Graph Colouring

1 Problems in natural science

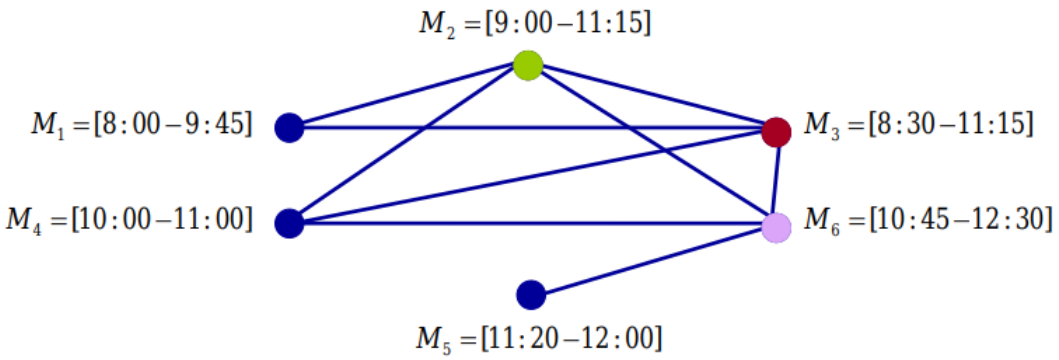
- Aim: Solve a ”real world” problem
- How: most times it is difficult to try ”real world” solutions until we find the best one
- Therefore
  - We rephrase the problem as an ”abstract” problem
  - This is called abstract modelling
  - We can easily
    - Try many alternative solutions to the abstract problem
    - and then ”translate” these solutions in terms of the original ”real world problem
- Usually the ”abstract model” involves graphs/networks

2 Radio Frequency assignment

- A network of radio transmitters
- Each of them must have an operating frequency
- if two nearby transmitters have the same frequency they **interfere** with each other
- We **pay** for every frequency we are using
- areal world problem: assign the frequencies such that:
  - no transmitters interfere
  - we pay as little as possible
- To solve the problem, we do not have to deal with the transmitters themselves
- We consider a graph problem
  - transmitter $\leftrightarrow$ vertex
  - one edge between two nearby transmitters
  - frequency of transmitter $\leftrightarrow$ colour of its vertex
- Our goal becomes graph colouring
  - assign the smallest number of colours to vertices where two connected vertices have different colour

3 Scheduling of talks

- A conference agency organises a set of business meetings
- each meeting has pre-defined start/finish times
- A hotel room can be used for many meetings, if they do not overlap
- The agency **pays** for every hotel room they book
- Real world problem: book hotel rooms such that:
  - All meetings take place
  - The agency pays as little as possible
- Let the meetings have the following start/finish times:



- Again, we consider a **graph colouring problem**:
  - meeting ↔ vertex
  - one edge between two overlapping meetings
  - room for a meeting ↔ colour of its vertex

4 Timetable of exams

- each exam takes one day
- two exams can be held concurrently if no student is registered in both of them
- Real world problem: create a timetable such that:
  - each student participates in all his/her exams
  - The exams **finish as soon as possible**
- Again we consider a graph colouring problem:
  - exam ↔ vertex
  - one edge between two exams with common students
  - Date of an exam ↔ colour of its vertex

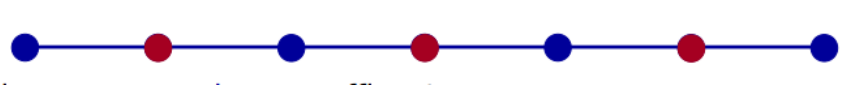
5 Graph colouring

All these problems (and many others) can be "rephrased" in terms of graph colourings, where:

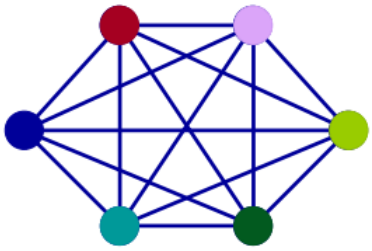
- The vertices correspond to transmitters, meetings, exams,...
- The colours correspond to frequencies, rooms, dates,...

But always:

- The edges correspond to some kind of **conflicts** (nearby transmitters, overlapping meetings, common students,...)
- In general, it is hard (NP-complete) to compute the smallest number of colours needed for a given graph
- However, in special cases it is easy:
  - What an graph is a path:



    always two colours suffice
  - When a graph is **complete** with k vertices



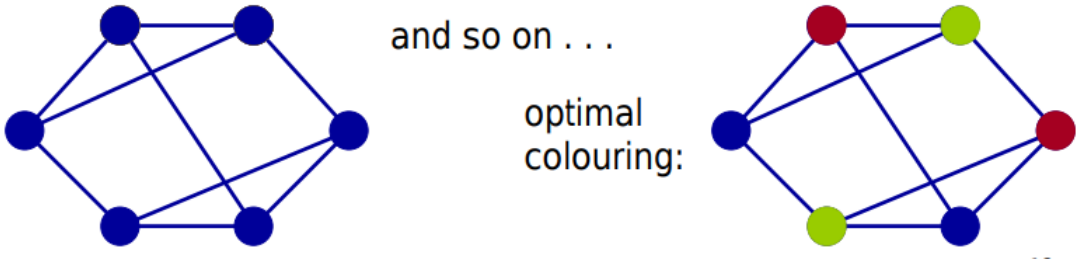
always k colours needed (every vertex is a different colour)

- Thus, if a graph includes a **complete subgraph** with **k vertices**
  - We need at least k colours for this graph
  - When a graph is complete with k vertices, k colours are always needed (each vertex a different colour)

6 A simple algorithm

How to check whether 3 colours are enough?

- The simplest algorithm: ”brute force”, i.e. enumerate exhaustively **all possible 3-colourings** until you:
  - either find a 3-colouring, where any two adjacent vertices have different colours
  - or otherwise conclude that you need more colours



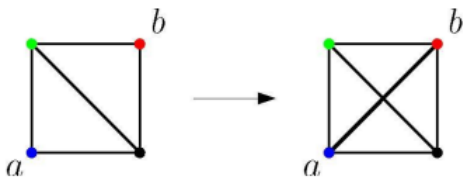
Let’s count how many steps we need (for n vertices)

- 3 choices for the 1st vertex, 3 choices for the 2nd vertex and so on
- All together 3^n steps, which is too many

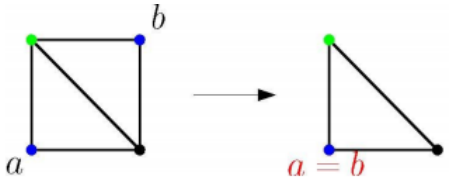
7 A more ”delicate” algorithm

We can ”refine” our search of a good colouring:

- If two vertices a and b are **not connected**, then in the optimal colouring:
  1. either a and b have **different** colours
  2. if a and b have the **same** colour
- In the first case: we can just add an edge between the vertices a and b

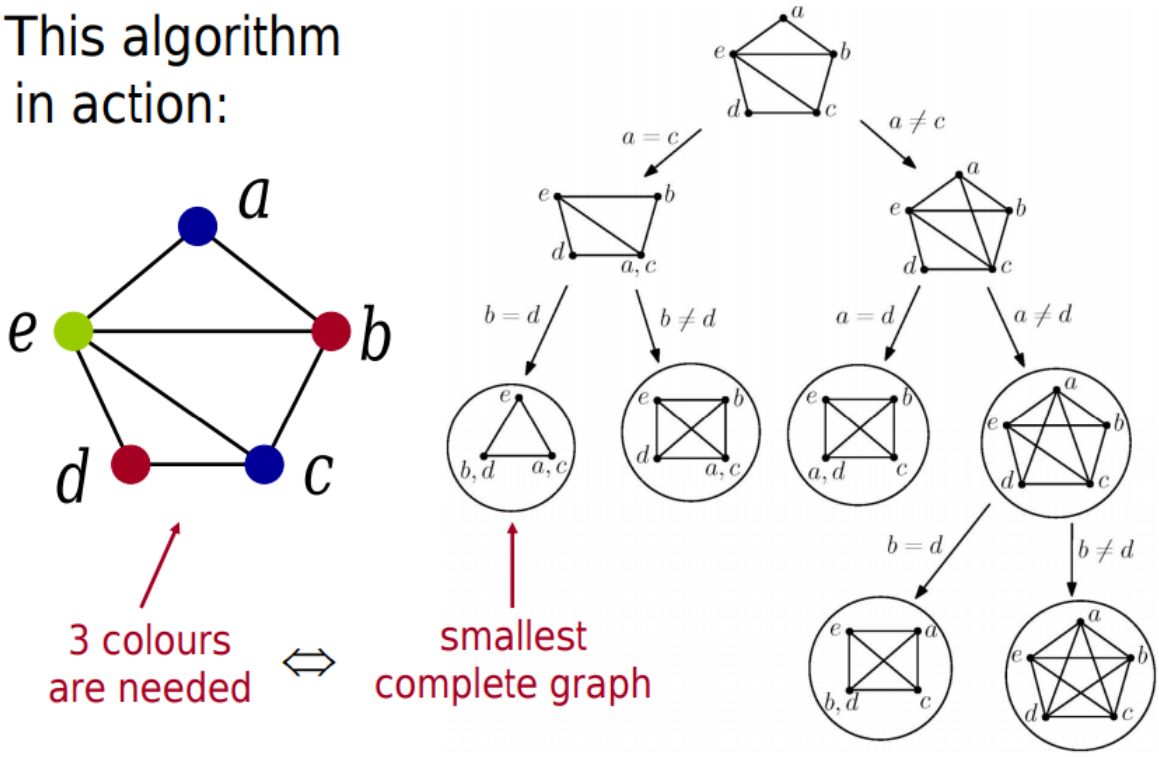


- In the second case: we can just ”merge” the two vertices a and b into one



This algorithm in action:

- At every iteration:
  - pick up a pair of non connected vertices a and b
  - try both possibilities a and b have
    - the same colour
    - different colours
  - For each case, iterate: find another pair of non connected vertices, iterate ...
- Stop when **no iteration** is possible i.e. when we end up only with complete graphs
- The smallest size of these complete graphs is equal to the smallest number of colours we need



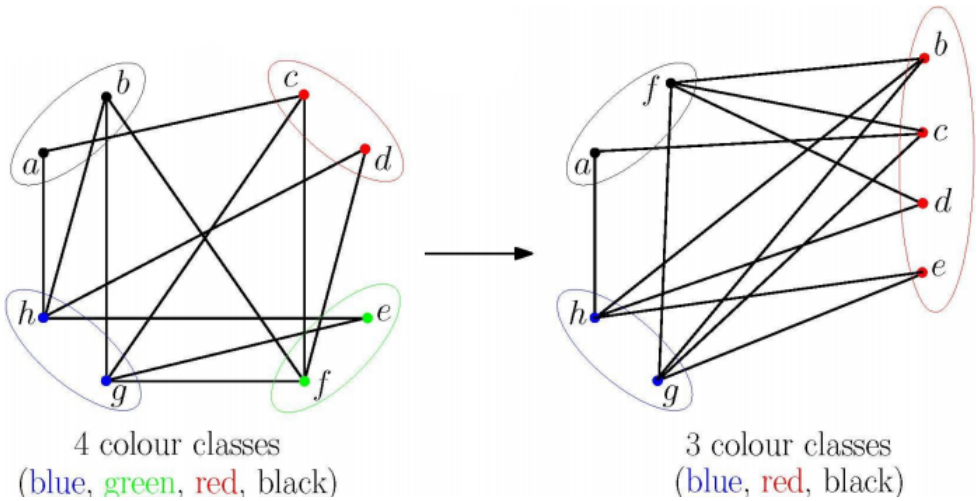
- However also this algorithm also has **exponential running time in the worst case**
- This is called backtracking:
  - try something (merge/add edge)
  - Then ”undo” and try something else

8 A different view of colouring

If a graph can be coloured with **k colours**, then:

- each two vertices a and b with the same colour are not connected to each other, i.e.

- The graph can be partitioned into k ”colour classes” each having **no edges**



Algorithmically, this formulation doesn’t help much

A graph is called k-partite, if it can be coloured with k different colours