January 2004

Finding Matchings in the Streaming Model

Andrew McGregor
University of Pennsylvania

Finding Matchings in the Streaming Model

Abstract

This report presents algorithms for finding large matchings in the streaming model. In this model, applicable when dealing with massive graphs, edges are streamed-in in some arbitrary order rather than residing in randomly accessible memory. For ε > 0, we achieve a 1/(1+ε) approximation for maximum cardinality matching and a 1/(2+ε) approximation to maximum weighted matching. Both algorithms use a constant number of passes.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-04-19.

Finding Matchings in the Streaming Model

Andrew McGregor[*]

Abstract

This report presents algorithms for finding large matchings in the streaming model. In this model, applicable when dealing with massive graphs, edges are streamed-in in some arbitrary order rather than residing in randomly accessible memory. For $\epsilon > 0$, we achieve a $\frac{1}{1+\epsilon}$ approximation for maximum cardinality matching and a $\frac{1}{2+\epsilon}$ approximation to maximum weighted matching. Both algorithms use a constant number of passes.

1 Introduction

Given a graph $G = (V, E)$, the *Maximum Cardinality Matching* (MCM) problem is to find the largest set of edges such that two adjacent edges are selected. More generally, for an edge-weighted graph, the *Maximum Weighted Matching* (MWM) problem is to find the set of edges whose summed weight is maximized while no two adjacent edges are selected. Both problems are well studied and exact polynomial solutions are known [1, 4, 5, 6]. The fastest of these algorithms solves MWM has running time $O(nm + n^2 \log n)$ where $n = |V|$ and $m = |E|$.

However, for massive graphs in real world applciations, it is sometimes impossible to store the graph in random access memory. In this case the above algorithms are not applicable. To get round this, in this paper we instead work in the graph streaming model discussed in [2, 3, 7]. In this model the edges of the graph stream-in in some arbitrary order. That is, for a graph $G(V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$, a ***graph stream*** is the sequence $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$, where $e_{i_j} \in E$ and $i_1, i_2, \ldots, i_m$ is an arbitrary permutation of $\{1, 2, \ldots, m\}$.

The main computational restriction of the model is that we have restricted space and therefore we can not store the entire graph. In this paper our space restriction is $\tilde{O}(n)$[1] which was identified as a "sweet-spot" for graph streaming in [7] and subsequently shown to be necessary for the verification of even the most primitive of graph properties [3]. We may however have multiple passes of the graph stream but, in this paper, we'll assume that we can only have $O(1)$ passes. To motivate this assumption one can consider external memory systems in which seek times are typically the bottleneck when accessing data.

In this paper we present algorithms that achieve the following approximation ratios:

1. For $\epsilon > 0$, a $\frac{1}{1+\epsilon}$ approximation to maximum cardinality matching.

2. For $\epsilon > 0$, a $\frac{1}{2+\epsilon}$ approximation to maximum weighted matching.

[*]Dept. Computer and Information Science, University of Pennsylvania, Email: andrewm@cis.upenn.edu

[1]Sometimes known as the *semi-streaming* space restriction

MCM and MWM have previously been studied under similar assumptions in [2]. The best previously attained results were a $\frac{1}{6}$ approximation to MWM and for $\epsilon > 0$, a $\frac{2}{3+\epsilon}$ approximation to MCM. Also in the course of this paper we tweak the $\frac{1}{6}$ approximation to MWM to give a $\frac{1}{3+2\sqrt{2}}$ approximation to MWM that uses only one pass of the graph stream.

2 Unweighted Matchings

In this section we describe a streaming algorithm that, for $\epsilon > 0$, computes a $(1-\epsilon)$ approximation to the MCM of the streamed graph. The algorithm will use a constant number of passes. We start by giving some basic definitions common to many matching algorithms.

Definition 1 (Basic Matching Theory Definitions). *Given a matching M in a graph $G = (V, E)$, we call a vertex* free *if it doesn't appear as the end point of any edge in M. A length $2i+1$* augmenting path *is a path $u_1 u_2 \ldots u_{2i+2}$ where u_1 and u_{2i+2} are free vertices and $(u_j, u_{j+1}) \in M$ for even j and $(u_j, u_{j+1}) \in E \setminus M$ for odd j.*

Note that if M is a matching and P is an augmenting path then $M \triangle P$ is a matching of size strictly greater than M. Our algorithm will start by finding a maximal matching and then, by finding short augmenting paths, increase the size of the matching by making local changes. Note that finding a maximal matching is easily achieved in one pass - we select an edge of our graph iff we have not already selected an adjacent edge. The following lemma simply shows us that when there are no longer many short augmenting paths, the size of the matching found can be lower bounded in terms of the size of the maximum cardinality matching OPT.

Lemma 1. *Let M be a maximal matching and* OPT*, a matching of maximum cardinality. Consider the connected components in* OPT$\triangle M$*. Ignoring connected components with the same number of edges from M as from* OPT*, let $\alpha_i M$ = number of connected components with i edges from M. Then*

$$\max_{1 \le i \le k} \alpha_i \le \frac{1}{2k^2(k+1)} \Rightarrow M \ge \frac{\text{OPT}}{1 + 1/k}$$

Proof: In each connected component with i edges from M there is either i or $i+1$ edges from OPT. Therefore, OPT $\le \sum_{1 \le i \le k} \alpha_i \frac{i+1}{i}|M| + \frac{k+2}{k+1}(1 - \sum_{1 \le i \le k} \alpha_i)|M|$. By assumption

$$\sum_{1 \le i \le k} \alpha_i \frac{i+1}{i} + \frac{k+2}{k+1}(1 - \sum_{1 \le i \le k} \alpha_i) \ge \frac{1}{k(k+1)} + \frac{k+2}{k+1} = (1 + 1/k)$$

The result follows. □

So, if there are $\alpha_i M$ components in OPT$\triangle M$ with $i+1$ edges from OPT and i edges from M, then there are at least $\alpha_i M$ length $2i+1$ augmenting paths for M. Finding an augmenting paths allows us to increase the size of M. Hence, if $\max_{1 \le i \le k} \alpha_i$ is small we already have a good approximation to OPT whereas, if $\max_{1 \le i \le k} \alpha_i$ is large then there exists $1 \le i \le k$ such that there are many length $2i+1$ augmenting paths.

Our approach to finding augmenting paths is based on looking for paths in a randomly chosen subgraph of G with a specific structure that we now define.

Definition 2. *Consider a directed graph whose n nodes are partitioned into $i+2$ layers $L_{i+1}, \ldots L_0$ and whose edgeset is a subset of $\cup_{1 \le j \le i+1} \{(u, v) : u \in L_j, v \in L_{j-1}\}$. We call the family of such graphs $\mathcal{L}_i$. We call a path from a node in L_l to a node in L_0, an l-*path.

```
**Algorithm** *Find-Matching*$(G, \epsilon)$
($*$ Finds a matching $*$)
**Output:** A matching
1.  Find a maximal matching $M$
2.  $k \leftarrow \lceil \frac{1}{\epsilon} + 1 \rceil$
3.  $r \leftarrow \frac{32(2k)^k 2 \ln 2}{b_k \alpha^*}$
4.  for $j = 1$ to $r$:
5.      for $i = 1$ to $k$:
6.          do $M_i \leftarrow$ Find-Aug-Paths$(G, M, i)$
7.      $M \leftarrow \text{argmax}_{M_i} |M_i|$
8.  return $M$

**Algorithm** *Find-Aug-Paths*$(G, M, i)$
($*$ Finds length $2i+1$ augmenting paths for a matching $M$ in $G$ $*$)
1.  $G' \leftarrow$Create-Layer-Graph$(G, M, i)$
2.  $\mathcal{P}$ =Find-Layer-Paths$(G', L_{i+1}, \frac{\ln 2}{r b_k}, i+1)$
3.  return $M \triangle \mathcal{P}$

**Algorithm** *Create-Layer-Graph*$(G, M, i)$
**Input:** a graph $G$, a matching $M$ and $i$
1.  if $v$ is a free vertex then $l(v) \in_R \{0, i+1\}$
2.  if $e = (u, v) \in M$ then pick $j \in_R [i]$ and $l(e) \leftarrow j$, $l(u) \leftarrow ja$ and $l(v) \leftarrow jb$ or vice versa.
3.  $E_i \leftarrow \{(u, v) \in E : l(u) = i+1, l(v) = ia\}, E_0 \leftarrow \{(u, v) \in E : l(u) = 1b, l(v) = 0\}$
4.  for $j = 0$ to $i+1$
5.      do $L_j \leftarrow l^{-1}(j)$
6.  for $j = 1$ to $i-1$
7.      do $E_j \leftarrow \{(u, v) \in E : l(u) = (j+1)b, l(v) = ja\}$
8.  return $G' = (L_{i+1} \cup L_i \cup \ldots \cup L_0, E_i \cup E_{i-1} \cup \ldots \cup E_0)$

**Algorithm** *Find-Layer-Paths*$(G', S, \delta, j)$
($*$ Finding many $j$-paths from $S \subset L_j$ $*$)
1.  Find maximal matching $M'$ between $S$ and untagged vertices in $L_{j-1}$
2.  $S' \leftarrow \{v \in L_{j-1} : \exists u, (u, v) \in M'\}$
3.  if $j = 1$
4.    then if $u \in \Gamma_{M'}(L_{j-1})$ then $t(u) \leftarrow \Gamma_{M'}(u)$, $t(\Gamma_{M'}(u)) \leftarrow \Gamma_{M'}(u)$
5.        if $u \in S \setminus \Gamma_{M'}(L_{j-1})$ then $t(u) \leftarrow$ "Dead"
6.        return
7.  repeat
8.      Find-Layer-Paths$(G', S', \delta^2, j-1)$
9.      for $v \in S'$ such that $t(v) \neq$"Dead"
10.         do $t(\Gamma_{M'}(v)) \leftarrow v$
11.     Find maximal matching $M'$ between untagged vertices in $S$ and untagged vertices in $L_{j-1}$.
12.     $S' \leftarrow \{v \in L_{j-1} : \exists u, (u, v) \in M'\}$
13. until $|S'| \leq \delta |L_{j-1}|$
14. for $v \in S$ untagged
15.     do $t(b) \leftarrow$ "Dead".
16. return
```

Figure 1: An Algorithm for Finding Large Unweighted Matchings

At the core of the algorithm for finding length $2i+1$ augmenting paths is an algorithm for finding a nearly maximal set of $i+1$-paths in a graph $G' \in \mathcal{L}_i$. See algorithm *Find-Layer-Paths* in Figure 1. The algorithm finds node disjoint $i+1$-paths by doing something akin to a depth first search. Finding a maximal set of node disjoint $i+1$-paths can easily be achieved in the RAM model by actually doing a DFS, deleting nodes of found $i+1$-paths and deleting edges when backtracking. For the streaming model our algorithm works by finding maximal matchings, first between the first and second levels and then between the nodes in the second that were matched in the first matching, and the third level, and then between the nodes in the third level that were matched in the second matching and the fourth level and so on. When the matching between some subset S of a level L_i and L_{i-1} falls below some threshold we declare all vertices in S that haven't been used in $i+1$-paths, to be dead-ends or just "Dead". The hope is that when we declare vertices dead we are only slightly reducing the number of possible node disjoint $i+1$ paths. For each node v the algorithm maintains a tag indicating if it is "Dead" or, if we've found a $i+1$ path involving v, the next node in the path.

Lemma 2 (Running Time and Correctness of ***Find-Layer-Paths*****).** *Given $G' \in \mathcal{L}_i$, Find-Layer-Paths algorithm finds at least $(\beta - \delta)|M|$ $i+1$-paths where $\beta|M|$ is the size of some maximal set of $i+1$-paths. Furthermore the algorithm takes $O(1)$ passes.*

Proof: *Find-Layer-Paths*$(\cdot,\cdot,\cdot,l)$ is called with argument $\delta^{2^{i+1-l}}$. During the running of *Find-Layer-Paths*$(\cdot,\cdot,\cdot,l)$ when we run line 15, we rule out at most $2\delta^{2^{i+1-l}}|L_{l-1}|$ $i+1$-paths. Let E_l be the number of times *Find-Layer-Paths*$(\cdot,\cdot,\cdot,l)$ is called: $E_{i+1}=1$ and $E_l = E_{l+1}/\delta^{2^{i+1-l}}$ hence $E_l = \delta^{-\sum_{0\le j\le i+1-l-1} 2^j} \ge \delta^{-2^{i+1-l}+1}$. Hence we remove at most $E_l \delta^{2^{i+1-l}}|L_l| = \delta|L_l|$. Note that when nodes are labeled dead in a call to *Find-Layer-Paths*$(\cdot,\cdot,\cdot,l)$, they really are dead and declaring them as such rules out no $i+1$-paths. Hence the total number of paths not found is at most $\delta\sum_{1\le j\le i}|L_j| \le \delta|M|$. The number of invocations of the recursive algorithm is

$$\sum_{1\le l\le i+1} E_l = \sum_{1\le l\le i+1} \delta^{-2^{i+1-}+1} \le \delta^{-2^{k-1}}$$

i.e. $O(1)$ and each invocation requires one pass to find a maximal matching. □

When looking for length $2i+1$ augmenting paths for a matching M in graph G we randomly create a layered graph $G' \in \mathcal{L}_{i+1}$ using *Create-Layer-Graph* such that $i+1$-paths in G' correspond to length $2i+1$ augmenting paths.

Theorem 1. *If G has $\alpha_i M$ length $2i+1$ augmenting paths, then the number of length i-paths found in G' is*

$$b_i(\beta_i - \delta)$$

where $b_i = \frac{1}{2i+2}$ and β_i has a distribution that dominates $\mathbf{Bin}(\alpha_i|M|, \frac{1}{2(2k)^k})$.

Proof: Consider a length $2i+1$ augmenting path $P = u_0u_1\ldots u_{2i+1}$ in G. The probability that P appears as an i-path in G' is at least

$$2\mathbb{P}\left(l(u_0)=0\right)\mathbb{P}\left(l(u_{2i+1})=i+1\right)\prod_{j\in[i]}\mathbb{P}\left(l(u_{2j})=ja \text{ and } l(u_{2j-1})=jb\right) = \frac{1}{2(2i)^i}$$

Given that the probability each augmenting path surviving to G' is independent, the number of length i-paths in G' is distributed as $\mathbf{Bin}(\alpha_i|M|, \frac{1}{2(2k)^k})$. The size of a maximal set of node disjoint i-paths is at least a $\frac{1}{2i+2}$ fraction of the maximum size node-disjoint set i-paths. Combining this with lemma 2 gives the result. □

Theorem 2 (Correctness). *With probability $1 - f$ by running $O(\log \frac{1}{f})$ copies of the algorithm Find-Matching in parallel we find a $1 - \epsilon$ approximation to the matching of maximum cardinality.*

Proof: We show that the probability that a given run of *Find-Matching* fails to find a $(1 + \epsilon)$ approximation is bounded above by e^{-1}. The result then follows.

Assume that for each of the r phases in the algorithm $\max \alpha_i \geq \alpha^* = \frac{1}{2k^2(k-1)}$. (By Lemma 1, if this is ever not the case we are done.) Therefore, in each phase we augment our matching by $\max(|\beta_i - \delta|M||^+ b_i)$ to increase the size of our matching by a fraction

$$(1 + \max(b_i|\beta_i - \delta|^+)/|M|) \geq (1 + b_k|\beta - \delta|^+/|M|)$$

where $\beta \gtrsim X = \mathbf{Bin}(\alpha^*|M|, \frac{1}{2(2k)^k})$. Let $(X_i)_{1\leq i\leq r}$ be independent identically distributed (as X) rv's and $r = \frac{32(2k)^k 2\ln 2}{b_k \alpha^*}$. Let $Y = \prod_{1\leq i\leq r}(1 + b_k|X_i - \delta|^+/|M|)$. Therefore

$$\begin{aligned}
\mathbb{P}(Y \geq 2) &= \mathbb{P}(\ln Y \geq \ln 2) \\
&= \mathbb{P}\left(\sum_{1\leq i\leq r} \ln(1 + b_k|X_i - \delta|^+/|M|) \geq \ln 2\right) \\
&\geq \mathbb{P}\left(\sum_{1\leq i\leq r} X_i \geq |M|(2\ln 2/b_k + r\delta)\right) \\
&= \mathbb{P}(Z \geq 3|M|\ln 2/b_k)
\end{aligned}$$

where $Z = \mathbf{Bin}(\alpha^*|M|r, \frac{1}{2(2k)^k})$. Hence, by an application of the Chernoff bound,

$$\mathbb{P}(Y \geq 2) \geq 1 - \mathbb{P}(Z < 2|M|\ln 2/b_k) > 1 - e^{-(1-1/16)^2 \ln 2|M|/b_k} \geq 1 - e^{-1}$$

$\square$

3 Weighted Matching

We now turn our attention to finding maximum weighted matchings. Here each edge $e \in E$ of our graph G has a weight $w(e)$. For a set of edges S let $w(S) = \sum_{e\in S} w(e)$. We seek to maximize $w(S)$ subject to the constraint that S contains no two adjacent edges.

Consider the algorithms given in Figure 2. The algorithm *Find-Weighted-Matching* can be viewed as a parameterization of the one pass algorithm given in [2] in which γ was implicitly equal to 1. The algorithm *Find-Weighted-Matching-Multipass* generalizes this to a multi-pass algorithm that in effect, runs the one pass algorithm per pass until the improvement yielded falls below some threshold. We start by recapping some notation introduced in [2]. While rather macabre, this notation is nevertheless helpful for developing intuition.

Definition 3. *In a given pass of the graph stream, we say that an edge e is* born *if $e \in M$ at some point during the execution of the algorithm. We say that an edge is* killed *if it was born but subsequently removed from M by a newer heavier edge. This new edge* murdered *the killed edge. We say an edge is a* survivor *if it is born and never killed. For each survivor e, let the* Trail of the Dead $T(e) = C_1 \cup C_2 \cup \ldots$, *where $C_0 = \{e\}$, $C_1 =$ the edges murdered by e , and $C_i = \cup_{e'\in C_{i-1}}$ the edges murdered by e'*

Lemma 3. *For a given pass let the set of survivors be S. The weight of the matching found at the end of that pass is therefore $w(S)$.*

1. *$w(T(S)) \leq w(S)/\gamma$*
2. *$\textsc{Opt} \leq (1+\gamma)\left(w(T(S)) + 2w(S)\right)$*

Proof:

1. For each murdering edge e, $w(e)$ is at least $(1+\gamma)$ the cost of murdered edges, and an edge has at most one murderer. Hence, for all i, $w(C_i) \geq (1+\gamma)w(C_{i+1})$ and therefore $(1+\gamma)w(T(e)) = \sum_{i\geq 1}(1+\gamma)w(C_i) \leq \sum_{i\geq 0} w(C_i) = w(T(e)) + w(e)$. The first point follows.
2. We can charge the costs of edges in $\textsc{Opt}$ to the $S \cup T(S)$ such that each edge $e \in T(S)$ is charged at most $(1+\gamma)w(e)$ and each edge $e \in S$ is charged at most $2(1+\gamma)w(e)$. See [2] for details.

□

Hence in the one pass algorithm we get an $\frac{1}{\frac{1}{\gamma}+3+2\gamma}$ approximation ratio since

$$\textsc{Opt} \leq (1+\gamma)(w(T(S)) + 2w(S)) \leq (3 + \frac{1}{\gamma} + 2\gamma)w(S)$$

The maximum of this function is achieved for $\gamma = \frac{1}{\sqrt{2}}$ giving approximation ratio $\frac{1}{3+2\sqrt{2}}$. This represents only a slight improvement over the $1/6$ ratio attained previously. However, a much more significant improvement is realized in the multi-pass algorithm *Find-Weighted-Matching-Multipass*.

Theorem 3. *The number of passes is $< \frac{(\log 3/2+\sqrt{2})}{\log((2\epsilon)^3/(3(3+2\epsilon)^2))} + 1$ and we achieve a $\frac{1}{2(1+\epsilon)}$ approximation.*

Proof: First we prove the number of passes result. Well since we increase the weight of our solution by a factor $1+\kappa$ each time we do the second step, we start with a $1/(3+2\sqrt{2})$ approximation to optimum so if we do step 2

$$\log_{1+\kappa}(3/2+\sqrt{2}) = \frac{\log(3/2+\sqrt{2})}{\log((2\epsilon)^3/(3(3+2\epsilon)^2))}$$

times we already have an $1/2$ approx.

Let M_i be the matching constructed after the i pass of the data. Let $B_i = M_i \cup M_{i-1}$. Now

$$(1+\gamma)(w(M_{i-1}) - w(B_i)) \leq w(M_i) - w(B_i)$$

and so

$$\frac{w(M_i)}{w(M_{i-1})} = \frac{w(M_i)}{w(M_{i-1}) - w(B_i) + w(B_i)} \geq \frac{(1+\gamma)w(M_i)}{w(M_i) + \gamma w(B_i)}$$

And so if $\frac{w(M_i)}{w(M_{i-1})} < (1+\kappa)$, we get that $w(B_i) \geq \frac{\gamma-\kappa}{\gamma+\gamma\kappa}w(M_i)$.

Using Lemma 3, for all i,

$$OPT \leq (1/\gamma + 3 + 2\gamma)(w(M_i) - w(B_i)) + 2(1+\gamma)w(B_i)$$

```
Algorithm Find-Weighted-Matching($G, \gamma$)
(* Finds Large Weighted Matchings in One Pass *)
1.   $M \leftarrow \emptyset$
2.   for each edge $e \in G$
3.       do if $w(e) > (1+\gamma)w(\{e'|e' \in M, e'$ and $e$ share an end point$\})$
4.           then $M \leftarrow M \cup \{e\} \setminus \{e'|e' \in M, e'$ and $e$ share an end point$\}$ return M

Algorithm Find-Weighted-Matching-Multipass($G, \epsilon$)
(* Finds Large Weighted Matchings *)
1.   $\gamma \leftarrow \frac{2\epsilon}{3}$
2.   $\kappa \leftarrow \gamma \left(\frac{\gamma}{1+\gamma}\right)^2$
3.   Find a $\frac{1}{3+2\sqrt{2}}$ weighted matching, $M$
4.   repeat
5.       $S \leftarrow w(M)$
6.       for each edge $e \in G$
7.           do if $w(e) > (1+\gamma)w(\{e'|e' \in M, e'$ and $e$ share an end point$\})$
8.               then $M \leftarrow M \cup \{e\} \setminus \{e'|e' \in M, e'$ and $e$ share an end point$\}$
9.   until $\frac{w(M)}{S} \leq 1+\kappa$
10.  return M
```

Figure 2: Algorithms for Finding Large Weighted Matchings

since edges in B_i have empty trails of the dead. So if $w(B_i) \geq \frac{\gamma-\kappa}{\gamma+\gamma\kappa}w(M_i)$ we get

$$\begin{aligned} OPT &\leq (1/\gamma + 3 + 2\gamma)(w(M_i) - w(B_i)) + 2(1+\gamma)w(B_i) \\ &\leq (1/\gamma + 3 + 2\gamma - (1/\gamma + 1)\frac{\gamma-\kappa}{\gamma+\gamma\kappa})w(M_i) \\ &\leq (2+3\gamma)w(M_i) \end{aligned}$$

Since $\gamma = \frac{2\epsilon}{3}$ the claimed approximation ratio follows. $\square$

4 Conclusions

New constant pass, $\tilde{O}(n)$ space, constant time-per-edge space streaming algorithms have been presented for the MCM and MWM problems. The MCM algorithms uses a novel randomized technique that allows us to find augmenting paths and finds a matching of size $(1-\epsilon)$OPT. The MWM algorithm builds on previous work to find a matching whose weight is at least $(1/2-\epsilon)$OPT.

References

[1] J. Edmonds. *Maximum matching and a polyhedron with 0,1-vertices.* J. Res. Nat. Bur. Standards, 69B:125-130, 1965.

[2] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri and J. Zhang. *On Graph Problems in a Semi-Streaming Model* To appear in the 31st International Colloquium on Automata, Languages and Programming, 2004

[3] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri and J. Zhang. *Graph Distances in the Streaming Model: The Value of Space* Yale University Technical Report YALEU/DCS/TR-1288, April 2004.

[4] H.N. Gabow, *Data Structures for Weighted Matching and Nearest Common Ancestors with Linking* SODA 1990, 434443

[5] J.E. Hopcroft and R.M. Karp. *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs.* SIAM Journal on Computing, 2(4):225-231, (1973).

[6] S. Micali and V.V. Vazirani, *An O(V E) Algorithm for Finding Maximum Matching in General Graphs*, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science (1980)

[7] S. Muthukrishnan *Data Streams: Algorithms and Applications* Available at http://athos.rutgers.edu/$\sim$muthu/stream-1-1.ps 2003