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

Lecture 16: Matching Wrap-Up and Logic

Harvard SEAS - Fall 2025 *2025-10-28*

1 Announcements

- Midterm grades released. Median is $42/64 \to 2.875$ on 4.0 scale (low B), lower than last year, despite being curved more generously.
- Conjecture: too much reliance on collaborators on psets (whether human or AI). There's no substitute for doing the hard work of problem-solving yourself for developing your own understanding.
- Deadline for us to approve pass/fail postponed to next Monday 11/3.

Recommended Reading:

- Hesterberg–Vadhan 17
- Lewis–Zax 9–10
- Roughgarden IV, Sec. 21.5, Ch. 24

2 Loose Ends: Modelling the Prioritarian Objectives

- Assign each patient v to have *two* weights $w_0(v)$ and $w_1(v)$, with $w_0(v)$ giving what v's expected welfare (e.g. as measured by QALYs) would be if they do not receive a kidney donation and $w_1(v)$ giving what v's expected welfare would be if they do receive a kidney donation.
- *Maximin* objective: Applied to kidney exchange, we can mathematically model the maximin principle by seeking a matching M that first attempts to maximize:

  $$\min\left(\{w_0(v) : v \text{ unmatched by } M\} \cup \{w_1(v) : v \text{ matched by } M\}\right).$$

  If two matchings M and M' have the same value of the above objective function, i.e. the worst-off patients v in each of the matchings have the same expected welfare, then we turn to comparing the second-worst-off patients in the two matchings, and if that turns out to be a tie, we proceed to the third-worst-off patients and so on.

- *Prioritarian* objective: For some concave, non-decreasing function f (e.g. $f(x) = \sqrt{x}$ or $f(x) = \log x$, maximizes

$$\sum_{v:v \text{ unmatched by } M} f(w_0(v)) + \sum_{v:v \text{ matched by } M} f(w_1(v)).$$

  This is equivalent to solving the Maximum Vertex-Weighted Matching problem with vertex weights $w(v) = f(w_1(v)) - f(w_0(v))$.

3 Loose Ends: Maximum Matching Algorithm

`MaxMatchingAugPaths(`G`)`:
Input : A bipartite graph $G = (V, E)$
Output : A maximum-size matching $M \subseteq E$

Remove isolated vertices from G;
Let V_0, V_1 be the bipartition (i.e. 2-coloring) of V;
$M = \emptyset$;
repeat
  Let U be the vertices unmatched by M, $U_0 = V_0 \cap U$, $U_1 = V_1 \cap U$;
  Use BFS to find a shortest alternating walk P that starts in U_0 and ends in U_1;
  if $P \neq \perp$ **then** augment M using P via Lemma 3.3;
until $P = \perp$;
return M

Algorithm 3.1: `MaxMatchingAugPaths()`

Lemma 3.1 ("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 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. *G has an alternating walk with respect to M that starts in U_0 and ends in U_1.*
2. *Every shortest alternating walk from U_0 to U_1 is an augmenting path.*

Lemma 3.2 ("Shortest alternating walks are easy to find")**.** *Finding shortest alternating walks in bipartite graphs reduces to finding shortest paths in directed graphs in time $O(n + m)$, where $n = |V|$ and $m = |E|$.*

Lemma 3.3 ("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 construct a matching M' with $|M'| = |M| + 1$ in time $O(n)$.*

[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.

Using these lemmas we will prove:

Theorem 3.4. Maximum Matching *can be solved in time* $O(mn)$ *on bipartite graphs with* m *edges and* n *vertices.*

We defer the proof of Lemmas 3.1 to the textbook.

Proof sketch of Lemma 3.2.

$\square$

Proof of Lemma 3.3.

$\square$

4 Propositional Logic

Motivation: Logic is a fundamental building block for computation (e.g. digital circuits) and a very expressive language for encoding computational problems we want to solve.

Definition 4.1 (boolean formulas, informal)**.** A *boolean formula* φ is a formula built up from a finite set of variables, say $x_0, \ldots, x_{n-1}$, using the logical operators $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT) and parentheses.

Every boolean formula φ on n variables defines a boolean function, which we'll also denote by $\varphi : \{0,1\}^n \to \{0,1\}$, where we interpret 0 as false and 1 as true, and give $\wedge, \vee, \neg$ their usual semantics (meaning).

The Lewis–Zax text (textbook for CS 20) contains formal, inductive definitions of boolean formulas and the corresponding boolean functions.

Example 4.2.

$$\varphi_{maj}(x_0, x_1, x_2) = (x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_0)$$

is a boolean formula. It evaluates to 1 if

$$\varphi_{pal}(x_0, x_1, x_2, x_3) = ((x_0 \wedge x_3) \vee (\neg x_0 \wedge \neg x_3)) \wedge ((x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2))$$

is a boolean formula. It evaluates to 1 if

We now turn to two important special cases of boolean formulas.

Definition 4.3 (DNF and CNF formulas)**.**

- A *literal* is a variable (e.g. x_i) or its negation ($\neg x_i$).
- A *term* is an AND of a sequence of literals.
- A *clause* is an OR of a sequence of literals.
- A boolean formula is in *disjunctive normal form (DNF)* if it is the OR of a sequence of terms.
- A boolean formula is in *conjunctive normal form (CNF)* if it is the AND of a sequence of clauses.

Q: For each of the examples φ_{maj} and φ_{pal} above, is it in DNF, CNF, both, or neither?

One reason that DNF and CNF are commonly used is that they can express all boolean functions:

Lemma 4.4. *For every boolean function $f : \{0,1\}^n \to \{0,1\}$, there are boolean formulas φ and ψ in DNF and CNF, respectively, such that $f \equiv \varphi$ and $f \equiv \psi$, where we use $\equiv$ to indicate equivalence as functions, i.e. $f \equiv g$ iff $\forall x : f(x) = g(x)$.*