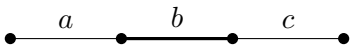
Sublinear Time via Greedy Algorithms

1 Topics covered:

- Matchings, maximality, and the connection to independent sets in a line graph.
- Estimating matching size by sampling vertices and querying a membership oracle.
- Locally simulating greedy matching using a shared random edge ordering.
- Bounding the expected exploration using decreasing-rank paths.
- Briefly Introducing property testing

Throughout Notes: $G = (V, E)$ is a finite simple undirected graph with $n = |V|$, $m = |E|$, and maximum degree at most Δ. We have adjacency-list access and can sample a vertex uniformly.

Definition 1. A *matching* is a set $M \subseteq E$ whose edges have pairwise disjoint endpoints. It is *maximal* if no edge can be added while preserving this property. It is *maximum* if it contains largest amount of edges among all matchings in G.



On this path, $\{b\}$ is maximal: both other edges touch b. However, $\{a, c\}$ is a larger matching. Maximality prevents adding another edge to the set, but it does not prevent improving the solution by replacing edges.

1.1 The line-graph connection

The *line graph* $L(G)$ has one vertex for each edge of G, with two vertices adjacent exactly when their corresponding edges share an endpoint. Thus, a matching in G is an independent set in $L(G)$, and maximality is preserved.

For an edge $e = \{u, v\}$, let $\mathrm{Adj}(e)$ be the other edges sharing an endpoint with e. Then

$$|\,\mathrm{Adj}(e)| = (\deg(u) - 1) + (\deg(v) - 1) \leq 2\Delta - 2.$$

This bounding helps control the branching of the oracle's recursive algorithm. (We will use the simpler bound of 2Δ moving forward.

2 Reducing size estimation to local membership queries

We first imagine we have a deterministic oracle $\mathcal{O}_M(e)$ that can tell us if e belongs to a fixed matching M. We will be using the same matching for all queries to this oracle.

$$x(v) = \begin{cases} 1, & \text{if } v \text{ is incident to an edge of } M, \\ 0, & \text{otherwise.} \end{cases}$$

We can compute $x(v)$ using at most Δ oracle calls, by calling once per edge of v. Another property is that every matching edge has exactly two distinct endpoints, so:

$$\sum_{v \in V} x(v) = 2|M|. \tag{1}$$

2.1 The estimator and its expectation

Fix $0 < \varepsilon < 1$. Sample $v_1, \ldots, v_s$ independently and uniformly from V, *with replacement*, and set $X_i = x(v_i)$. Then we can define:

$$\widehat{m}_0 = \frac{n}{2s} \sum_{i=1}^{s} X_i, \qquad \widehat{m} = \widehat{m}_0 + \frac{\varepsilon n}{2}. \tag{2}$$

Where $\widehat{m}_0$ is an unbiased estimator of $|M|$, as it creates a sample average of the fraction of matched vertices and multiples by n to gets the number. (Divided by 2 again since every matching edge has two endpoints)

By (1), a uniformly sampled vertex is matched with probability $p = 2|M|/n$. Linearity of expectation therefore gives

$$\mathbb{E}[\widehat{m}_0] = \frac{n}{2s} \sum_{i=1}^{s} \mathbb{E}[X_i] = \frac{n}{2s} \cdot s \cdot \frac{2|M|}{n} = |M|.$$

Which leaves $\mathbb{E}[\widehat{m}] = |M| + \varepsilon n/2$ with a deliberate upward shift to overestimate. This is so when it DOESN'T happen, we do not under-count and we over-count by $\leq \epsilon n$.

2.2 Accuracy of the estimate

Recall that

$$\mathbb{E}[\widehat{m}] = |M| + \frac{\varepsilon n}{2}.$$

With $s = \Theta(1/\varepsilon^2)$ independent samples, the additive Chernoff–Hoeffding bound gives

$$\Pr\left[|\widehat{m} - \mathbb{E}[\widehat{m}]| \geq \frac{\varepsilon n}{2}\right] \leq \frac{1}{3},$$

Thus, with probability at least 2/3, the output is within $\varepsilon n/2$ of its expectation. Substituting the expectation gives

$$|M| \leq \widehat{m} \leq |M| + \varepsilon n.$$

This explains why we added $\varepsilon n/2$ to the estimator. It shifts the error interval so on success, the estimate does not underestimate the matching size and overestimates it by at most εn.

The number of samples depends only on ε, not on n. Each sampled vertex requires at most Δ membership queries, so the total is $O(\Delta/\varepsilon^2)$ oracle calls.

3 A local oracle for greedy matching

3.1 Fixing the global solution

Given an ordering of E, the sequential greedy algorithm starts with $M = \varnothing$ and processes every edge $e = \{u, v\}$ in a pre-established order. It adds e exactly when both endpoints are currently unmatched.

The result is a matching because a new edge never touches an already selected edge. It is maximal because any edge that could be included must have been rejected for having an en endpoint already matched.

To randomize the order, assign each edge e an independent rank r_e uniformly in $(0, 1)$ and process edges in increasing rank.

Consistency across queries. We can assume the that ranks are distinct and sampled once and reused. Every occurrence of the same edge has the same rank in separate oracle calls. After fixing r, the oracle is deterministic and refers to one well-defined matching M_r. This is needed because fresh ranks for each query would not provide this consistency.

3.2 What greedy does at an edge

When greedy reaches e, only earlier edges can already be in the matching. Moreover, only edges in $\mathrm{Adj}(e)$ can block e. Because of this, we can construct the following recursive algorithm:

Oracle $\mathsf{InMatching}(e)$

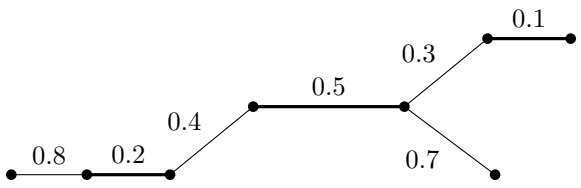
1. For each $e' \in \mathrm{Adj}(e)$:
   (a) If $r_{e'} < r_e$, recursively evaluate $\mathsf{InMatching}(e')$.
   (b) If that recursive call returns TRUE, return FALSE immediately.
2. Return TRUE.

For every fixed assignment of distinct ranks, $\mathsf{InMatching}(e)$ returns True iff $e \in M_r$.

4 Understanding the dependencies

4.1 A worked example

Consider the following graph, with ranks written beside its edges. The middle edge of rank 0.5 is the queried edge.



The oracle's reasoning is:

1. The 0.5 edge must check its lower-ranked neighbors 0.3 and 0.4. Its neighbor of rank 0.7 is later in the order and cannot block it.
2. The 0.3 edge checks the 0.1 edge. The latter has no lower-ranked neighbor, so it is selected. Therefore the 0.3 edge is rejected.
3. The 0.4 edge checks the 0.2 edge. Both neighbors of the 0.2 edge have larger ranks, so it is selected. Therefore the 0.4 edge is rejected.
4. Neither lower-ranked neighbor of the 0.5 edge is selected, so the queried edge is selected.

The bold edges are the resulting matching. Notice that a smaller rank alone does not block the query: the smaller-ranked edge must also actually belong to the matching.

4.2 Why arbitrary orderings can be expensive

Suppose a path has edges $e_1, \ldots, e_\ell$ in path order and ranks

$$r_{e_1} < r_{e_2} < \cdots < r_{e_\ell}.$$

To decide whether e_ℓ is selected, the recursive oracle must first determine whether $e_{\ell-1}$ is selected. This continues back to e_1. The final answer depends on alternating accept/reject decisions along the entire path, requiring linear recursive calls even though the maximum degree is two.

To avoid this we can apply random ranks. This does not make such a chain impossible, but it helps make long decreasing chain very unlikely.

5 Expected cost

5.1 One path and its rank ordering

Fix a candidate path of k recursive steps,

$$p = (e_0, e_1, \ldots, e_k), \qquad e_{i+1} \in \mathrm{Adj}(e_i).$$

For the oracle to explore this path, its ranks must satisfy

$$r_{e_0} > r_{e_1} > \cdots > r_{e_k}. \tag{3}$$

There are k steps but $k+1$ *edge labels*, including the root. Assuming the labels are distinct (no cycles), their ranks are iid random variables. All $(k+1)!$ relative orderings are equally likely and there is only one combination that is completely decreasing. Thus

$$\Pr[r_{e_0} > r_{e_1} > \cdots > r_{e_k}] = \frac{1}{(k+1)!}. \tag{4}$$

Anther case is the oracle may stop early after finding a selected edge in another branch. Therefore the correct statement for the actual execution is:

$$\Pr[p \text{ is explored}] \leq \frac{1}{(k+1)!}.$$

5.2 Counting paths at a given depth

Each call has at most 2Δ possible neighboring edges, so there are at most $(2\Delta)^k$ candidate call paths of length k from the root. (This deliberately overcounts like repeated edges and early stopping). Thus, by linearity of expectation,

$$\mathbb{E}[\text{number of calls at depth } k] \leq \frac{(2\Delta)^k}{(k+1)!}.$$

6 From the query tree to the full complexity bound

6.1 Summing over all depths

Let T be the total number of recursive calls, including the root. Summing the expected number of calls over all depths gives

$$\begin{aligned}
\mathbb{E}[T] &\leq \sum_{k=0}^{\infty} \frac{(2\Delta)^k}{(k+1)!} \\
&= \frac{1}{2\Delta} \sum_{j=1}^{\infty} \frac{(2\Delta)^j}{j!} && \text{(substitute } j = k+1\text{)} \\
&= \frac{e^{2\Delta} - 1}{2\Delta} && \text{(use } e^x = \sum_{j=0}^{\infty} x^j/j!\text{)}.
\end{aligned}$$

6.2 Composing the oracle with the estimator

The estimator samples $s = O(\varepsilon^{-2})$ vertices and may issue up to Δ top-level membership queries per sample. Hence its expected total graph-query cost is

$$O\left(\frac{\Delta e^{2\Delta}}{\varepsilon^2}\right) = \boxed{\frac{2^{O(\Delta)}}{\varepsilon^2}}.$$

For fixed Δ and ε, the expected query count is independent of n. However, the bound is exponential in Δ and this is an expectation not a worst-case. As explained before, although unlikely, rare rank assignments can still cause large explorations. Correctness of an individual membership answer is exact; randomness controls the cost and the error of the size estimate. Also, estimated solution is a maximal matching, but not guaranteed to be a maximum matching.

7 Introduction to property testing

This section was briefly covered at the end of class.

Let $\mathcal{P}$ be a collection of n-vertex graphs with a desired property. Property testing aims to distinguish graphs in $\mathcal{P}$ from graphs that are far from $\mathcal{P}$, using few queries.

Definition 2. In the edge-deletion model, a graph G of maximum degree at most Δ is *ε-close* to $\mathcal{P}$ if removing at most $\varepsilon\Delta n$ edges produces a graph in $\mathcal{P}$. Otherwise, it is *ε-far* from $\mathcal{P}$.

The tester should accept graphs in $\mathcal{P}$ and reject graphs that are ε-far. For graphs that are close but not in $\mathcal{P}$, either answer is allowed.

