Efficient Algorithms for Matching Problems

Dr David Manlove

Department of Computing Science, University of Glasgow, Glasgow G12 8QQ
e-mail: davidm@dcs.gla.ac.uk

Funded by a Royal Society of Edinburgh / Scottish Executive Personal Research Fellowship

Abstract

In many practical contexts we seek to allocate applicants to posts using a centralised matching scheme

Typically we have:

- a set of *applicants* $a_1, a_2, \ldots, a_n$
- a set of *posts* $p_1, p_2, \ldots, p_m$
- applicants have preferences over posts
- posts may have preferences over applicants
- each post has a *capacity* (max no. of applicants it can take on)

This gives rise to a *matching problem*

The aim of this research is to explore the existence of *efficient algorithms* (computer programs) for solving matching problems

Application scenarios

- Scottish Pre-Registration House Office Allocation Scheme (SPA)
  - run by NHS Education Scotland
  - applicants are graduating medical students
  - posts are surgical or medical placements in hospitals
- Scottish Executive Teacher Induction Scheme
  - applicants are trainee teachers
  - posts are probationary teaching placements in schools
- Birmingham LEA School Matching Scheme
  - applicants are primary school pupils
  - posts are primary school places

Similar examples in USA, Canada, France, Norway, Spain, Singapore, Turkey…

Example – allocating medical students to hospitals

5 students $s_1, s_2, \ldots, s_5$ and 5 hospitals $h_1, h_2, \ldots, h_5$
Each hospital has capacity 1 and preferences are as follows:

- $s_1 : h_1\ h_2\ h_3$
- $s_2 : h_1\ h_2\ h_3\ h_4$
- $s_3 : h_1\ h_2\ h_3\ h_4\ h_5$
- $s_4 : h_1\ h_3\ h_5\ h_4$
- $s_5 : h_2\ h_5\ h_4$

- student s_1 prefers hospital h_1 to hospital h_2, etc.
- student s_2 does not find hospital h_5 acceptable
- hospitals do not have preferences over students

Why use a centralised matching scheme?

Many situations such as the following have occurred in the past:

1. student s_1 makes an arrangement with hospital h_2
2. student s_2 applies to h_2 but is rejected, since h_2 is full
3. student s_2 goes elsewhere
4. student s_1 receives a better offer from h_1
5. student s_1 leaves h_2 and goes to h_1
6. hospital h_2 has now lost a student!

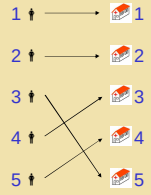
- Systems involving ad-hoc arrangements are widely regarded as unsatisfactory (Roth, 1984, Roth & Sotomayor, 1990)
- Centralised matching schemes automate the allocation process and help to avoid these problems

What is a *matching*?

1. Each student is allocated to at most one hospital
2. No hospital is allocated more students than its capacity
3. No student is allocated to an unacceptable hospital

E.g.:

- $s_1 : (h_1)\ h_2\ h_3$
- $s_2 : h_1\ (h_2)\ h_3\ h_4$
- $s_3 : h_1\ h_2\ h_3\ h_4\ (h_5)$
- $s_4 : h_1\ (h_3)\ h_5\ h_4$
- $s_5 : h_2\ h_5\ (h_4)$



i.e. student s_1 is allocated hospital h_1 etc.

What do we mean by the *best* matching?

Usually there are many different matchings
– how do we choose the best?

Different meanings of "best" – one possibility is:

1. Match as many students to hospitals as possible
2. Subject to 1, match as many students to their 1st-choice hospital
3. Subject to 2, match as many students to their 2nd-choice hospital etc.

This is called a *greedy maximum* matching

How does a centralised matching scheme work?

1. Applicants form preferences over posts
   - possibly involves open days / reading published material
2. Posts form preferences over applicants (if applicable)
   - possibly on the basis of exam results / interviews
3. Preference data and capacity information passed to administrators of centralised matching scheme
4. Computer program constructs allocation of applicants to posts

   … 00100011110011001110011 …
5. Allocation is published; applicants and posts agree (at outset) to be bound by the results

The need for efficient algorithms

At the heart of a centralised matching scheme there is an an *algorithm* (computer program) to find the best matching

Why not just try all possible matchings and choose the best?

- suppose there are n students and n hospitals
- in the worst case we have to generate $n!$ matchings
- if $n=50$, that means 3×10^{64} matchings
- a powerful computer capable of generating 10^6 matchings per second will take 10^{49} centuries to generate all matchings!

We seek *efficient* algorithms - usually need additional insight into the underlying mathematical structure of the problem

1: Introduction