Vertex Colouring

Complement to Chapter 7, "The Case of the Hooded Man"
and Chapter 9, "The Sudoku Apprentice"

In 1852, a young Englishman, Francis Guthrie, wondered if it were always possible to colour a map using four colours, while respecting the condition that two neighbouring countries could not be of the same colour. It wasn't until 1976 that two American researchers, Kenneth Appel and Wolfgang Haken, from the University of Illinois, managed to answer this question in the affirmative. More than a century went by between when the problem was set out and when it was solved, even though it appears to be very simple. Throughout those years, researchers of course did tackle the question, and many attempts gave rise to new mathematical developments, as well as to the problem being formulated in terms of graphs. The map to colour was replaced by a graph, each country being represented by a vertex and two neighbouring countries being linked by an edge.

Definition

Colouring the vertices in graph G means assigning colours to the vertices such that each edge in G has ends of two different colours. We are also usually seeking to determine a colouring that uses as few colours as possible. The smallest number of colours needed to colour the vertices in a graph G is called the "chromatic number" of G and is noted as $\chi(G)$.

We can also colour the edges while making sure that edges touching a given vertex are not of the same colour. This is what we defined earlier, in the document dealing with matchings.

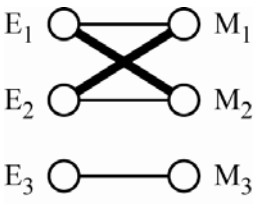
Definition

Colouring the edges in a graph G means assigning colours to the edges such that the edges which have an end in common with one another are of different colours. We are also generally seeking to determine a colouring that uses as few colours as possible. The smallest number of colours needed to colour the edges of a graph G is called the "chromatic index" of G and is noted as q(G).

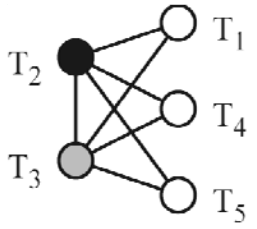
Let's once again take the edge colouring example we saw in an earlier document. We need to carry out five tasks, T_1, T_2, T_3, T_4 and T_5, with each of the tasks taking one day to complete. Tasks T_1 and T_2 must be performed by employee E_1, tasks T_3 and T_4 by employee E_2 and task T_5 by employee E_3. Tasks T_1 and T_3 require machine M_1, tasks T_2 and T_4 need machine M_2 and task T_5 machine M_3. Knowing that each employee can only carry out one task at a time and that each machine can only be used by one employee at a time, how many days are needed at minimum to perform the five tasks?

This problem is equivalent to colouring the edges of the graph below using the minimum number of colours, such that all the edges that touch one another are of different colours.

Each colour corresponds to one day, and we need two days. For example, we can perform tasks T_1, T_4 et T_5 on the first day, and tasks T_2 and T_3 the second day.



Let's consider the same problem now, but with an additional constraint. Tasks T_2, T_3 and T_5 monopolize the company's only remote-control machine, and it cannot help with the performance of two tasks simultaneously. The minimum number of days needed for carrying out the five tasks can be determined by colouring the vertices in the graph below with a minimum number of colours, such that no edges have two ends of the same colour. Once again, each colour corresponds to one day. So we need three days. For example, we can perform tasks T_1, T_4 and T_5 the first day, task T_2 the second day and task T_3 the third day.



In fact, many concrete problems can boil down to seeking to partition a set of objects into subsets that feature only elements that are compatible two by two. For instance, determining a school schedule means partitioning time into periods during which only courses that can take place simultaneously are given. We can model this type of problem in terms of colouring the vertices of a graph: the vertices are the objects being partitioned and the edges represent the incompatibilities between these objects. For the school schedule example, the vertices are the courses that need to be placed in the timetable, and we link two courses with an edge if they can't be given simultaneously (because they are taught by the same teacher, are given to the same students, or must be taught in the same room).

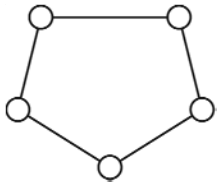
The problem of colouring the edges in a graph was addressed in an earlier document. Here, we are interested in determining the chromatic number. Note that when considering the line graph L(G) of a graph G, we know of course that colouring the edges of G is equivalent to colouring the vertices of L(G). The chromatic index of G is therefore equal to the chromatic number of its line graph, which can be noted as:

$$q(G) = \chi(L(G))$$

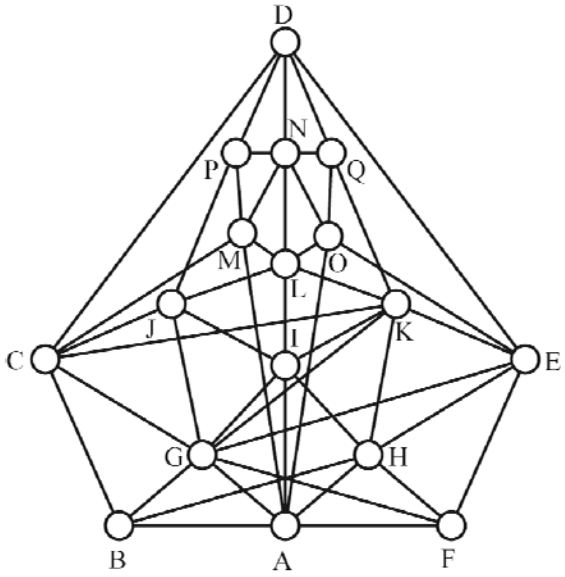
Definition

A "clique" in a graph is a set of vertices all related two by two. The size of the biggest clique in a graph G is noted as $\omega(G)$.

Given that the vertices in a clique must all be of different colours, we deduce that the chromatic number $\chi(G)$ of G cannot be less than $\omega(G)$. It may, however, be strictly greater than $\omega(G)$, such as in the case below, with the pentagon example. We need $\chi(G)=3$ colours to colour its vertices, while $\omega(G)=2$.



As another example, we can provide the graph Manori draws to determine the guilty party in the robbery of Mrs. Rossier's grocery store.

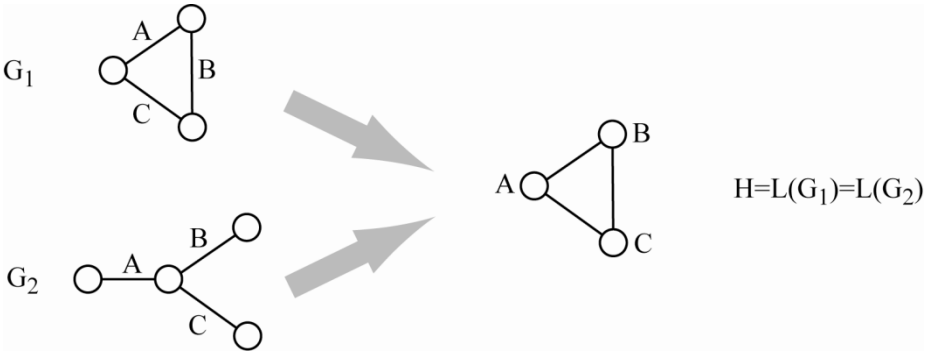


Manori demonstrated that while the graph contains no cliques with four vertices, its chromatic number is greater than 3. We can only colour it in three colours if we remove vertex I.

As a reminder, Vizing showed that the chromatic index is always at most equal to the highest degree + 1, or $q(G) \leq \Delta(G)+1$. We can make the following observations:

- A set of edges in G that all have one end in common corresponds to a clique in its line graph L(G), which shows that $\omega(L(G)) \geq \Delta(G)$;
- If L(G) is not a triangle, a clique in L(G) corresponds to a set of edges all touching a given vertex in G, which shows that $\Delta(G) \geq \omega(L(G))$.

In sum, if L(G) is not a triangle, we get $\Delta(G)=\omega(L(G))$. If L(G) is a triangle, as we already noted in another document, and as illustrated once again below, it is possible that G is also a triangle, in which case $2=\Delta(G) < \omega(L(G))=3$.

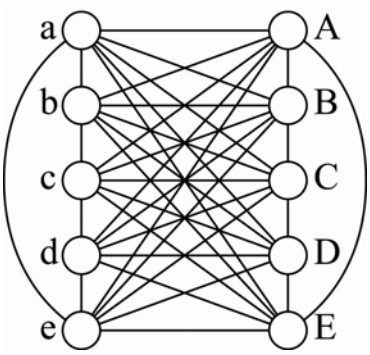


Given that $q(G)=\chi(L(G))$, the result demonstrated by Vizing can be rewritten as follows:

$$\chi(L(G)) \leq \omega(L(G))+1.$$

By this reasoning laid out above, this result is only true if L(G) is not a triangle. It is also true if L(G) is a triangle because in that case, $3=\chi(L(G)) < \omega(L(G))+1=4$.

In other words, we have just seen that the chromatic number of a line graph is never much higher than the size of its biggest clique. This result, however, is no longer necessarily valid for all graphs. It has been demonstrated that the difference between $\chi(G)$ and $\omega(G)$ can be as big as you can imagine. Here is an example in which $\chi(G) = \omega(G)+2$. This graph was built as follows. We first considered two pentagons, one made up of vertices a, b, c, d and e, and the other of vertices A, B, C, D and E. We then added an edge between all the vertices in the first pentagon and those in the second pentagon.



Given that we need three colours to colour a pentagon, we need six colours to colour this graph. Also, the biggest clique in a pentagon contains two vertices, which means that the biggest clique in the graph as a whole is of size 4. In sum, we get $\chi(G)=6=4+2=\omega(G)+2$.

The problem of determining the chromatic number of a graph G is difficult to solve. The best algorithms known to date do not allow us to determine the chromatic number of graphs with more than 100 vertices. These problems can, however, be easier for certain

graph families. For example, for a bipartite graph that includes at least one edge, we know that $\chi(G)=2$ because we can assign one colour to one of the parts of the V partition, and another colour to the second part. We also saw that it's easy to colour an interval graph. For other large graphs, we have no choice but to rely on heuristics, which provide solutions of a reasonable quality in a reasonable time. Here, let's look at a few of these heuristics.

Constructive methods for colouring the vertices of a graph travel over the vertices sequentially, assigning each vertex the smallest possible colour (the colours are numbered). The quality of the solution thus obtained depends largely on the order in which the vertices are considered. This order can be chosen to start with, or built dynamically. It is important to note that all graphs possess a vertex order based on which the constructive method produces a colouring using the minimum number of colours. To see this, we simply need o consider the colouring of graph G in $\chi(G)$ colours and order the vertices of G by first taking the vertices of colour 1, then those of colour 2, and so forth.

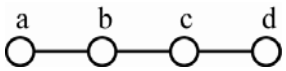
Property

A constructive method never uses more than $\Delta(G)+1$ colours.

Proof

When we colour a vertex, a maximum of $\Delta(G)$ colours can already appear in its neighbourhood. □

It is interesting to note that a sequential colouring algorithm can be in error on the following graph, which only contains four vertices.



If we colour this graph in the order a, d, b, c, we will assign colour 1 to a and d, colour 2 to b and colour 3 to c. The graph could, however, easily be coloured using only two colours (colour 1 for a and c and colour 2 for b and d). This graph is noted P_4 for "path on 4 vertices."

Chvátal demonstrated that if a graph contains no quadruplet of vertices a, b, c and d with a P_4 structure (meaning that a is linked to b but not to c and d, b is linked also to c but not to d, and c is also linked to d), then the constructive sequential algorithm will produce an optimal colouring in $\chi(G)$ colours, regardless of the order used.

The most common constructive colouring methods can be described as follows. First, among the constructive methods based on static orders, the best known are the following:

RANDOM

The vertices are sorted in a random order.

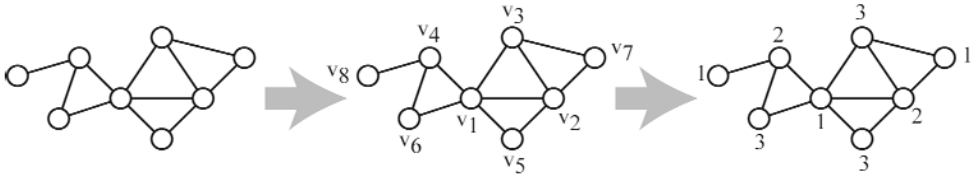
LF (Largest First)

The vertices are sorted by degree in non-increasing order.

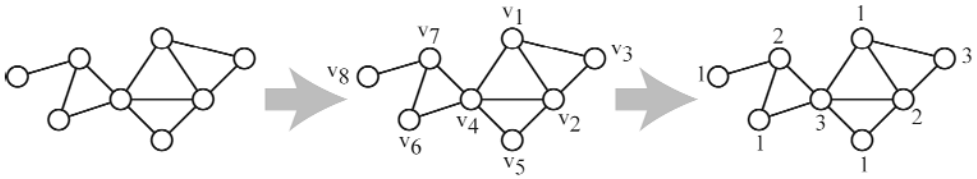
SL (Smallest Last)

The order $v_1, v_2, \ldots, v_{|V|}$ of the vertices is such that v_i is of the smallest degree in the graph containing only vertices $v_1, v_2, \ldots, v_i$.

Illustration



Colouring with the LF algorithm



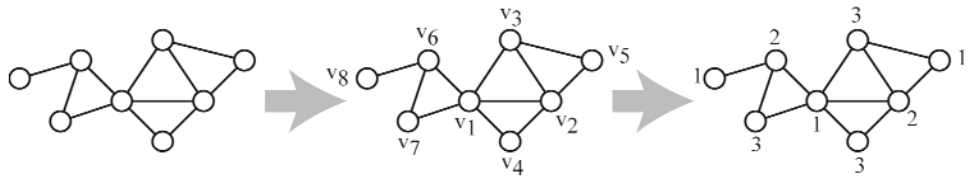
Colouring with the SL algorithm

Given any partial colouring of G, let's define the *degree of saturation* DSAT(v) of a vertex v in G as the number of different colours already assigned to the neighbouring vertices of v. In a subset W of vertices in G, $DEG_W(v)$ measures the number of w vertices in W that are neighbours of v. In what follows, subset A of vertices corresponds to the set of vertices that are not yet coloured. Among the constructive methods based on dynamic orders, the most frequently used is this one:

DSATUR

The order of vertices is built by choosing, at every step, a vertex v of A, which maximizes DSAT(v). If two vertices, v and w, are of the same degree of saturation, v precedes w as long as $DEG_A(v) \geq DEG_A(w)$.

Illustration



Colouring with the DSATUR algorithm

The table below provides a few indications on the performances of these four heuristics when we apply them to randomly generated graphs. We note as $G_{n,d}$ a random graph made up of n vertices and in which each edge has a d probability of existing, independently of the other edges. The probability d is called the *density* of the graph. The results of the table below are averages calculated from ten random graphs. For each

heuristic, we provide the average number of colours used to colour the graph $G_{n,d}$ in question.

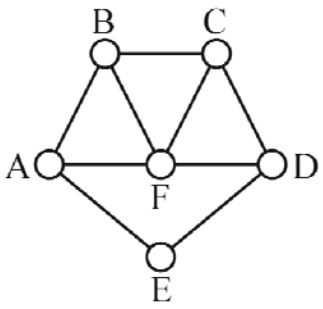
n	d	RANDOM	LF	SL	DSATUR
100	0.4	17.60	16.10	16.65	14.70
	0.5	21.35	20.15	20.50	18.45
	0.6	26.20	24.50	24.85	22.60
300	0.4	39.00	37.10	38.10	34.30
	0.5	48.20	46.00	46.80	43.30
	0.6	61.00	58.20	58.90	53.70
500	0.4	57.20	56.00	56.80	51.00
	0.5	72.40	69.20	71.80	65.00
	0.6	91.40	87.60	89.60	82.80
1000	0.4	99.50	96.50	97.00	92.00
	0.5	126.50	122.00	124.50	116.00
	0.6	160.50	155.00	157.00	146.50

For random graphs with up to 1,000 vertices, all these methods have calculation times of a second or two, or even less. Note that much more efficient algorithms exist, but they are heavier to implement. For instance, for a 1,000-vertex graph with a density of 0.5, the results of the table above are between 116 and 127 colours, while the best algorithms known to date manage to determine colourings of these same graphs with 82 colours (but with several hours of calculation time).

We conclude this document by coming back to Manori once again, who used the colouring of vertices in a graph to find the solution for the first Sudoku puzzle he was given by young Lei. In fact, all Sudoku problems come down to colour an 81-vertex graph in nine colours with a few vertices that already have colours assigned. This graph contains one vertex per box and two vertices are linked by an edge if they correspond with boxes that are found on the same line, in the same column, or in the same 3 x 3 square in the grid.

For the grid given to him by Lei, Manori only considers the six specific boxes represented below.

					5	7		6
	3		1		A	B	9	F
							5	C
	1	9	5		E		4	D
					3			7
					6			
					7			1
								3



Manori notices that these six boxes can only contain the numbers 2, 4 and 8. So he needs to colour the graph using three colours, one for 2, another for 4 and the third for 8. He then notes that the only possible solution assigns one colour to A and C, a second colour to B and D, and finally a third colour to E and F. Then, it's simply a matter of observing that boxes D and E cannot be filled with the number 4 to conclude that the colour representing the number 4 can only be assigned to boxes A and C.