- Chapter 1
- A First Problem: Stable Matching
  - The Problem
  - Formulating the Problem
  - Design the Algorithm
  - Analyzing the Algorithm
  - Extensions
  - All Executions Yield the Same Matching
- Five Representative Problems
  - Interval Scheduling
  - Weighted Interval Scheduling
  - Bipartite Matching
  - Independent Set
  - Competitive Facility Location

Chapter 1

This chapter is an introduction to algorithms as a whole. It begins with an example of the Stable Matching Problem, which demonstrates some of the issues of algorithm design in a concrete manner. The chapter then introduces a list of five "representative problems".

A First Problem: Stable Matching

The Problem

The Stable Matching Problem originated in 1962 by David Gale and Lloyd Shapley, who wanted to create a *self-enforcing* matching process. One example that was thought of was that of college students applying for summer internships. Each group (both the companies and the students) have a preference for who they are matched with. One major issue is that the students could change their mind, and leave an employer for another, or an employer could drop a student for a preferable one. These issues can cause a lot of chaos in the situation.

This is where a situation where self-interest itself ensures stability. This could be where an employer prefers all of it's current employees to any new applicants, or a student prefers its current employer to any others. These situations would indicate that neither would desire to change, causing the situation to be come stabilized.

Gale and Shapley asked: Given a set of preferences among employers and applicants, can we assign applicants to employers so that for every employer E, and every applicant A who is not scheduled to work at E, at least one of the following two things is the case?

i. E prefers every one of its accepted applicants to A; or
ii. A prefers her current situation over working for employer E.

If this holds then the outcome is stable, as self-interest prevents any altercations.

Formulating the Problem

This example of companies and students has a lot of complications, so it is useful to simplify the problem to a "bare-bones" version. This version can be viewed as the problem of devising a system by which each of n men and n women can end up getting married.

Consider a set $M = m_1, ldots, m_n$ of n men, and a set $W = w_1, ldots, w_n$ of n women. A _matching_ S is a *set* of ordered pairs, of (m, w), with the property that each member of M and W is in at most one pair in S. A *perfect matching* S' is a matching with the property that each member of M and W are in *exactly* one pair in S'.

Matchings and perfect matchings are objects that will recurs frequently; they arise naturally in modeling a wide range of algorithmic problems.

Now the notion of *preferences* is added. Each m
inM *ranks* all the women; we will say that m *prefers* w *to* w' if m ranks w higher than w'. This will be m's *preference list*. Each woman also ranks all the men in the same fashion.

There can be *instability* when a m and w rank each other higher than their current match. This would indicate that they would go off with each other. We will say that this is *instability with respect to* S.

Our goal is to create a set of marriages that has no instabilities. We will say a matching S is *stable* if (i) it is perfect, and (ii) there is no instability with respect to S. Two important questions must be asked:

- Does there exist a stable matching for every set of preference lists?
- Given a set of preference lists, can we efficiently construct a stable matching if there is one?

Design the Algorithm

Consider some basic ideas that motivate the algorithm.

- Initially, everyone is unmarried. Suppose an unmarried man m chooses the woman w who ranks highest on his preference list and *proposes* to her. They are then paired into an intermediate state (*engagement*), and later if a man m' proposes to her, and she prefers m' over m, then she is then re-paired with m'.

- Suppose we are at a state where some man and women are *free*, and other are *engaged*. The next step would be for a free man m to propose to the highest-ranked women w to whom he has not already proposed. If w is also free, then m and w become engaged. Otherwise, if w is already engaged to some other man m', she determines which of m or m' ranks higher, and becomes engaged with that one, and the other becomes free.
- Finally, the algorithm will terminate when no one is free. And this perfect matching is returned.

Analyzing the Algorithm

Analyzing the algorithm provides some interesting notes.

1. w remains engaged from the point at which she receives her first proposal; and the sequence of partners to which she is engaged only get higher in her preference list.
2. The sequence of women to whom m proposes get lower in his preference list.
3. The algorithm terminates after at most n^2 iterations. Proving the upper-bound of the running time of an algorithm can be done through finding a measure of *progress*, or some way of saying that each step takes the algorithm closer to completion.
4. If m is free at some point in the execution of the algorithm, then there is a woman to whom he has not yet proposed.
5. The set S returned at termination is a perfect matching.
6. The algorithm returns a set of pairs S that is a stable matching.

Extensions

There can be an *unfairness* that arises, because the men are the ones proposing, and if all the men have different women for their first choice, then all men are immediately in agreement, and the women's preferences are not taken into account.

All Executions Yield the Same Matching

We will say a woman w is a valid partner of m if there is some S that contains (m, w). We will say that w is the best partner of m if w is a valid partner of m and there is no other woman whom m ranks higher than w that is a valid partner. Every execution of the algorithm results with the best possible outcome for every man simultaneously.

In contrast the algorithm pairs women with their worst valid partner.

Five Representative Problems

There are some high-level strategies that are used commonly in the fundamentals of design techniques. These are demonstrated in these five representative problems.

Interval Scheduling

Weighted Interval Scheduling

Bipartite Matching

Independent Set

Competitive Facility Location