CS1200: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Lecture 14: Matching

Harvard SEAS - Fall 2025 *2025-10-21*

1 Announcements

Recommended Reading:

- Hesterberg–Vadhan 15
- CLRS 25.1

2 The Computational Problem

A motivating problem. In the US, there is a nationwide system for arranging kidney transplants, matching patients (who need a kidney) with donors (who are willing to donate a kidney). Each donor can only donate one kidney (they need their other one to survive!) and only to certain patients (due to blood type and HLA type compatibilities). There over 100,000 patients currently on the kidney waiting list in the US, with a little over 25,000 donations happening per year, and patients spending an average of about 3.6 years on the waiting list. Algorithms like what we will cover in this chapter play a significant role in saving patients through kidney donations.

Q: How can we model kidney exchange graph-theoretically?

Definition 2.1. For a graph $G = (V, E)$, a *matching* in G is a subset $M \subseteq E$ such that ____ .

If a vertex v is incident to an edge in M, we say v is *matched* by M; otherwise we say it is *unmatched*.

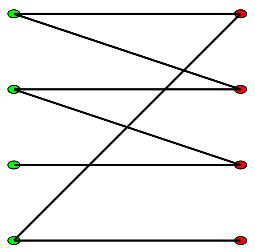
We will focus on problem of finding the largest matching in a graph, called Maximum Matching.

Input: A graph $G = (V, E)$

Output: A matching $M \subseteq E$ in G of maximum size

Computational Problem Maximum Matching

Let's find a maximum matching in the graph below.



Q: What does our modeling omit? Specifically, what additional structure do kidney exchange compatibility graphs have that's not captured in Maximum Matching?

Fact: Maximum Matching can be efficiently reduced to Independent Set. (See textbook.) Why is this not so useful?

3 Alternating and Augmenting Paths

We might try to find a maximum matching by a *greedy strategy,* but this may not find the maximum-size matching. Let's see how it works on the example above (with a particular ordering of edges):

The greedy strategy is not unsalvageable, but we must add an operation more sophisticated than adding a single nonconflicting edge. To do so, we first introduce two graph theory notions related to matchings: *alternating walks* and *augmenting paths*.

Definition 3.1. Let $G = (V, E)$ be a graph, and M be a matching in G. Then:

1. An *alternating walk* W in G with respect to M is

2. An *augmenting path* P in G with respect to M is

Let's see an augmenting path with respect to the maximal (but not maximum) matching we found via the greedy algorithm:

Notice that if for every edge e in the augmenting path, we "switch" whether or not we include e in the matching, we get a strictly larger matching:

This suggests a natural algorithm for maximum matching: repeatedly try to find an augmenting path and use it to grow our matching. But we need to argue that augmenting paths always exist and we can find them efficiently. To turn this idea into an algorithm for finding maximum matchings, we need two things:

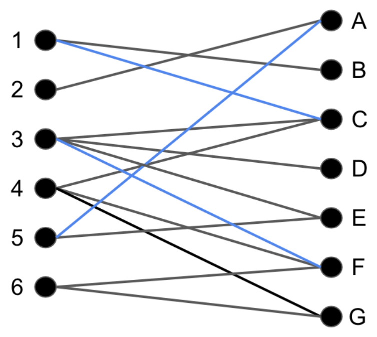
1.

2.

The former is the focus of Section 4 below. We can get the latter guarantee from the following theorem, for which a proof is in the textbook.

Theorem 3.2 (Berge's Theorem)**.** *Let $G = (V, E)$ be a graph, and $M \subseteq E$ be a matching. If (and only if) M is not a maximum-size matching, then (and only then) G has an augmenting path with respect to M.*

Example 3.3. For the following graph drawn below, and the matching $M = \{\{1, C\}, \{3, F\}, \{5, A\}\}$, let's see which sequences of vertices are and are not alternating walks and augmenting paths.



|M| = 3

Sequence	Alternating Walk	Augmenting Path
$2, A, 5$		
$G, 4$		
$3, F, 4, C, 3$		
$1, C, 6$		
$6, F, 3, C, 1, B$		

4 Maximum Matching Algorithm

Like in a pure greedy strategy, we will try to grow our matching M one step at a time, building a sequence $M_0 = \emptyset, M_1, M_2, \ldots$, with $|M_k| = k$. However, to get M_k from M_{k-1} we will sometimes make use of an augmenting path (whose existence is guaranteed by Theorem 3.2) rather than just adding an edge. Specifically, we'll use the following three lemmas (two of which apply only to bipartite graphs, which is the case we will focus on):

Lemma 4.1 ("Certain shortest alternating walks are augmenting paths")**.** *Let G be bipartite, with bipartition (V_0, V_1),*[1] *and let M be a matching in G that is not of*

[1]Recall that a *bipartite* graph is a graph that is 2-colorable, and a *bipartition* of a bipartite graph is a partition of the vertex set $V = V_0 \cup V_1$ into the 2 color classes given by a 2-coloring. Thus all of the edges in the graph have one endpoint in V_0 and one endpoint in V_1.

maximum size. Let U be the vertices that are not matched by M, and $U_0 = V_0 \cap U$ and $U_1 = V_1 \cap U$. Then:

1.

2.

Lemma 4.2 ("Shortest alternating walks are easy to find")**.** *Finding shortest alternating walks in bipartite graphs reduces to*

Lemma 4.3 ("Augmenting paths can be used to efficiently grow matchings")**.** *Given a graph $G = (V, E)$, a matching M, and an augmenting path P with respect to M, we can*

Using these lemmas we will prove:

Theorem 4.4. Maximum Matching *can be solved in time $O(mn)$ on bipartite graphs with m edges and n vertices.*

We defer the proof of Lemmas 4.1 to the textbook.

Proof sketch of Lemma 4.2.

□

Proof of Lemma 4.3.

□

Example 4.5. We repeatedly apply Lemma 4.3 to our example matching from Example 3.3. We first consider the augmenting path $(G, 4)$ and grow our matching accordingly.

Next, we add the augmenting path $(6, F, 3, C, 1, B)$.

Note that $\{3, F\}$ and $\{1, C\}$ are not in the final matching, since they were in both the previous matching and the augmenting path (so they are "switched out" of the matching).

Putting it all together, we have the following algorithm:

```
  MaxMatchingAugPaths($G$):
  Input   : A bipartite graph $G = (V, E)$
  Output  : A maximum-size matching $M \subseteq E$
0 Remove isolated vertices from $G$;
1 Let $V_0, V_1$ be the bipartition (i.e. 2-coloring) of $V$;
2 $M = \emptyset$;
3 repeat
4     Let $U$ be the vertices unmatched by $M$, $U_0 = V_0 \cap U$, $U_1 = V_1 \cap U$;
5     Use BFS to find a shortest alternating walk $P$ that starts in $U_0$ and
        ends in $U_1$;
6     if $P \neq \bot$ then augment $M$ using $P$ via Lemma 4.3;
7 until $P = \bot$;
8 return $M$
```

Algorithm 4.1: `MaxMatchingAugPaths()`