Teaching Material: Stable Matching in Theory and in Practice

Bahar Rastegari, University of Bristol

EASSS 2018

What is in this document?

This document includes some background information on computational complexity, approximation algorithms, integer linear programming and parameterised complexity. If you are not familiar with these topics or want a quick refresher, it is a good idea to have a look at the following sections before the tutorial. I have benefited from the references at the end of this document ([1, 2, 4]) for writing these sections.

1 Computational complexity

Given two functions f and g, we say $f(n) = O(g(n))$ if there are positive constants c and N such that $f(n) \leq c \cdot g(n)$ for all $n \geq N$. An algorithm for a problem has time complexity $O(g(n))$ if its running time f satisfies $f(n) = O(g(n))$ where n is the input size. An algorithm runs in **polynomial time** if its time complexity is $O(n^k)$ for some constant k, where n is the input size.

A *decision problem* is a problem whose solution is "yes" or "no" for any input. Most problems occurring in practice are *optimisation problems*: *for example, given a graph G and two vertices s and t, find a shortest path between s and t in G.* Every optimisation problem has a corresponding decision problem: *for example, given a graph G and two vertices s and t, is there a path between s and t in G with length at most k?*

A decision problem belongs to the complexity class **P** if it can be solved by a polynomial-time algorithm. If an optimisation problem is solvable in polynomial-time, then the corresponding decision problem is also solvable in polynomial time.

A decision problem belongs to the complexity class **NP** if it can be *verified* in polynomial time. That is, if given a *certificate* of a solution, it can be verified in polynomial time whether the certificate is correct. *Example: for the problem of finding a shortest path between s and t of length k, a list of k vertices is a certificate. Once such a list is given, verifying the correctness is easy.* Any problem in **P** is also in **NP** and we write this as **P**$\subseteq$**NP**. One of the most important and challenging open problems in computer science is whether **P**=**NP**.

An algorithm *reduces* a decision problem P to another decision problem B if it transforms any instance α of P into some instance β of B such that (1) the transformation takes polynomial time and (2) α is a "yes" instance of P if and only if β is a "yes" instance of B. Such a reduction implies that B is at least as hard as P.

A decision problem P is **NP**-hard if every other problem in **NP** is reducible to P. Moreover, a decision problem P is **NP**-complete if it is **NP**-hard and it belongs to **NP**. To prove that a problem P is **NP**-hard it is enough to show that a problem B, already known to be **NP**-complete, is reducible to P. If a decision problem is **NP**-complete it has no polynomial-time algorithm unless **P**=**NP**.

So what if a problem is **NP**-complete? Do we give up? Of course not. Various approaches have been explored by computer scientists in order to get around a problem being **NP**-hard. Three of these have been explained in the following sections.

2 Approximation algorithms

An optimisation problem involves maximising or minimising (subject to a suitable measure) over a set of feasible solutions for a given instance. For example, the **graph colouring problem** asks, given a graph G, what is the minimum number of colours required so that we can colour the vertices of G such that no two vertices sharing the same edge have the same color.

Given an optimisation problem P, if P's corresponding decision problem is **NP**-hard it implies that P has no polynomial-time algorithm unless **P**=**NP**. Even though **NP**-hardness is defined for decision problems, sometimes we choose to be sloppy and say that P is **NP**-hard.

An *approximation algorithm* A for an optimisation problem P is a polynomial-time algorithm that produces a feasible solution $A(I)$ for any instance I of P. A has performance guarantee c, for some $c > 1$ if

- $|A(I)| \leq c \cdot opt(I)$ for any instance I (in the case where P is a minimisation problem)
- $|A(I)| \geq (1/c) \cdot opt(I)$ for any instance I (in the case where P is a maximisation problem)

where $|A(I)|$ is the size of the solution produced by A and $opt(I)$ is the size of an optimal solution. We say that A is a c-approximation algorithm for problem P.

3 Integer programming

In an optimisation problem we seek to maximise or minimise an objective given some constraints. In the graph colouring problem, e.g., the constraints are that no two adjacent vertices have the same colour, and the objective is to minimise the number of colours used. If the objective can be specified as a linear function of certain variables, and the constraints as linear equalities or inequalities on those variables, then we have a *linear programming problem*. If for a solution to be feasible the variables need to be integers, then we have an *integer linear programming problem*.

An integer program is formally written as:

$$\min \mathbf{c}^T \cdot \mathbf{x}$$

$$\text{subject to } A\mathbf{x} \leq \mathbf{b}$$

where $\mathbf{c} = (c_1, c_2, \ldots, c_n)^T$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ and $\mathbf{b} = (b_1, b_2, \ldots, b_m)^T$. $A = (a_{i,j})$ $(1 \leq i \leq m, 1 \leq j \leq n)$, the c_i, b_j and a_{ij} are real-valued known coefficents and the x_i are integer-valued variables. Note that we can easily translate problems in which we wish to maximise rather than minimise the objective function into this form.

Linear programming is solvable in polynomial time, whereas general integer programming problem is **NP**-hard. However, there are some powerful solvers out there!

3.1 Example: graph colouring

The graph colouring problem is **NP**-hard. Suppose that $G = (V, E)$ and that $|V| = n$. Clearly, no colouring can use more than n colours. Let us define the following binary variables:

- $x_{v,c}$ for all $v \in V$ and all c, $1 \leq c \leq n$. $x_{v,c} = 1$ if vertex v has colour c and is 0 otherwise.
- y_c for all c, $1 \leq c \leq n$. $y_c = 1$ if colour c is used and 0 otherwise.

We now define the following linear program:

$$
\begin{aligned}
\textbf{min} \quad & \sum_{1 \leq c \leq n} y_c & & \\
\textbf{subject to} \quad & \sum_{1 \leq c \leq n} x_{v,c} = 1 & & \forall v \in V \\
& x_{u,c} + x_{v,c} \leq 1 & & \forall c (1 \leq c \leq n) \forall \{u, v\} \in E \\
& n y_c \geq \sum_{v \in V} x_{v,c} & & \forall c (1 \leq c \leq n) \\
& x_{v,c} \in \{0, 1\} & & \forall v \in V, \forall c (1 \leq c \leq n) \\
\textbf{and} \quad & y_c \in \{0, 1\} & & \forall c (1 \leq c \leq n)
\end{aligned}
$$

The first line of constraints implies that each vertex must have one colour. The second line ensures that adjacent vertices have distinct colours and the third line implies that if a colour c is used then $y_c = 1$. The last two lines of constraints state that variables are all binary-valued. The objective is to minimize the number of colours used.

4 Parameterised Complexity

Parameterised complexity provides a multivariate framework for the analysis of hard problems: if a problem is known to be **NP**-hard, so that we expect the running-time of any algorithm to depend exponentially on some aspect of the input, we can seek to restrict this combinatorial explosion to one or more *parameters* of the problem rather than the total input size. This has the potential to provide an efficient solution to the problem if the parameter(s) in question are much smaller than the total input size. A parameterised problem with total input size n and parameter k is considered to be tractable if it can be solved by a so-called *FPT algorithm*, an algorithm whose running time is bounded by $f(k) \cdot n^{\mathcal{O}(1)}$, where f can be any computable function. Such problems are said to be fixed parameter tractable, and belong to the complexity class **FPT**.

It should be emphasised that, for a problem to be in **FPT**, the exponent of the polynomial must be independent of the parameter value; problems which satisfy the weaker condition that the running

time is polynomial for any constant value of the parameter(s) (so that the degree of the polynomial may depend on the parameters) belong to the class **XP**.

For further background on the theory of parameterised complexity, check these excellent books [2, 3].

References

[1] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill/MIT, 3 edition, 2009.

[2] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer London, 2013.

[3] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[4] David Manlove. The hospitals/residents problem and its variants. Summer School on Matching Problems, Markets and Mechanisms, Budapest, Hungary, 2013.