Lecture 1: The Stable Matching Problem

Date: August 5, 2026
Instructor: Aniket Basu Roy
Scribe: *[Sample Notes — Instructor]*

Overview. Every problem we study this semester will be treated in the same four steps: *formalize* the problem precisely, *design* an algorithm, *prove* the algorithm correct, and *analyze* its efficiency. Today we carry out all four steps in full on our first problem, the Stable Matching Problem, and then preview five further problems that map out the rest of the course.

1 The Stable Matching Problem

Motivation. Consider assigning a set of medical residents to hospitals, or more simply, a set of n men to n women for marriage, where each person ranks all members of the opposite set by preference. We want an assignment that is *self-enforcing*: no two people who are not matched to each other would both prefer to be.

Definition 1 (Instance)**.** An instance consists of a set $M = \{m_1, \ldots, m_n\}$ of men and a set $W = \{w_1, \ldots, w_n\}$ of women, together with, for each person, a strict total order (*preference list*) over the members of the opposite set.

Definition 2 (Perfect matching)**.** A *perfect matching* S is a set of n pairs (m, w), $m \in M$, $w \in W$, such that every person appears in exactly one pair.

Definition 3 (Instability)**.** Given a perfect matching S, a pair (m, w) *not* in S is an *instability* if m prefers w to his partner in S, *and* w prefers m to her partner in S. A matching with no instability is *stable*.

Worked example. With $M = \{m_1, m_2\}$, $W = \{w_1, w_2\}$ and preferences $m_1 : w_1 \succ w_2$, $m_2 : w_1 \succ w_2$, $w_1 : m_2 \succ m_1$, $w_2 : m_1 \succ m_2$: the matching $\{(m_1, w_1), (m_2, w_2)\}$ is *unstable*, since (m_2, w_1) is an instability. The matching $\{(m_1, w_2), (m_2, w_1)\}$ is stable. So an arbitrary perfect matching need not be stable — an algorithm is needed.

2 The Gale–Shapley Algorithm

Algorithm 1: Gale–Shapley

Input: Preference lists for all $m \in M$ and $w \in W$
Output: A stable matching S

```
Initialize all m ∈ M and w ∈ W to be free
while some man m is free and has not proposed to every woman do
    Let w be the highest-ranked woman on m's list to whom m has not yet proposed
    if w is free then
        (m, w) become engaged
    else
        if w prefers m to her current partner m' then
            m' becomes free
            (m, w) become engaged
        else
            w rejects m; m remains free
return the set of engaged pairs S
```

Algorithm 1 has men propose in decreasing order of their own preference; once engaged, a woman only ever trades up. Traced on the example above: $m_1 \to w_1$ (engaged); $m_2 \to w_1$, who prefers m_2, so w_1 switches and m_1 is freed; $m_1 \to w_2$ (engaged). Output: $\{(m_1, w_2), (m_2, w_1)\}$, matching the stable matching found above.

3 Correctness

Claim 1 (Termination)**.** *The algorithm terminates after at most n^2 iterations.*

Proof. Each iteration is a proposal, and each man proposes to each woman at most once, so there are at most n^2 proposals in total. ■

Claim 2 (Perfect matching)**.** *The set S returned is a perfect matching.*

Proof sketch. Once engaged, a woman never becomes free again (she only trades up). So it suffices to show no man is rejected by every woman. If man m were rejected by all n women, each rejection came from a woman currently engaged to someone she preferred — but a woman's partner only improves over time, so at the end all n women are engaged, contradicting that m is unmatched among n men and n women. ■

Theorem 1. *The matching S returned by the algorithm is stable.*

Proof. Suppose not: some $(m, w) \notin S$ is an instability, so m prefers w to his partner w' in S. Since men propose in decreasing preference order, m must have proposed to w before w', and been rejected — either immediately, or later when w traded up. Either way, at the moment of rejection w was engaged to someone she preferred to m, and by the perfect-matching claim, her partner only improves afterward. So w prefers her final partner to m, contradicting that (m, w) is an instability. ■

4 Five Representative Problems: A Roadmap

To preview the shape of the course, five more problems, ranging across the techniques we will develop:

- **Interval Scheduling** — solved by a greedy algorithm (next unit).
- **Weighted Interval Scheduling** — solved by dynamic programming (later this semester).
- **Bipartite Matching** — a special case of network flow (post-midsem).
- **Independent Set** — the canonical NP-complete problem (post-midsem).
- **Competitive Facility Location** — addressed via approximation algorithms (post-midsem).

None of these is solved today; each returns, solved, once we have built the right technique.

Reference. Kleinberg, J., & Tardos, E. (2006). *Algorithm Design*, Chapter 1. Pearson/Addison-Wesley.