# Sublinear Time and Space Algorithms 2026A – Lecture 11
# Maximum Matching and Sampling an Edge Uniformly*

Robert Krauthgamer

## 1 Maximum Matching

**Problem definition:**

Input: An $n$-vertex graph $G = (V, E)$ of maximum degree $D$, represented such that one has direct access to the $j$-th neighbor of a vertex.

Definition: A matching is a set of edges that are incident to distinct vertices.

Goal: Compute the maximum size of a matching in $G$.

Note: The matching itself is too large to report in sublinear time, we only estimate its cost using $(\alpha, \beta)$-approximation, i.e., $OPT/\alpha - \beta \leq ALG \leq OPT + \beta$.

**Theorem [Nguyen and Onak, 2008]:** There is an algorithm that gives $(2, \varepsilon n)$ approximation to the maximum matching size in time $D^{O(D)}/\varepsilon^2$.

Main idea: It is well-known that *maximal matching* (note: maximal means with respect to containment) is a 2-approximation for *maximum matching*. We will pick one such matching essentially implicitly, and then estimate its size by sampling.

**Algorithm GreedyMatching:**

1. Start with an empty matching $M$.

2. Scan the edges (in arbitrary order), and add each edge to $M$ unless it is adjacent to an edge already in $M$.

**Lemma:** The output $M$ is a maximal matching, and thus its size is at least half that of a maximum matching.

**Exer:** Prove this lemma.

**Algorithm ApproxGreedyMatching:**

1. choose random edge priorities $p(e) \in [0, 1]$, implicitly defining a permutation $\pi$ of the edges

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.

2. choose $s = O(D/\varepsilon^2)$ edges $e_1, \ldots, e_s$ uniformly at random from the $Dn$ possibilities (note that each edge has two "chances" to be chosen, and some choices may lead to no edge, if the actual degree is smaller than $D$)

3. for each edge $e_i$, compute an indicator $X_i$ for whether $e_i$ belongs to the maximal matching $M$ corresponding to $\pi$, by exploring the neighborhood of $e_i$ incrementally

[stop if the algorithm took too many steps altogether]

4. report $X = \frac{Dn}{2s} \sum_i X_i$

**Running time:** Let $M$ be a greedy matching constructed according to the priorities $p$ (i.e., permutation $\pi$). As seen in class, to determine whether a single $e_i \in M$, we only "expect" to explore paths of length up to $k = O(D)$. Thus, the expected running time is $O(sD \cdot D^{cD}) \leq D^{O(D)}/\varepsilon^2$, and by Markov's inequality there is small probability to exceed it by much.

**Correctness:** As sketched in class, it follows by conditioning on the priorities $p$ (hence the permutation $\pi$ and matching $M$ are fixed), and applying Chebychev's inequality to $X = \frac{Dn}{2s} \sum_i X_i$ (using the randomness of the $s$ samples).

## 2 Sampling an Edge Uniformly

**Problem definition:** Input: A graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges represented as the adjacency list for each vertex (thus can access the degree and outgoing edges of any vertex).

Goal: report an edge sampled uniformly from $E$.

**Easy case:** If $G$ is regular (all vertices have the same degree), one can sample uniformly a vertex and then sample uniformly an outgoing edge.

The problem is harder when the graph is not regular, not even approximately.

**Theorem [Eden, Narayanan and Tetek, 2023]:** There is an algorithm that, given also $m$, samples an edge uniformly in expected time $O(n/\sqrt{m})$. (In fact, an $O(1)$-approximation to $m$ suffices.)

Remark: If $G$ is connected, we saw that $O(1)$-approximation to $m$ can be computed in time $O(\sqrt{n})$, which can be absorbed in $O(n/\sqrt{m})$.

We will see today a weaker result of [Eden and Rosenbaum, 2018], where the output distribution is within $(1+\varepsilon)$-approximation (pointwise):

$$\forall e \in E, \qquad \Pr[\mathit{output} = e] \in (1 \pm \varepsilon)\tfrac{1}{|E|}.$$

This algorithm always outputs an edge, and its expected running time is $O(n/\sqrt{\varepsilon m})$. One can of course terminate it after slightly more time and output FAIL, and argue that it happens with small probability by Markov's inequality.

**Exer (example application):** Show how to estimate within additive $\pm\delta m$ (whp) the fraction of edges inside a subset $S \subset V$ by using $O(1/\delta^2)$ independent samples of a uniformly random edge.

**Proof plan:** Think of every edge as two directed anti-parallel edges (as in the regular case). Our goal is to report every directed edge $uv$ with the same probability, and report FAIL with the remaining (not too large) probability. The final algorithm repeats this until the output is an edge (and not FAIL).

Fix a threshold $\theta = \sqrt{2m/\varepsilon}$, and partition $V$ based on vertex degrees into *light and heavy*, namely $L = \{v \in V : \deg(v) \leq \theta\}$ and $H = V \setminus L$.

A (directed) edge $uv$ is called *light* if its tail $u$ is light. Otherwise the edge is called *heavy*.

Sampling a light edge: Sample uniformly a vertex $u'$ and check it is light, then sample uniformly $j \in [\theta]$ and check if the $j$-th edge from $u'$ exists. If both checks pass, report that $j$-th edge. Otherwise report FAIL. Clearly

$$\forall \text{ light edge } uv, \qquad \Pr[output = uv] = \tfrac{1}{n\theta}. \tag{1}$$

Sampling a heavy edge: First sample a light edge $u'v'$ as described above. Now check that the head $v'$ is heavy and sample uniformly an incident edge $v'w'$. If the check passes report the edge $v'w'$. Otherwise report FAIL. Let $\deg_L(v)$ denote the number of light edge into $v$; then

$$\forall \text{ heavy edge } vw, \qquad \Pr[output = vw] = \tfrac{\deg_L(v)}{n\theta} \cdot \tfrac{1}{\deg(v)}. \tag{2}$$

**Lemma:** If $v \in H$ then $\frac{\deg_L(v)}{\deg(v)} \in [1 - \varepsilon, 1]$.

**Proof:** $\deg_H(v) \leq |H| \leq \frac{2m}{\theta} = \varepsilon\theta < \varepsilon \deg(v)$.

**Caveat:** Which of the two cases/algorithms should we run? Here it's easy, we just pick between the two options with equal probabilities.

**Rejection sampling:** The above uses a technique called rejection sampling, where the algorithm may decide to reject the sample, i.e., report FAIL instead. It could be because the sample turns out to be "bad" (e.g., the vertex is not light), or with some probability (e.g., fixed or that depends on the sample-at-hand) to effectively reduce the probability of reporting this sample.

**Algorithm ApproxSample:**

Input: $G$, $m$, $\varepsilon$

1. $u' \leftarrow$ uniformly random vertex
2. $j \leftarrow U([\theta])$, where $\theta = \sqrt{m/\varepsilon}$
3. if $\deg(u') > \theta$ or $j > \deg(u')$ then return FAIL
4. $v' \leftarrow j$-th neighbor of $u'$
5. $b \leftarrow$ Bernoulli(1/2)
6. if $b = 1$ then
7. return $u'v'$
8. else if $v'$ is heavy then
9. $w' \leftarrow$ random neighbor of $v'$
10. return $v'w'$
11. return FAIL

**Analysis:** Based on (1),(2) and the Bernoulli variable $b$:

$$\begin{aligned} \forall \text{ light edge } uv, \qquad & \Pr[output = uv] = \tfrac{1}{2n\theta}; \\ \forall \text{ heavy edge } vw, \qquad & \Pr[output = vw] = \tfrac{1}{2n\theta} \cdot p(v), \text{ where } p(v) = \tfrac{\deg_L(v)}{\deg(v)} \in [1-\varepsilon, 1]. \end{aligned}$$

The probability to report an edge (and not FAIL) is at least $2m \cdot \frac{1-\varepsilon}{n\theta} = \Omega(\frac{\sqrt{\varepsilon m}}{n})$, hence the expected number of attempts until succeeding to sample an edge is indeed $O(\frac{n}{\sqrt{\varepsilon m}})$.

QED

**First Improvement:** The output has a "bias" of factor $p(v) = \frac{\deg_L(v)}{\deg(v)}$.

The algorithm can estimate $p(v)$ within additive $\delta$, by picking $O(1/\delta^2)$ random neighbors of $v$ and counting how many of them are light. This takes only $O(1/\delta^2)$ time.

If the algorithm knows $p(v)$ exactly, it could just add another round of rejection sampling: Keep a light edge with probability $1-\varepsilon$, and a heavy edge with probability $\frac{1-\varepsilon}{p(v)} \leq 1$. We could use fixed $\varepsilon$ (say $\varepsilon = 1/3$).

If $p(v) \in [\frac{2}{3}, 1]$ then having additive error $\delta$ is equivalent to multiplicative error $1 + O(\delta)$. The output will still be $(1+\delta)$-approximation of the distribution, but the running time improves from multicplicative dependence (on $\varepsilon$) to additive dependence (on $\delta$, recall $\varepsilon = 1/3$).

**Second Improvement:** Apply a method (called Bernoulli factory) that uses in expectation $O(1)$ Bernoulli variables (coins) with unknown parameter $p \in [\frac{2}{3}, 1]$ to generate a Bernoulli variable with parameter $\frac{2/3}{p}$ (more precisely, a quantity proportional to $\frac{1}{p}$) to "fix" the heavy case.

This crux is that the method is independent of $p$.

Remark: The method may use also coins of its own.

**Exer:** Give an example for such a method, for instance using random samples from $B(p)$ to generate $B(1/2)$ (assuming $p \geq 2/3$).