/e \).

**Proof:** From the main lemma 19:

\[
1 - x_t \leq \sum_{1 \leq s \leq t} x_s / n = \frac{S_t}{n}
\]

As \( x_t = S_t - S_{t-1} \):

\[
S_t \cdot (1 + \frac{1}{n}) \geq 1 + S_{t-1}
\]

\[
S_t \geq \frac{n}{n+1} \cdot S_{t-1} + \frac{n}{n+1} \cdot S_{t-1} - 1
\]

\[
S_t \geq \frac{n}{n+1} \cdot S_{t-1} + \frac{n}{n+1} \cdot S_{t-2}
\]

For \( t = n \), we can write:

\[
S_n \geq \frac{n}{n+1} + \left( \frac{n}{n+1} \right)^2 + \ldots \left( \frac{n}{n+1} \right)^{n-1} + \left( \frac{n}{n+1} \right)^{n-1} \cdot S_1
\]
\[ S_n \geq \frac{n}{n+1} \sum_{i=0}^{n-2} \left( \frac{n}{n+1} \right)^i + \left( \frac{n}{n+1} \right)^{n-1}.S_1 \]

We can replace the geometric sum by: \( \frac{1-(n/n+1)^{n-1}}{1-(n/n+1)} \).

\[ S_n \geq \frac{n}{n+1} \cdot \frac{1-(n/n+1)^{n-1}}{1-(n/n+1)} + \left( \frac{n}{n+1} \right)^{n-1}.S_1 \]

\[ S_n \geq \frac{n}{n+1} \cdot \frac{1-(1-1/n+1)^{n-1}}{1-(n/n+1)} + \left( \frac{n}{n+1} \right)^{n-1}.S_1 \]

\[ S_n \geq n.(1-(1-1/n+1)^{n-1}) + \left( \frac{n}{n+1} \right)^{n-1}.S_1 \]

Recall that \( \lim_{n \to \infty} (1 - 1/x)^x = 1/e \), hence:

\[ S_n \geq n.(1 - 1/e) \]

\[ \Box \]
References


