Lecture 3

Graph Colouring

The material from the first two lectures provides enough background that we can begin to discuss a problem—graph colouring—that is both mathematically rich and practically applicable.

Reading:
The material for this lecture appears, in very condensed form, in Chapter 9 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition (Available from `SpringerLink`)

A somewhat longer discussion, with many interesting exercises, appears in

> John M. Harris, Jeffry L. Hirst and Michael J. Mossinghoff (2008), *Combinatorics and Graph Theory*, 2nd edition. (Available from `SpringerLink`)

3.1 Notions and notation

Definition 3.1. *A* k***-colouring*** *of an undirected graph* $G(V, E)$ *is a function*

$$\phi : V \to \{1, \ldots, k\}$$

that assigns distinct values to adjacent vertices: that is, $(u, v) \in E \Rightarrow \phi(u) \neq \phi(v)$. *If* G *has a* k*-colouring then it is said to be* k***-colourable****.*

I'll refer to the values assigned by $\phi(v)$ as "colours" and say that a graph is k-colourable if one can draw it in such a way that no two adjacent vertices have the same colour. Examples of graphs and colourings include

- K_n, the complete graph on n vertices is clearly n-colourable, but not $(n-1)$ colourable;
- $K_{m,n}$, the complete bipartite graph on groups of m and n vertices, is 2-colourable.

Both classes of example are illustrated in Figure 3.1.

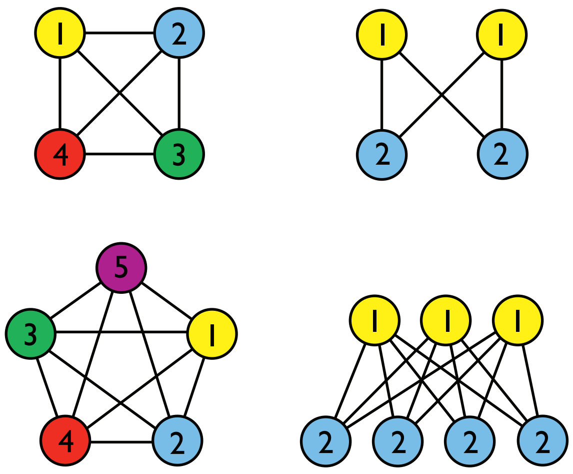


Figure 3.1: *The complete graphs K_4 and K_5 as well as the complete bipartite graphs $K_{2,2}$ and $K_{3,4}$, each coloured using the smallest possible number of colours. Here the colouring is represented in two ways: as numbers giving $\phi(v)$ for each vertex v and with, well, colours (see the electronic version).*

Definition 3.2. *The* ***chromatic number*** $\chi(G)$ *is the smallest number k such that G is k-colourable.*

Thus, as the examples above suggest, $\chi(K_n) = n$ and $\chi(K_{m,n}) = 2$. The latter is a special case of the following easy lemma, whose proof is left as an exercise.

Lemma 3.3. *A graph G has chromatic number $\chi(G) = 2$ if and only if it is bipartite.*

Another useful result is

Lemma 3.4. *If H is a subgraph of G and G is k-colourable, then so is H.*

and an immediate corollary is

Lemma 3.5. *If H is a subgraph of G then $\chi(H) \leq \chi(G)$.*

which comes in handy when trying to prove that a graph has a certain chromatic number.

The proof of Lemma 3.4 is straightforward: the idea is that constraints on the colours of vertices arise from edges and so, as every edge in H is also present in G, it can't be any harder to colour H than it is to colour G. Equivalently: if we have a colouring of G and want a colouring of H we can simply use the same colour assignments. To be more formal, say that the vertex and edge sets of G are V and E, respectively, while those of H are $V' \subseteq V$ and $E' \subseteq E$. If a map $\phi_G : V \to \{1, \ldots, k\}$ is a k-colouring of G, then $\phi_H : V' \to \{1, \ldots, k\}$ defined as the restriction of ϕ_G to V' produces a k-colouring of H.

Lemma 3.5 then follows from the observation that, although Lemma 3.4 assures us that H has a colouring that uses $\chi(G)$ colours, it may also be possible to find some other colouring that uses fewer.

3.2 An algorithm to do colouring

The chromatic number $\chi(G)$ is defined as a kind of ideal: it's the minimal k for which we can find a k-colouring. This might make you suspect that it's hard to find $\chi(G)$ for an arbitrary graph—how could you ever know that you'd used the smallest possible number of colours? And, aside from a few exceptions such as those in the previous section, you'd be right to think this: there is no known fast (we'll make the notion of "fast" more precise soon) algorithm to find an optimal (in the sense of using the smallest number of colours) colouring.

3.2.1 The greedy colouring algorithm

There is, however, a fairly easy way to to compute a (possibly non-optimal) colouring $c : V \to \mathbb{N}$. The idea is to number the vertices and then, starting with $c(v_1) = 1$, visit the remaining vertices in order, assigning them the lowest-numbered colour not yet used for a neighbour. The algorithm is called *greedy* because it has no sense of long-range strategy: it just proceeds through the list of vertices, blindly choosing the colour that seems best at the moment.

Algorithm 3.6 (Greedy colouring)**.**
Given a graph G with edge set E, vertex set $V = \{v_1, \ldots, v_n\}$ and adjacency lists A_v, construct a function $c : V \to \mathbb{N}$ such that if the edge $e = (v_i, v_j) \in E$, then $c(v_i) \neq c(v_j)$.

```
(1) Initialize
    Set $c(v_j) \leftarrow 0$ for all $1 \le j \le n$
    $c(v_1) \leftarrow 1$
    $j \leftarrow 2$

(2) $c(v_j) \leftarrow \min\big(k \in \mathbb{N} \mid k > 0 \text{ and } c(u) \neq k \;\forall u \in A_{v_j}\big)$

(3) Are we finished? Is $j = n$?
      • If so, stop: we've constructed a function $c$ with the desired properties.
      • If not, set $j \leftarrow (j + 1)$ and go to step (2).
```

Remarks

- The algorithm above is meant to be explicit enough that one could implement it in R or MATLAB. It thus includes expressions such as $j \leftarrow 2$ which means "set j to 2" or "j gets the (new) value 2". The operator $\leftarrow$ is sometimes called the *assignment* operator and it appears in some form in all the programming languages I know. Sometimes it's expressed with notation like `j = j + 1`, but this is a jarring, nonsensical-looking thing for a mathematician and so I'll avoid it.

- We will discuss several more algorithms in this course, but will not be much more formal about how they are specified. This is mainly because a truly rigorous account of computation would take us into the realms of computability theory, a part of mathematical logic, and would require much of the rest of the term, leaving little time for our main subjects.

Finally, to emphasize further the mechanical nature of greedy colouring, we could rewrite it in a style that looks even closer to MATLAB code:

Algorithm 3.7 (Greedy colouring: as pseudo-code)**.**
Given a graph G with edge set E, vertex set $V = \{v_1, \dots, v_n\}$ and adjacency lists A_v, construct a function $c : V \to \mathbb{N}$ such that if the edge $e = (v_i, v_j) \in E$, then $c(v_i) \neq c(v_j)$.

```
(1) Set $c(v_j) \leftarrow 0$ for $1 \leq j \leq n$.
(2) $c(v_1) \leftarrow 1$.
(3) for $2 \leq j \leq n$ {
(4)         Choose a colour $k > 0$ for vertex $v_j$ that differs from those of its neighbours
            $c(v_j) \leftarrow \min\left(k \in \mathbb{N} \mid k > 0 \text{ and } c(u) \neq k \ \forall u \in A_{v_j}\right)$
(5) } End of loop over vertices $v_j$.
```

Both versions of the algorithm perform exactly the same steps, in the same order, so comparison of these two examples may clarify the different approaches to presenting algorithms.

3.2.2 Greedy colouring may use too many colours

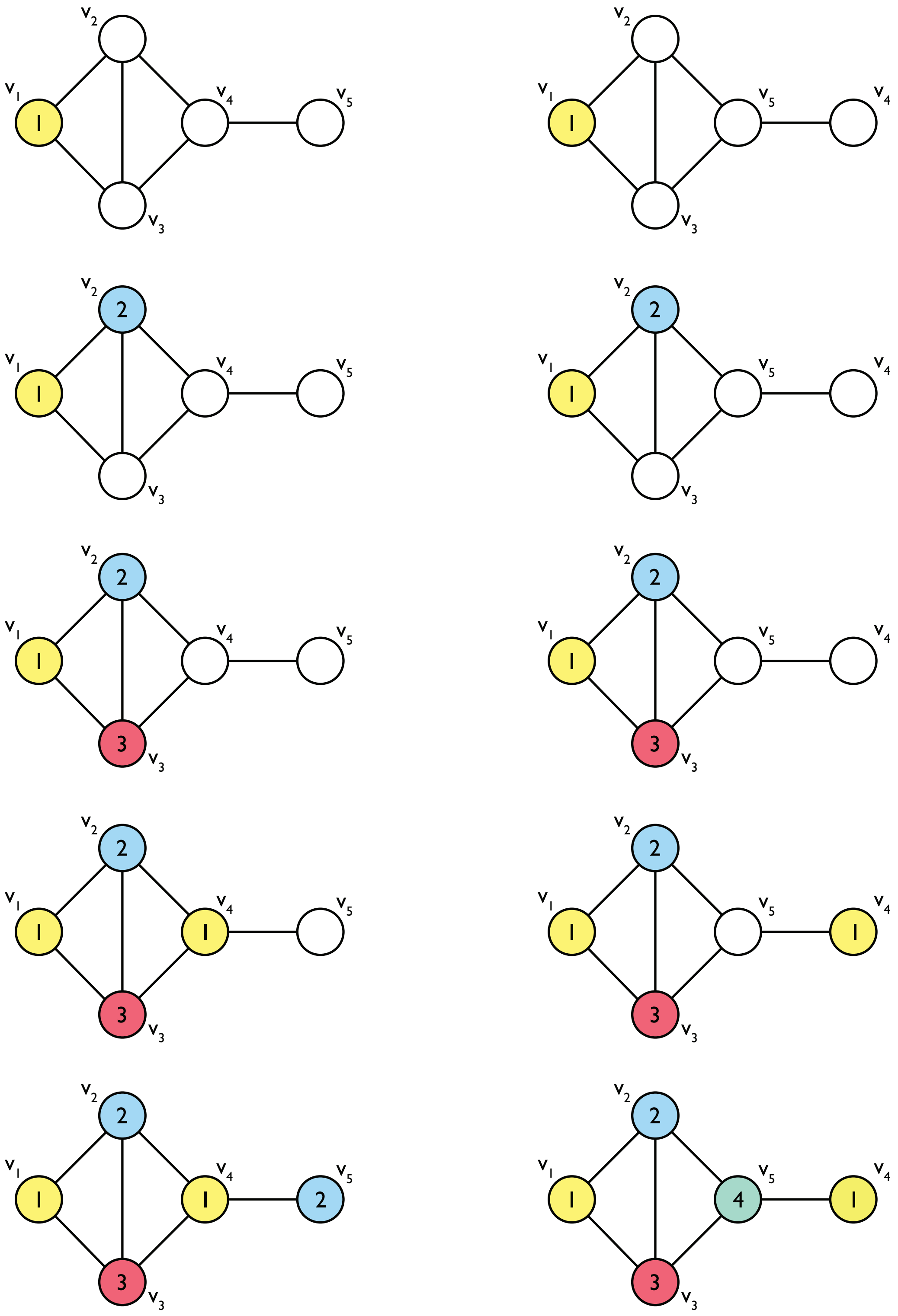
If we use Algorithm 3.7 to construct a function $c : V \to \mathbb{N}$, then we can regard it as a k-colouring by setting $\phi(v_j) = c(v_j)$, where k is given by

$$k = \max_{v_j \in V} c(v_j). \tag{3.1}$$

For the reasons discussed above, this k provides only an upper bound on the chromatic number of G. To drive this point home, consider Figure 3.2, which illustrates the process of applying the greedy colouring algorithm to two graphs, one in each column.

For the graph in the left column—call it G_1—the algorithm produces a 3-colouring, which is actually optimal. To see why, notice that the subgraph consisting of vertices v_1, v_2 and v_3 (along with the associated edges) is isomorphic to K_3. Thus we need at least 3 colours for these three vertices and so, using Lemma 3.5, we can conclude that $\chi(G_1) \geq 3$. On the other hand, the greedy algorithm provides an explicit example of a 3-colouring, which implies that $\chi(G_1) \leq 3$, so we have proven that $\chi(G_1) = 3$.

The graph in the right column–call it G_2—is isomorphic to G_1 (a very keen reader could write out the isomorphism explicitly), but its vertices are numbered differently and this means that Algorithm 3.7 colours them in a different order and arrives at a sub-optimal k-colouring with $k = 4$.



v_2 v_1 1 v_4 v_5 v_3
v_2 v_1 1 v_5 v_4 v_3
v_2 2 v_1 1 v_4 v_5 v_3
v_2 2 v_1 1 v_5 v_4 v_3
v_2 2 v_1 1 v_4 v_5 3 v_3
v_2 2 v_1 1 v_5 v_4 3 v_3
v_2 2 v_1 1 v_4 1 v_5 3 v_3
v_2 2 v_1 1 v_5 v_4 1 3 v_3
v_2 2 v_1 1 v_4 1 v_5 2 3 v_3
v_2 2 v_1 1 v_5 4 v_4 1 3 v_3

Figure 3.2: *Two examples of applying Algorithm 3.7: the colouring process runs from the top of a column to the bottom. The graphs in the right column are the same as those in the left, save that the labels on vertices 4 and 5 have been switched. As in Figure 3.1, the colourings are represented both numerically and graphically.*

3.3 An application: avoiding clashes

I'd like to conclude by introducing a family of applications that involve avoiding some sort of clash—where some things shouldn't be allowed to happen at the same time or in the same place. A prototypical example is:

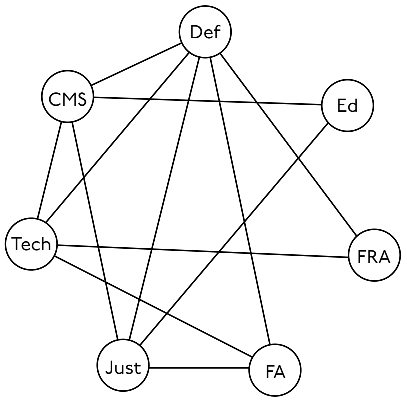
Example 3.8. *Suppose that a group of ministers serve on committees as described below:*

Committee	***Members***
Culture, Media & Sport	*Alexander, Burt, Clegg*
Defence	*Clegg, Djanogly, Evers*
Education	*Alexander, Gove*
Food & Rural Affairs	*Djanogly, Featherstone*
Foreign Affairs	*Evers, Hague*
Justice	*Burt, Evers, Gove*
Technology	*Clegg, Featherstone, Hague*

What is the minimum number of time slots needed so that one can schedule meetings of these committees in such a way that the ministers involved have no clashes?

One can turn this into a graph-colouring problem by constructing a graph whose vertices are committees and whose edges connect those that have members in common: such committees can't meet simultaneously, or their shared members will have clashes. A suitable graph appears at left in Figure 3.3, where, for example, the vertex for the Justice committee (labelled Just) is connected to the one for the Education committee (Ed) because Gove serves on both.

The version of the graph at right in Figure 3.3 shows a three-colouring and, as the vertices CMS, Ed and Just form a subgraph isomorphic to K_3, this is the smallest number of colours one can possibly use and so the chromatic number of the committee-and-clash graph is 3. This means that we need at least three time slots to schedule the meetings. To see why, think of a vertex's colour as a time slot: none of the vertices that receive the same colour are adjacent, so none of the corresponding committees share any members and thus that whole group of committees can be scheduled to meet at the same time. There are variants of this problem that involve, for example, scheduling exams so that no student will be obliged to be in two places at the same time or constructing sufficiently many storage cabinets in a lab so that chemicals that would react explosively if stored together can be housed separately: see this week's Problem Set for another example.



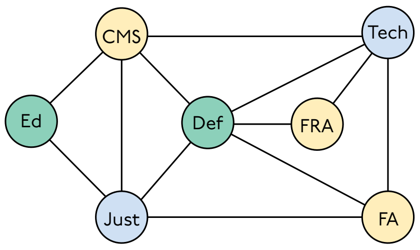


Figure 3.3: *The graph at left has vertices labelled with abbreviated committee names and edges given by shared members. The graph at right is isomorphic, but has been redrawn for clarity and given a three-colouring, which turns out to be optimal.*