Chapter 5
Applications and Extensions

We are now at a point in this book where we have seen a number of different algorithms for the graph colouring problem and have noted many of their relative strengths and weaknesses. In this chapter we now present a range of problems, both theoretical- and practical-based, for which such algorithms might be applied including face colouring, edge colouring, pre-colouring, solving Latin squares and sudoku puzzles and testing for short circuits on printed circuit boards. Note that these problems actually represent special cases of the general graph colouring problem in that their underlying graphs conform to specific topologies, as we shall see.

This chapter also considers a variant of the graph colouring problem where not all of the graph is visible to an algorithm, or where the graph is subject to change over time. Such problems can arise when setting up wireless ad hoc networks and also in some timetabling applications. We then go on to consider problems that *extend* and therefore generalise the graph colouring problem, specifically list colouring, equitable colouring and weighted graph colouring. Detailed real-world applications of graph colouring are also the subject of Chapters 6, 7 and 8.

Note that, in contrast to the rest of this book, the first two sections of this chapter are concerned with colouring the *faces* of graphs and the *edges* of graphs as opposed to the vertices. As we will see, the latter two problems can be converted into equivalent formulations of the vertex colouring problem using the concepts of *dual graphs* and *line graphs* respectively. However, it is often useful for face and edge colouring problems to be considered as separate problems; hence we will often use the term "vertex colouring" instead of "graph colouring" to avoid any ambiguities.

5.1 Face Colouring

In the face colouring problem we want to colour the *spaces* between vertices and edges, as opposed to the vertices themselves. Face colouring is specifically concerned with planar graphs which, as we saw in Chapter 1, are graphs that can be

R.M.R. Lewis, *A Guide to Graph Colouring*,
DOI 10.1007/978-3-319-25730-3_5

drawn on a plane so that no edges cross one another. When drawn in this way planar graphs can be divided into faces, including one unbounded face that surrounds the graph. Figure 5.1, for example, shows a planar graph comprising ten faces: nine bounded faces and one unbounded face (numbered 10 in the figure). The *boundary* of a face is the set of edges that surrounds it and, when the face is bounded, these edges form a cycle.

It is evident by inspecting Figure 5.1 that the number of faces seems to be related to the number of vertices and edges of the graph. In fact, this relationship can be stated explicitly due to an elegant theorem that was first noted in the 1700s by mathematician Leonhard Euler (1707–1783):

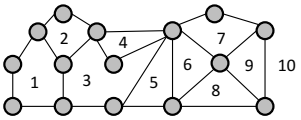

Fig. 5.1 Planar graph with $n = 15$ vertices, $m = 23$ edges and $f = 10$ faces

Theorem 5.1 (Euler's characteristic) *Let G be a planar graph with n vertices, m edges, and f faces. Then*

$$n - m + f = 2$$

Proof. The proof is via induction on the number of faces f. If $f = 1$, then the graph contains no cycles and must therefore be a tree. Since the number of edges in a tree $m = n - 1$, the theorem holds because $n - (n-1) + 1 = 2$.

Now assume $f \geq 2$, meaning that G must now contain at least one cycle. Let $\{u, v\}$ be an edge in one of these cycles. Since this cycle divides two faces, say F_1 and F_2, removing $\{u, v\}$ from G to form a subgraph G' will have the effect of joining F_1 and F_2 with all other faces remaining unchanged. Hence G' has $f - 1$ faces.

Let n', m', and f' be the number of vertices, edges, and faces in G'. Thus, $n' = n$, $m' = m - 1$, $f' = f - 1$, and $n - m + f = n' - m' + f' = 2$. □

We see that Euler's characteristic does indeed hold for the example graph in Figure 5.1 since $n - m + f = 15 - 23 + 10 = 2$ as required.

When considering the face colouring problem it is necessary to restrict ourselves to planar graphs that contain no *bridges*. A bridge is defined as an edge in a graph G whose removal increases the number of components. When a graph contains a bridge $\{u, v\}$, the unbounded face will surround the graph, but will also feature $\{u, v\}$ on its boundary twice, making it impossible to colour. Hence planar graphs containing bridges are not considered further in this section.

Let us now consider the maximum number of edges that a graph can feature while retaining the property of planarity. Consider a connected planar graph G with n vertices, m edges, and f faces. Also write f_i for the number of faces in G that are surrounded by exactly i edges in their boundaries. Clearly $\sum_i f_i = f$ and, assuming that G does not contain a bridge,

$$\sum_i = if_i = 2m \tag{5.1}$$

since every edge is on the boundary of exactly two faces. We can use this relationship in conjunction with Euler's characteristic to give an upper bound on the number of edges in a planar graph. This result also involves knowledge of the *girth* of a graph, defined as follows.

Definition 5.1 *The* girth *of a graph G is the length of the shortest cycle in G. If G is acyclic (i.e., contains no cycles), then its girth equals infinity.*

Theorem 5.2 *Let G be a planar graph with $n \geq 3$ vertices, m edges, f faces and no bridges. Then G has at most $3n-6$ edges. Furthermore, if G has a girth g (for $3 \leq g \leq \infty$), then:*

$$m \leq \max\left\{\frac{g}{g-2}(n-2), n-1\right\}$$

Proof. For $g = 3$, we get $\max\{\frac{3}{3-2}(n-2), n-1\} = 3(n-2)$, giving $m \leq 3n-6$ as required. Hence we only need to prove the second assertion above.

If $g > n$, then this implies $g = \infty$ meaning that G has no cycles and is therefore a tree. Hence, $m = (n-1) \leq n$. Now assume that $g \leq n$ and that the assertion holds for smaller n's. Also assume without loss of generality that G is connected. From earlier we know that:

$$2m = \sum_i if_i = \sum_{i \geq g} if_i \geq g\sum_i f_i = gf.$$

Hence by Euler's characteristic (Theorem 5.1), we get:

$$m + 2 = n + f \leq n + \frac{2}{g}m$$

and so

$$m \leq \frac{g}{g-2}(n-2)$$

as required. □

Theorem 5.2 can often be used to decide whether a graph is planar or not. For example, the complete graph with five vertices K_5 cannot possibly be planar because it has $n = 5$ vertices and $m = 10$ edges, meaning $m \leq 3n-6$ is not satisfied. As another example, the complete bipartite graph with six vertices $G = (V_1, V_2, E)$, where $E = \{\{v,u\} : v \in V_1, u \in V_2\}$ and $|V_1| = |V_2| = 3$, is also not bipartite since it has $m = 9$ edges, $n = 6$ vertices, and a girth of 4, meaning that $m = 9 > \frac{4}{4-2}(6-2) = 8$. Less obvious, but profoundly more useful, however, is the amazing fact that a graph is planar *if and only if* it does not contain a subgraph that is a subdivision of either of these two examples. This result, due to Kuratowski (1930), has been used alongside similar results to help construct a number of efficient (polynomial-time)

algorithms for determining whether a graph is planar or not, including the Path Addition method of Hopcroft and Tarjan (1974) and the more recent Edge Addition method of Boyer (2004).

5.1.1 Dual Graphs, Colouring Maps, and the Four Colour Theorem

The close relationship between the problems of vertex colouring and face colouring becomes apparent when we consider the concept of *dual graphs*. Given a planar graph G, the *dual* of G, denoted by G^*, is constructed according to the following steps. First, draw a single vertex v_i^* inside each face F_i of G. Second, for each edge e in G, draw a line e^* that crosses e but no other edge in G, and that links the two vertices in G^* corresponding to the two faces in G that e is separating.

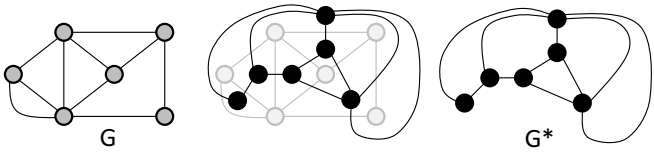

Fig. 5.2 Illustration of how to convert a planar graph G to its dual G^*

This procedure is demonstrated in Figure 5.2. Here, the vertices in G are shown in grey, and the vertices in G^* are shown in black. G has six faces in total, five bounded faces and one unbounded face. The unbounded face is represented by the top vertex of G^* in the example and is made adjacent to all vertices in G^* whose corresponding faces in G have an edge on the exterior of the graph. Note that G^* may also have multiple edges between a pair of vertices, as is occurring on the right-hand side of the example graph.

It is clear from the figure that the process of forming duals is reversible: that is, we can use the same process to form G from G^*. It is also clear that because G is planar, its dual G^* must also be planar. We are also able to state simple relationships between the number of vertices, faces and edges in G and G^* such as the following:

Theorem 5.3 *Let G be a connected planar graph with n vertices, m edges, and f faces. Also, let G^* be the dual of G, comprising n^* vertices, m^* edges, and f^* faces. Then $n^* = f$, $m^* = m$ and $f^* = n$.*

Proof. It is clear that $n^* = f$ due to the method by which duals are constructed. Similarly $m^* = m$ because all edges in G^* intersect exactly one edge each in G (and vice versa). The third relation follows by substituting the previous two relationships into Euler's characteristic applied to both G and G^*. □

Recall from Chapter 1 that the Four Colour Theorem (or "conjecture" as it was at the time) was originally stated in 1852 by Francis Guthrie, who hypothesised that four colours are sufficient for colouring the faces of any map such that neighbouring faces have different colours. In the context of graph theory, a map can be

Fig. 5.3 The territories of mainland Australia (left), and the corresponding planar graph (right)

represented by a bridge-free planar graph G, with the faces of G representing the various regions of the map, edges representing borders between regions, and vertices representing points where the borders intersect. An illustration using a map of Australia (excluding the Australian Capital Territory) is given in Figure 5.3.

The following theorem now reveals the close relationship between the vertex colouring and face colouring problems:

Theorem 5.4 *Let G be a connected planar graph without loops, and let G^* be its dual. Then the vertices of G are k-colourable if and only if the faces of G^* are k-colourable.*

Proof. Since G is connected, planar, and without loops, its dual G^* is a planar graph with no bridges. If we have a k-colouring of the vertices of G, then each face of G^* can now be assigned to the same colour as its corresponding vertex in G. Because no adjacent vertices in G have the same colour, it follows that no adjacent faces in G^* will have the same colour. Thus the faces of G^* are k-colourable.

Now suppose that we have a k-colouring of the faces of G^*. Since every vertex of G is contained in a face of G^*, each vertex in G can assume the colour of its corresponding face in G^*. Again, since no adjacent faces in G^* are allocated the same colour, this implies no adjacent vertices in G are given the same colour. □

This result is important because it tells us that the faces of any map, represented as a planar graph G^* with no bridges, can be k-coloured by simply determining a vertex k-colouring of its dual graph G. The result also tells us that we can take any theorem concerning the vertex colouring of a planar graph and then state a corresponding theorem on the face colouring of its dual, and vice versa. One elegant theorem that arises from this characteristic demonstrates a link between a special type of topology known as Eulerian graphs and graphs that are bipartite.

Definition 5.2 *A graph G is* Eulerian *if and only if the degree of all vertices in G are even.*

This gives rise to the following theorem:

Theorem 5.5 *The faces of a planar graph with no bridges G are 2-colourable if and only if G is Eulerian.*

Proof. Recall that a graph's vertices are 2-colourable if and only if it is bipartite. Hence we need to show that the dual of any planar Eulerian graph is bipartite, and vice versa.

Let G be an Eulerian planar graph. By definition, all vertices in G are even in degree. Since the degree of a vertex in G corresponds to the number of edges surrounding a face in the dual G^*, the edges surrounding each face in G^* constitute cycles of even length. Hence according to Theorem 2.8, G^* is bipartite.

Conversely, let G^* be bipartite. This means G^* contains no odd cycles and, since G is planar, all faces are surrounded by an even number of edges. Hence all vertex degrees in G are even, making G Eulerian. □

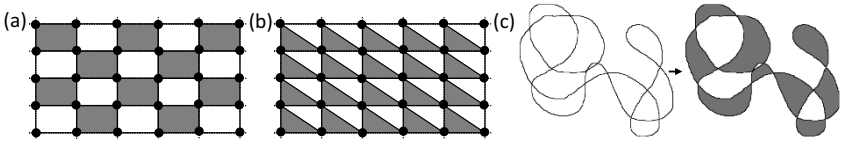

Fig. 5.4 Examples of face colourings using two colours

Practical examples of Theorem 5.5 arise in the tiling industry where we are often interested in laying tiles of two different colours such that adjacent tiles do not have the same colour. Two example patterns are shown in Figure 5.4(a) and (b). Close examination of these patterns reveals the underlying graphs to be Eulerian as expected. Another example arises in the childhood doodling game in which a single connected line is drawn on a piece of paper, with the faces then being coloured using just two colours. Figure 5.4(c) shows an example of this game. We see that each time the line crosses itself, the degree of the vertex existing at this intersection increases by 2; hence vertex degrees are always even as expected.

The connection between face k-colourings of maps and vertex k-colourings of their planar duals allows us to conclude that the Four Colour Theorem for maps is equivalent to the statement that the vertices of all loop-free planar graphs are 4-colourable.[1] This concept was hinted upon in Section 1.2 where, in Figure 1.7, we took a map of Wales, constructed its (planar) dual graph, 4-coloured its vertices, and then converted this solution back into a 4-colouring of the faces of original map. The task of proving that four colours are sufficient for the vertices of *any* planar graph (and therefore the faces of any map) was formally one of the most famous unsolved problems in the whole of mathematics. It was eventually solved in controversial circumstances by Kenneth Appel and Wolfgang Haken in 1976. Their proof is both

[1] Recall that loops (i.e., edges of the form $\{v_i, v_i\}$) are disallowed in the vertex colouring problem.

very long and involved using months of computation time to test and classify a large number of different graph configurations. Consequently, we end this section by restricting ourselves to proving the weaker Six and Five Colour Theorems, before giving a general history of the Four Colour Theorem itself.

Theorem 5.6 *The vertices of any loop-free planar graph are 6-colourable.*

Proof. Let G be a planar graph with $n \geq 3$. According to Theorem 5.2, G has at most $3n-6$ edges. This means that the minimal degree of G cannot exceed 5. Thus in every subgraph G' of G there is a vertex with degree of at most $\delta = 5$. Therefore, according to Theorem 2.6, we get $\chi(G) \leq 5+1$. □

With some additional reasoning we can improve this result to get the following:

Theorem 5.7 (Heawood (1890)) *The vertices of any loop-free planar graph are 5-colourable.*

Proof. For contradictory purposes, suppose this statement to be false, and let G be a planar graph with chromatic number $\chi(G) = 6$ and a minimal number of vertices n. Because of Theorem 5.2, G must have a vertex v with $\deg(v) \leq 5$. Let $G' = G - \{v\}$. We know that G' can be 5-coloured using, say, colours labelled 1 to 5. Each of these colours must also be used to colour at least one neighbour of v; otherwise G would also be 5-colourable. We can now assume that v has five neighbours, say $u_1, u_2, \ldots, u_5$, arranged in a clockwise fashion around v, with colours $c(u_i) = i$.

Now denote by $G'(i,j)$ the subgraph of G' spanned by vertices with colours i and j. Suppose that u_1 and u_3 belong to separate components of $G'(1,3)$. Interchanging the colours 1 and 3 in the component of $G'(1,3)$ containing u_1 will give us another feasible 5-colouring of G'. However, in this 5-colouring, both u_1 and u_3 will have the same colour, meaning that a spare colour now exists for v. This implies that G is in fact 5-colourable.

Since u_1 and u_3 must belong to the same component $G'(1,3)$, we now deduce the existence of a path $P_{1,3}$ in G' whose vertices are coloured using colours 1 and 3 only. Similarly, G' must also contain a path $P_{2,4}$ using colours 2 and 4. However, this is impossible in a planar graph since the cycle $u_1, P_{1,3}, u_3$ separates u_2 from u_4, meaning that $P_{2,4}$ cannot be drawn without edges crossing (see Figure 5.5). Hence G cannot be planar. □

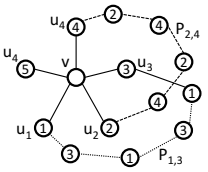

Fig. 5.5 Depicting paths $P_{1,3}$ and $P_{2,4}$ used in the proof of Theorem 5.7. Colour labels are written inside the vertices

5.1.2 Four Colours Suffice

In the proof of Theorem 5.7 we make use of the notation $G(i, j)$, which denotes the subgraph induced by taking only those vertices coloured with colours i and j in G. Note that individual components of $G(i, j)$ are actually Kempe chains (see Definition (3.1)). These are named after the mathematician Alfred Kempe (1849–1922), who used them in his infamous false proof for the Four Colour Theorem in 1879.

As we saw in Chapter 1, the conjecture that all maps can be coloured using at most four colours was first pointed out in 1852 by Francis Guthrie (1831–1899), who, at the time, was a student at University College London. Guthrie passed these observations on to his brother Frederick who, in turn, passed them on to his mathematics tutor Augustus De Morgan (1807–1871). De Morgan was not able to provide a conclusive proof for this conjecture, but the problem, being both easy to state and tantalisingly difficult to solve, captured the interest of many of the most notable mathematicians of the era, including William Hamilton (1805–1865), Arthur Cayley (1821–1895) and Charles Pierce (1839–1914). Indeed, over time the Four Colour Conjecture was to become one of the most famous unsolved problems in all of mathematics.

In 1879 a student of Arthur Cayley, Alfred Kempe, announced in *Nature* magazine that he had proved the Four Colour Theorem, publishing his result in the *American Journal of Mathematics* (Kempe, 1879). In his arguments, Kempe made use of his eponymous Kempe chains in the following way. Suppose we have a map in which all faces except one are coloured using colours 1, 2, 3, or 4. If the uncoloured face, which we shall call F, is not surrounded by faces featuring all four colours, then obviously we can colour F using the missing colour. Therefore, suppose now that F is surrounded by faces F_1, F_2, F_3, and F_4 (in that order), which are coloured using colours 1, 2, 3, and 4 respectively. There are now two cases to consider:

Case 1: There exists no chain of adjacent faces from F_1 to F_3 that are alternately coloured with colours 1 and 3.

Case 2: There *is* a chain of adjacent faces from F_1 to F_3 that are alternately coloured with colours 1 and 3.

If Case 1 holds then F_1 can be switched to colour 3, and any remaining faces in the chain can also have their colours interchanged. This operation retains the feasibility of the solution (no adjacent faces will have the same colour) and also means that no face adjacent to F will have colour 1. Consequently F can be assigned to this colour.

If Case 2 holds then there cannot exist a chain of faces from F_2 to F_4 using only colours 2 and 4. This is because, for such a chain to exist, it would need to cross the chain from F_1 to F_3, which is impossible on a map. Thus, Case 1 holds for F_2 and F_4, allowing us to switch colours as with Case 1.

The arguments of Kempe were widely accepted among mathematicians of the day; he was promptly elected a Fellow of the Royal Society and also went on to be knighted in 1912. The Four Colour *Conjecture* was now the Four Colour *Theorem*.

This all changed 11 years later when, in 1890, English mathematician Percy Heawood (1861–1955) shocked the mathematics fraternity by publishing an example map which exposed a flaw in Kempe's arguments (Heawood, 1890). Though he failed to supply his own proof, Heawood had shown that the Four Colour Theorem was indeed still a conjecture. In the same publication Heawood did show, however, that arguments analogous to Kempe's could be used to prove that all maps are 5-colourable, as we saw in Theorem (5.7). In later work, Heawood also proved that if the number of edges around each region of a map is divisible by 3 then the map can be 4-coloured.

As the decades passed, the problem that had first been pointed out by Guthrie in 1852 still remained unproven. Some piecemeal progress towards a solution was made with one proof showing that four colours were sufficient for colouring maps of up to 27 faces. This was followed by proofs for up to 31 faces, and then 35 faces. However, it would turn out that methods used by Kempe and his contemporaries in early papers would ultimately pave the way forward.

To start, the focus of research turned towards writing proofs concerning the vertices of loop-free planar graphs (i.e., the dual graphs of maps). In the first half of the twentieth century, researchers also concentrated their efforts on reducing these graphs to special cases that could be identified and classified. The idea was to produce a minimal set of configurations that could each be tested. Initially, this set was thought to contain nearly 9,000 members, which was considered far too large for mathematicians to study individually. This compelled some to turn towards using computers in order to design specialised algorithms for testing them.

Ultimately, the first conclusive proof of the Four Colour Theorem was produced in 1976 by mathematicians Kenneth Appel (1932-2013) and Wolfgang Haken (b. 1928), who showed that no configuration can exist that will appear in a minimal counterexample to the Four Colour Theorem (Appel and Haken, 1977a,b,c). In research carried out at the University of Illinois, together they reduced the set of configurations to just 1,936 members, which were then individually checked by computer. As was later stated in Appel's obituary in *The Economist* on May 4th, 2013:

> Both he and Dr Haken hugely exceeded their time allocation on the computer, which belonged to the university administration department. ... Their proof depended on both hand-checking by family members and then brute-force computer power; the result was published in over 140 pages in the *Illinois Journal of Mathematics* and 400 pages of further diagrams on microfiche. They also, in the old fashioned way, chalked the message on a blackboard in the mathematics department: FOUR COLOURS SUFFICE.

At the time, this proof was controversial, with some mathematicians publicly questioning the legitimacy of a proof in which much of the work had been carried out by computer. (How might we guarantee the reliability of the algorithms and hardware?) However, despite these worries independent verification soon convinced the majority that the Four Colour Theorem had indeed finally been proved. Hence we are now able to state:

Theorem 5.8 (The Four Colour Theorem) *The vertices of any loop-free planar graph are 4-colourable. Equivalently, the faces of any map are 4-colourable.*

In recent years, refinements to Haken and Appel's proof have been made, with Robertson et al. (1997) showing that, through various acts of trickery, only 633 configurations need to be considered. (A short summary of this proof, together with a description of a polynomial-time algorithm for 4-colouring planar graphs, can be found at http://people.math.gatech.edu/∼thomas/FC/fourcolor.html.) However, a proof along more "traditional" lines still remains elusive and, to this day, the Four Colour Theorem remains an excellent example, along with Fermat's Last Theorem, of a problem that is very easy to state, but exceptionally difficult to prove.

Readers interested in finding out more about the fascinating history of the Four Colour Theorem are invited to consult the very accessible book *Four Colors Suffice: How the Map Problem Was Solved* by Wilson (2003).

5.2 Edge Colouring

Another way in which graphs might be coloured is by assigning colours to their *edges*, as opposed to their vertices or faces. This gives rise to the *edge colouring problem* where we seek to colour all edges of a graph so that no pair of edges sharing a common vertex (incident edges) have the same colour, and so that the number of colours used is minimal. The edge colouring problem has applications in scheduling round robin tournaments and also transferring files in computer networks (de Werra, 1988; Coffman et al., 1985). The minimum number of colours needed to edge colour a graph G is called the *chromatic index*, denoted by $\chi'(G)$. This should not be confused with the *chromatic number* $\chi(G)$, which is the minimum number of colours needed to *vertex* colour a graph G.

As mentioned earlier, the edge colouring and vertex colouring problems are very closely related because we are able to edge colour any graph by simply vertex colouring its corresponding *line graph*.

Definition 5.3 *Given a graph G, the* line graph *of G, denoted by $L(G)$, is constructed by using each edge in G as a vertex in $L(G)$, and then connecting pairs of vertices in $L(G)$ if and only if the corresponding edges in G share a common vertex as an endpoint.*

An example conversion between a graph G and its line graph $L(G)$ is shown in Figure 5.6(a). From this process, it is natural that the number of vertices and edges in $L(G)$ is related to the number of vertices and edges in G.

Theorem 5.9 *Let $G = (V,E)$ be a graph with n vertices and m edges. Then its line graph $L(G)$ has m vertices and*

$$\frac{1}{2}\sum_{v\in V}\deg(v)^2 - m$$

edges.

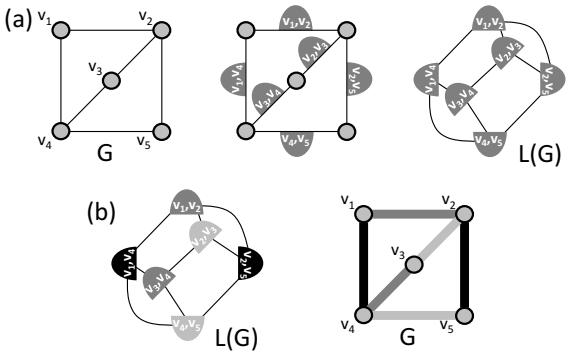

Fig. 5.6 Illustration of (a) how to convert a graph G into its line graph $L(G)$, and (b) how a vertex k-colouring of $L(G)$ corresponds to an edge k-colouring of G

Proof. Since each edge in G corresponds to a vertex in $L(G)$ it is obvious that $L(G)$ has m vertices. Now let $\{u,v\}$ be an edge in G. This means that $\{u,v\}$ is a vertex in $L(G)$ with degree $\deg(u)+\deg(v)-2$. Hence the total number of edges in $L(G)$ is:

$$\frac{1}{2}\sum_{\{u,v\}\in E}(\deg(u)+\deg(v)-2)=\frac{1}{2}\sum_{\{u,v\}\in E}(\deg(u)+\deg(v))-m$$

Note that the degree of each vertex v appears exactly $\deg(v)$ times in this sum. Hence, we can simplify the expression to that stated in Theorem 5.9 as required.

□

Figure 5.6(b) also demonstrates the way in which a vertex k-colouring of the line graph $L(G)$ corresponds to an edge colouring of G. Consequently, rather like the way in which a face colouring problem can be solved by vertex colouring a graph's dual, any edge colouring problem stated on a graph G can be tackled by applying a vertex colouring algorithm to its line graph $L(G)$.

We now discuss some important results concerning the chromatic index of a graph.

Theorem 5.10 *Let K_n be the complete graph with $n>1$ vertices. Then $\chi'(K_n)=n-1$ if n is even; else $\chi'(K_n)=n$.*

Proof. When n is odd, the edges of K_n can be coloured using n colours by the following process. First, draw the vertices of K_n in the form of a regular n-sided polygon. Next, select an arbitrary edge on the boundary of this polygon and colour it, together with all edges parallel to it, using colour 1. Now moving in a clockwise direction, select the next edge on the boundary and colour it, together with its parallel edges, with colour 2. Continue this process until all edges have been coloured.

It is easy to demonstrate that the edges of K_n are not $(n-1)$-colourable by the fact that the largest number of edges that can be assigned the same colour is $(n-1)/2$;

it then follows, because the number of edges in K_n is $\frac{n(n-1)}{2}$, that n colours are required.

When n is even, a similar process can be followed, where a regular $(n-1)$-sided polygon is constructed, with the remaining vertex being placed in the centre. The same method for the $(n-1)$ case is then followed, with edges perpendicular to the edges currently being coloured also being assigned to the same colour. As in the previous case it is easily shown that no feasible edge colouring of K_n exists using fewer than n colours. □

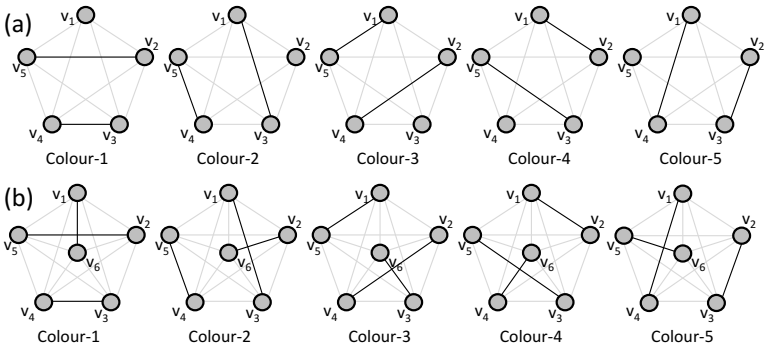

Fig. 5.7 Illustrating how optimal edge colourings can be constructed for complete graphs with (a) K_5 and (b) K_6 using the circle method

The method used in the proof of Theorem 5.10 is often referred to as the circle (or polygon) method and was originally proposed by mathematician and Church of England minister Thomas Kirkman (1806-1895) (Kirkman, 1847). An important practical use of this method is for constructing round-robin sports leagues, where we have a set of n teams that are required to play each other once across a sequence of rounds. Figure 5.7 provides examples of this method for $n = 5$ and $n = 6$. Here, the vertices can be thought of as "teams", with edges representing "matches" between these teams. Each colour then represents a round in the schedule. Considering Figure 5.7(a), where $n = 5$, the first round involves matches between team-v_2 and team-v_5 and between team-v_3 and team-v_4, with team-v_1 receiving a bye. The next round then involves matches between team-v_1 and team-v_3 and between team-v_4 and team-v_5, with team-v_2 receiving a bye, and so on. The pattern is similar when n is even, as shown in Figure 5.7(b), except that no team receives a bye. Applications of graph colouring to sports scheduling problems are considered in more detail in Chapter 7.

A further result, due to König (1916), concerns the chromatic index of bipartite graphs.

Theorem 5.11 (Konig's Line Colouring Theorem) *Let $G = (V_1, V_2, E)$ be a bipartite graph with maximal degree $\Delta(G)$. Then $\chi'(G) = \Delta(G)$.*

Proof. The proof is via induction on the number of edges m in G. It is sufficient to prove that if $m-1$ edges have been coloured using at most $\Delta(G)$ colours, then the remaining edge can be coloured using one of the $\Delta(G)$ available colours.

Suppose all edges except $\{u,v\}$ have been coloured. Then there exists at least one colour not incident to u, and one colour not incident to v. If the same colour is not incident to both u and v, then edge $\{u,v\}$ can be assigned to this colour. If this is not the case, without loss of generality we can say that: (a) $u \in V_1$ is incident to a grey edge, but not a black edge; and (b) $v \in V_2$ is incident to a black edge but not a grey edge.

Now consider a grey-black Kempe chain starting from u (that is, a component of G containing u that comprises only those vertices incident to a grey or black edge). Travelling along this chain from u involves alternating between vertex sets V_1 and V_2. However, we will never reach v because each time we arrive at a vertex in V_2 we do so via a grey edge, which v is not incident to.

Since v is not in the Kempe chain, we can now interchange the colours of the edges in this chain without affecting v (or indeed any other vertices outside of the chain). Hence the edge $\{u,v\}$ can be coloured grey, completing the edge colouring. □

The previous two theorems demonstrate that the edge colouring problem is solvable in polynomial time for both complete and bipartite graphs. We have also seen that, for both topologies, their chromatic indices $\chi'(G)$ are either $\Delta(G)$ or $\Delta(G)+1$. Somewhat surprisingly, it also turns that this feature applies to *any* graph G, as proved by Vizing (1964).

Theorem 5.12 (Vizing's Theorem) *Let G be a simple graph with maximal degree $\Delta(G)$; then $\Delta(G) \leq \chi'(G) \leq \Delta(G)+1$.*

Proof. When $\Delta(G)$ edges are incident to a vertex, these edges all require a different colour. Hence the lower bound is proved: $\Delta(G) \leq \chi'(G)$.

The upper bound can be proved via induction on the number of edges. Suppose that, using $\Delta(G)+1$ colours, we have coloured all edges in G except for the single edge $\{u,v_0\}$. Since $\Delta(G)$ gives the maximal degree, at least one colour will be unused at each of these two vertices. Now construct a series of edges, $\{u,v_0\},\{u,v_1\},\ldots$, and a sequence of colours, $c_0,c_1,\ldots$, as follows: Select a colour c_i that is an unused colour at v_i. Now, let $\{u,v_{i+1}\}$ be an edge with colour c_i. We stop (with $i=k$) when either c_k is an unused colour at u, or c_k is already used on an edge $\{u,v_{j<k}\}$.

Case 1: If c_k is an unused colour at u, then we can recolour $\{u,v_i\}$ with c_i for $0 \leq i \leq k$. We now need to simply recolour edges incident to u to complete the proof.

Case 2: Otherwise, we recolour $\{u,v_i\}$ with c_i for $0 \leq i < j$ and remove the colour from $\{u,v_j\}$. Observe that c_k (say, "grey") is missing at both v_j and v_k. Now let "black" be an used colour at u. If grey is unused at u then we can colour $\{u,v_j\}$ grey. If black is unused at v_j then we can colour $\{u,v_j\}$ black. However, if black is unused at v_k then we colour $\{u,v_i\}$ with c_i for $j \leq i < k$ and colour $\{u,v_k\}$

black, because none of the edges $\{u, v_i\}$ for $j \leq i < k$ will be coloured grey or black.

If neither case above holds, then we consider the subgraph of grey and black edges. The components of this subgraph will be paths and/or cycles. The vertices u, v_i, and v_k are the terminal vertices of paths; hence they cannot all belong to the same component. In this case, select a component containing just one of these vertices and interchange the colours of its edges. This means that one of the cases above now applies. □

In essence, Vizing's theorem tells us that the set of all graphs can be partitioned into two classes: "class one" graphs, for which $\chi'(G) = \Delta(G)$, and "class two" graphs, where $\chi'(G) = \Delta(G) + 1$. Holyer (1981) has shown that the problem of testing whether a graph belongs to class one is NP-complete. However, a number of polynomially bounded algorithms are available for colouring the edges of any graph using exactly $\Delta(G) + 1$ colours, such as the $\mathcal{O}(nm)$ algorithm of Misra and Gries (1992). The existence of such algorithms tells us that we can colour the edges of any graph using a maximum of one extra colour beyond its chromatic index.

We might now ask whether the existence of such tight bounds for the edge colouring problem helps us to garner further information about the vertex colouring problem. It is clear that if we were given the task of vertex colouring a line graph $L(G)$, one approach would be to convert $L(G)$ into its "original" graph G, and then solve the corresponding edge colouring problem on G. Since $\chi(L(G)) = \chi'(G)$, then according to Vizing's theorem this would immediately tell us that we need to use either $\Delta(G)$ or $\Delta(G)+1$ colours to feasibly colour the vertices of $L(G)$. Indeed, if G were a type two graph, then algorithms such as Mistra and Gries's could be used to quickly find the optimal edge colouring for G and therefore the optimal vertex colouring for $L(G)$. However, it should be remembered that this very attractive sounding proposal is only applicable when we wish to colour the vertices of a line graph that therefore has an "original" graph into which it can be converted. Unfortunately we cannot convert all graphs into an "original" graph in this way.

5.3 Precolouring

In the precolouring problem we are given a graph G for which some subset of the vertices $V' \subseteq V$ has already been assigned to colours. Our task is to then colour the remaining vertices in the set $V - V'$ so that the resultant solution is feasible and uses a minimal number of colours.

Applications of precolouring arise in register allocation problems (see Section 1.1.4) where certain variables must be assigned to specific registers, perhaps due to calling conventions or communication between modules. They also occur in areas such as timetabling and sports scheduling where we might be given a problem instance in which some of the events have been preassigned to particular timeslots.

Precolouring problems can be easily converted into a standard graph colouring problem using graph contraction operations.

Definition 5.4 *The* contraction *of a pair of vertices $v_i, v_j \in V$ in a graph G produces a new graph in which v_i and v_j are removed and replaced by a single vertex v such that v is adjacent to the union of the neighbourhoods of v_i and v_j; that is, $\Gamma(v) = \Gamma(v_i) \cup \Gamma(v_j)$.*

The following steps can now be taken. Given a precolouring problem instance defined on a graph G, let $V'(i)$ define the set of vertices precoloured with colour i. Assuming there are k different colours used in the precolouring, $\bigcup_{i=1}^{k} V'(i) = V'$ and $V'(i) \cap V'(j) = \emptyset$, for $1 \leq i \neq j \leq k$. First, for each set $V'(i)$, merge all vertices into a single vertex using a series of contraction operations. This has the effect of reducing the number of precoloured vertices to k. Next, add edges between each pair of the k contracted vertices to form a clique. Finally remove all colours from the vertices of this graph, and apply any arbitrary graph colouring algorithm to produce a feasible solution. A colouring of the original can then be obtained by simply reversing the above process. An example is provided in Figure 5.8.

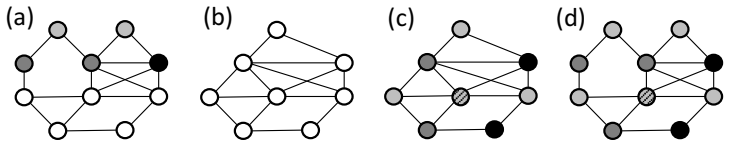

Fig. 5.8 How a precolouring problem (a), can be converted into a new graph via contraction operations on the precoloured vertices (b), then coloured (c), and then converted back into a solution to the original problem (d)

5.4 Latin Squares and Sudoku Puzzles

Another prominent area of mathematics for which graph colouring techniques are naturally suited is the field of Latin squares. Latin squares are $l \times l$ grids that are filled with l different symbols, each occurring exactly once per row and once per column. They were originally considered in detail by Leonhard Euler, who filled his grids with symbols from the Latin alphabet, though nowadays it is common to use the integers 1 through to l to fill the grids. Example Latin squares of different sizes are shown in Figure 5.9.

Latin squares have practical applications in various different areas, including scheduling and experimental design. For an application in scheduling, imagine that we have two groups of l people and we want to schedule meetings between all pairs of people belonging to different groups. Clearly in this case, l^2 meetings will take place in total and, since only l meetings can take place simultaneously, at least l

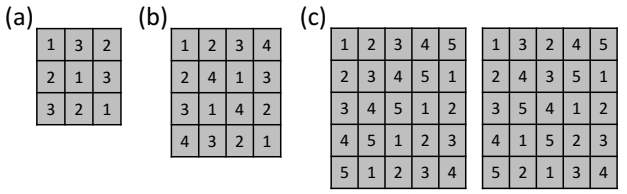

Fig. 5.9 Example Latin squares for (a) $l = 3$, (b) $l = 4$, and (c) $l = 5$

timeslots will be needed. Latin squares give solutions to such problems that make use of exactly l timeslots. To see this, let us name the members of team one as $r_1, r_2, \ldots, r_l$, which are represented by the rows in the grid, and the members of team two $c_1, c_2, \ldots, c_l$, represented by the columns. The characters within an $l \times l$ Latin square then represent the various timeslots to which the meetings are assigned. For example, the Latin square shown in Figure 5.9(a) schedules meetings between r_1 and c_1, r_2 and c_2, and r_3 and c_3 into timeslot 1; meetings between r_1 and c_3, r_2 and c_1, and r_3 and c_2 into timeslot 2; and meetings between r_1 and c_2, r_2 and c_3, and r_3 and c_1 into timeslot 3. Obviously, any $l \times l$ Latin square will provide a suitable meeting schedule fitting these criteria.

For an example application of Latin squares in experimental design, imagine that a medical trials centre wants to test the effects of l different drugs on a particular illness. Suppose further that the trials are to take place over l weeks using l different patients, with each patient receiving a single drug in each week. An $l \times l$ Latin square can be used to allocate treatments in this case, with rows representing patients, and columns representing weeks. This means that over the course of the l weeks each patient receives each of the l drugs once, and in each week all of the l drugs are tested. Looking at the 3×3 Latin square from Figure 5.9(a), for example, we see that Patients 1, 2, and 3 are administered Drugs 1, 2, and 3 (respectively) in Week 1; Drugs 3, 1, and 2 in Week 2; and Drugs 2, 3, and 1 in Week 3, as required.

Note that we are able to permute the rows and columns of a Latin square and still retain the property of each character occurring exactly one per column and once per row. It is therefore common to write Latin squares in their *standardised form*, whereby the rows and columns are arranged so that the top row and leftmost column of the grid have the characters in their natural order $1, 2, \ldots, l$. The other $l!(l-1)! - 1$ Latin squares that can be formed by permuting the rows and columns are then considered to be equivalent to this. The Latin square in Figure 5.9(b) is in standardised form, while the one in Figure 5.9(a) is not. It is also known that as l is increased, then so does the number of different $l \times l$ Latin squares. For $l = 11$ this figure is approximately 5.36×10^{33}, though for larger values of l these figures have so far proved too large to compute.

Figure 5.10 shows how the production of a Latin square can be expressed as a graph colouring problem. Here, as illustrated, the symbols used within the grid represent the colours. Each cell of the grid is then associated with a vertex, and edges are added between all pairs of vertices in the same row, and all pairs of vertices in the same column. This results in a graph $G = (V, E)$ with $n = l^2$ vertices and

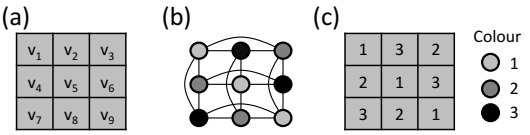

Fig. 5.10 Showing the relationship between graph colouring and Latin squares. Part (a) associates each grid cell with a vertex; (b) shows the corresponding graph together with a feasible colouring; and (c) gives a valid Latin square corresponding to this colouring

$m = l^2(l-1)$ edges, for which $\deg(v) = 2(l-1)\ \forall v \in V$. We also see that the set of vertices in each row forms a clique of size l, as do vertices in each column, implying that solutions using fewer than l colours are not possible.

Note that it is simple to produce a Latin square in standardised form for any value of l by simply using values $(1,2,\ldots,l)$ for the first row, $(2,3,\ldots,l,1)$ for the second row, $(3,4,\ldots,l,1,2)$ for the third row, and so on (see for example the left Latin square in Figure 5.9(c)). Hence graphs representing Latin squares are actually a particular topology for which the associated graph colouring problem can be easily solved in polynomial time for any value of l, without a need for resorting to heuristics or approximation algorithms. Graph colouring algorithms can, however, be used for producing different Latin squares to this.

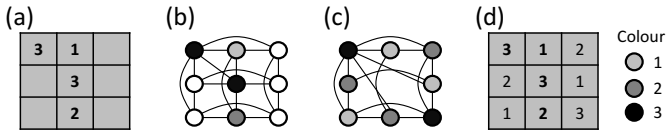

Fig. 5.11 A partial 3×3 Latin square with four filled-in cells (a); the corresponding precolouring problem (b); the same graph with a contraction of two vertices, together with a feasible colouring (c); and the corresponding Latin square solution (d)

Graph colouring algorithms arguably become more useful in this area when we consider the *partial* Latin square problem. This is the problem of taking a partially filled $l \times l$ grid and deciding whether or not it can be completed to form a Latin square. This problem has been shown to be NP-complete by Colbourn (1984).

Figure 5.11 gives an example of how the partial Latin square problem can be tackled using graph colouring principles. In essence it follows the same method as the previous example given in Figure 5.10, except that extra edges can now also be added between any pair of vertices predefined as having the same colour. Once this has been done, the same steps as with the precolouring problem (Section 5.3) can be followed, with contractions being used to make the graph smaller if desired. An l-colouring of this graph then corresponds to a completed $l \times l$ Latin square. Of course, depending on the values of the filled-in cells in the original problem, there could be zero, one, or multiple feasible l-colourings available.

5.4.1 Solving Sudoku Puzzles

The partial Latin square problem has become very popular in recent decades in the form of Sudoku puzzles. In Sudoku we are given a partially filled Latin square and the objective is to complete the remaining cells so that each column and row contains the characters $1,\ldots,l$ exactly once. In addition, Sudoku grids are also divided into l "boxes" (usually marked by bold lines) which are also required to contain the characters $1,\ldots,l$ exactly once; thus Sudoku can be considered a special case of the partial Latin square problem in which the constraint of appropriately filling out the "boxes" must also be satisfied. An example 9×9 Sudoku puzzle and corresponding solution is shown in Figure 5.12.

	2	4			7			
6								
		3	6	8		4	1	5
4	3	1			5			
5							3	2
7	9						6	
2		9	7	1		8		
	4			9	3			
3	1				4	7	5	

1	2	4	9	5	7	3	8	6
6	8	5	3	4	1	2	9	7
7	9	3	6	8	2	4	1	5
4	3	1	2	6	5	9	7	8
5	6	8	4	7	9	1	3	2
7	9	2	1	3	8	5	6	4
2	5	9	7	1	6	8	4	3
8	4	7	5	9	3	6	2	1
3	1	6	8	2	4	7	5	9

Fig. 5.12 A 9×9 Sudoku puzzle and corresponding solution

Because Sudoku is intended to be an enjoyable puzzle, problems posed in books and newspapers will nearly always be *logic solvable*.

Definition 5.5 *A Sudoku puzzle is* logic solvable *if and only if it features exactly one solution, which is achievable via forward chaining logic only.*

Puzzles that are not logic solvable require random choices to be made. In general these should be avoided because players will have to go through the tedious process of backtracking and reguessing if their original guesses turn out to be wrong.

As an example of how a player might deduce the contents of cells, consider the puzzle given in Figure 5.12. Here, we see that the cell in the seventh row and sixth column (shaded) must contain a 6, because all numbers 1 to 5 and 7 to 9 appear either in the same column, the same row, or the same box as this cell. If the problem instance is logic solvable (as indeed this one is), the filling in of this cell will present further clues, allowing the user to eventually complete the puzzle.

A number of algorithms for solving Sudoku puzzles are available online, such as those at www.sudokuwiki.org and www.sudoku-solutions.com. Such algorithms typically mimic the logical processes that a human might follow, with popular deductive techniques, such as the so-called X-wing and Swordfish rules, also being commonplace. In other areas of Sudoku research, Russell and Jarvis (2005) have shown that the number of essentially different Sudoku solutions (when symmetries such as rotation, reflection, permutation and relabelling are taken into account) is 5,472,730,538 for the popular 9×9 grids. McGuire et al. (2012) have also shown

that 9×9 Sudoku puzzles must contain at least 17 filled-in cells to be logic solvable (thus 9×9 puzzles with 16 or fewer filled-in cells will always admit more than one solution). Similar results for larger grids are unknown, however. Herzberg and Murty (2007) have also shown that at least $l - 1$ of the l characters must be present in the filled cells of a Sudoku puzzle for it to be logic solvable.

Despite the fact that Sudoku is a special case of the partial Latin squares problem, Yato and Seta (2003) have demonstrated that the problem of deciding whether or not a Sudoku puzzle features a valid solution is still NP-complete. Graph colouring algorithms can therefore be useful for solving instances of Sudoku, particularly those that are not necessarily logic solvable.

Sudoku puzzles can be transformed into a corresponding graph colouring problem in the same fashion as partial Latin square problems (see Figure 5.11), with additional edges also being imposed to enforce the extra constraint concerning the "boxes" of the grid. We now present two sets of experiments that illustrate the capabilities of the HEA and backtracking algorithms from Chapter 4 for solving Sudoku puzzles. In the first set of experiments we focus on Sudoku problems that are not necessarily logic solvable (random puzzles), while in the second set we focus on 9×9 grids that are logic solvable.

5.4.1.1 Solving Random Sudoku Puzzles

To generate problem instances that are not necessarily logic solvable, we can start by taking a completed Sudoku solution of a given size. Such solutions can be obtained from a variety of places such as the solution pages of a Sudoku book or newspaper, or by simply executing a suitable graph colouring algorithm on a graph representing a blank puzzle. In the next step of the procedure, this completed grid can then be randomly shuffled using the following five operators:

- Transpose the grid;
- Permute columns of boxes within the grid;
- Permute rows of boxes within the grid;
- Permute columns of cells within columns of boxes; and
- Permute rows of cells within rows of boxes.

Note that all of these shuffle operators preserve the validity of a Sudoku solution. Finally, a number of cells in the grid are made blank by going through each cell in turn and deleting its contents with probability $1 - p$, where p is a parameter to be defined by the user. This means that instances generated with a low value for p have a lower proportion of filled-in cells.

Figure 5.13 illustrates the performance of the HEA and backtracking algorithms on 9×9, 16×16 and 25×25 Sudoku grids respectively. In each case 100 instances for each value of p have been generated and, as in Chapter 4, a computation limit of 5×10^{11} constraint checks is imposed. For each algorithm two statistics are displayed. The *success rate* (SR) indicates the percentage of runs for which the algorithms have found a valid Sudoku solution (a feasible l-colouring) within the

computation limit. The *solution time* then indicates the mean number of constraint checks that it took to achieve these solutions. Note that only successful runs are considered in the latter statistic.

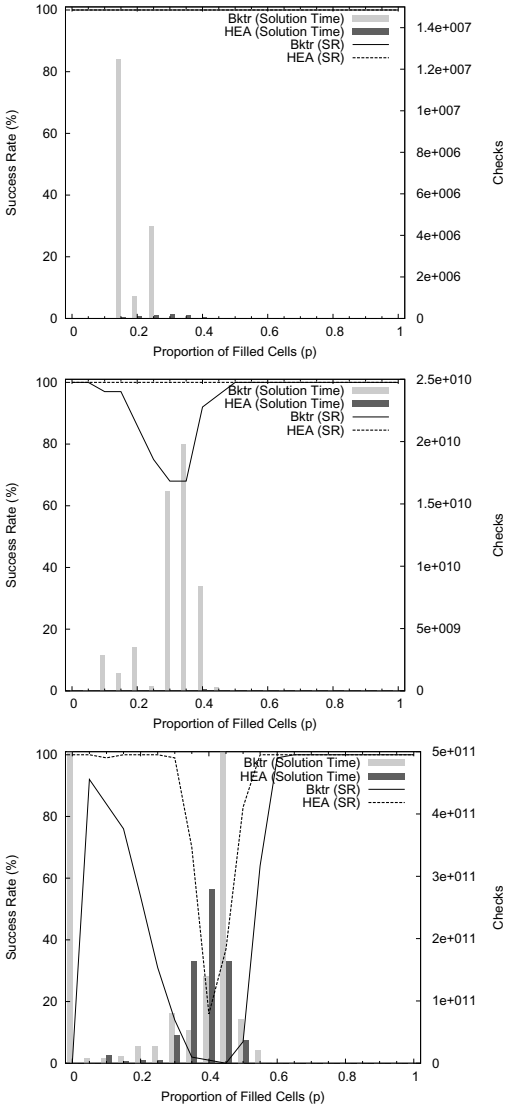

Fig. 5.13 Comparison of the HEA and backtracking algorithm's performance with random Sudoku instances of size 9×9, 16×16, and 25×25 respectively. Note the different scales on the vertical axes in each case

Looking at the results for 9×9 Sudoku puzzles first, we see that both algorithms feature a 100% success rate across all instances with only a very small proportion of the computation limit being required.[2] For 16×16 puzzles, similar patterns occur for the HEA, with all problem instances being solved, and no runs requiring more than one second of computation time. On the other hand, the backtracking algorithm features a dip in its success rate for values of p between 0.1 to 0.55, with a corresponding increase in solution times. Finally, with the larger 25×25 puzzles, this pattern becomes even more apparent, with both algorithms featuring dips in their success rates and subsequent increases in their solution times. However, these dips are less pronounced with the HEA, indicating its superior performance overall.

The dips in the success rates of these algorithms are analogous to the phase transition regions we saw with the flat graphs in Section 4.2. When p is low, although solution spaces will be larger, there will also tend to be many valid solutions within these spaces. Consequently, an effective algorithm should be able to find one of these within a reasonable amount of computation time, as is the case with the HEA. For high vales for p meanwhile, although there will only be a very small number of valid solutions (and perhaps only one), the solution space will be much smaller. Additionally, solutions to these highly constrained instances will also tend to reside at prominent optima (with a strong basin of attraction), thus also allowing easy discovery by an effective algorithm. However, instances at the boundary of these two extremes will cause greater difficulties. First, the solution spaces for these instances will still be relatively large, but they will also tend to admit only a small number of valid solutions. Second, because of their moderate number of constraints, the cost landscapes will also tend to feature more plateaus and local optima, making navigation towards a global optimum more difficult for the algorithm.

5.4.1.2 Logic Solvable Sudoku Puzzles

We now examine the performance of the HEA and backtracking algorithms on logic solvable instances, allowing us to examine the effect that the size of the solution space has on the difficulty of a puzzle when its solution is known to be unique. As we have seen, it is known that a 9×9 Sudoku puzzle must contain at least 17 filled-in cells to be logic solvable (McGuire et al., 2012). There is also an online resource containing over 49,000 different instances of these 17-clue puzzles maintained by Gordon Royle, available at:

`staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php`

For our tests we took a random sample of 100 of these 17-clue instances together with their corresponding (unique) solutions. Logic solvable puzzles with more than 17 filled-in cells were then also generated for each of these by randomly selecting an appropriate number of blank cells in the puzzles, and adding the corresponding

[2] On our equipment (3.0 GHz Windows 7 PC with 3.87 GB RAM) the longest run in the entire set took just 0.02 seconds.

entries from their solutions. This operation maintains the uniqueness of each puzzle's solution while reducing the size of its solution space. All other experimental details remain the same as in the previous subsection.

Figure 5.14 shows the relative performance of the two graph colouring algorithms. In all cases, valid solutions were found within the computation limit. When the solution space size is relatively large (17-20 filled-in cells) we see that the HEA requires up to 2.4×10^{10} constraint checks to find a solution.[3] However, beyond this point the puzzles are solved using very little computational effort—indeed, for more than 35 filled-in cells, solutions are achieved by the initial solutions produced by the DSATUR algorithm.

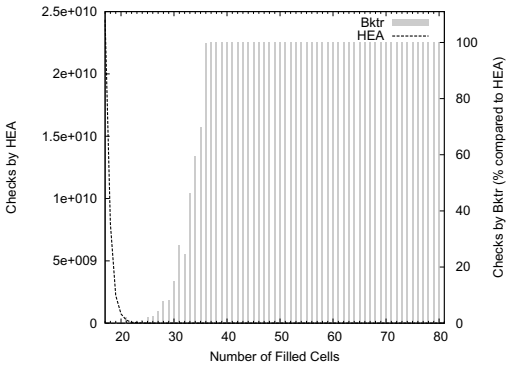

Fig. 5.14 Comparison of the HEA and backtracking algorithm's performance on 9×9 Sudoku grids with unique solutions

In contrast to our earlier results on random Sudoku puzzles, Figure 5.14 also shows that the backtracking algorithm outperforms the HEA with these problem instances. With 17-clue puzzles for example, the algorithm has identified the unique solutions using just 0.03% of the computational effort required by the HEA. Thus, unlike in the larger puzzles seen in the previous section, here the solution spaces seem suitably sized and structured for the backtracking algorithm to be able to identify the unique Sudoku solutions in very short spaces of time.

5.5 Short Circuit Testing

Another interesting practical application of graph colouring is due to Garey et al. (1976), who suggest its use in the process of testing for (undesired) short circuits in printed circuit boards. In their model, a circuit board is represented by a finite lattice of evenly spaced points on to which a set of n cycle-free components has

[3] This equated to approximately three minutes on our computer.

been printed. This set $\mathcal{P}$ is referred to as a *net pattern*, with individual components $P \in \mathcal{P}$ being called *nets*. Each net connects a number of points that are intended to be electrically common. An example net pattern comprising four components is shown in Figure 5.15(a). Note that connections between points are only permitted in vertical or horizontal directions.

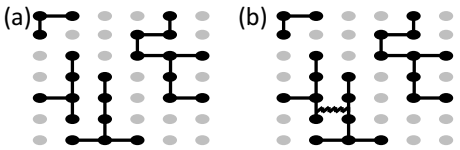

Fig. 5.15 An example net pattern (a), and a net pattern containing a short between two nets (b)

Given a net pattern $\mathcal{P}$, the problem of interest is to determine whether there exists some fault on the circuit board (due to the manufacturing process) whereby an extra conductor path has been introduced between two nets that are not intended to be electrically common. This is the case in Figure 5.15(b). These extra conductor paths are known as "shorts".

An obvious strategy to determine whether a short has occurred is to test each pair of nets $P_i, P_j \in \mathcal{P}$ in turn by applying an electrical current to P_i and seeing if this current spreads to P_j. However, Garey et al. (1976) suggest that the number of pairwise tests can be reduced significantly by making use of the following two observations.

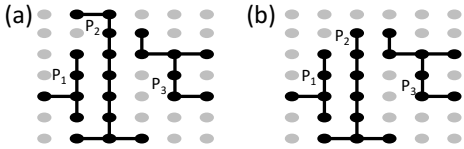

Fig. 5.16 Two further net patterns

First, it is noted that many pairs do not need to be tested. Consider, for example, the net pattern in Figure 5.16(a). Here it is unnecessary to test the pair P_1, P_3 because if there is a short between them, then shorts must also exist between pairs P_1, P_2 and P_2, P_3. Since the objective of the problem is to determine if *any* shorts exist, testing either P_1, P_2 or P_2, P_3 is therefore sufficient. Furthermore, if we go on to consider the net pattern in Figure 5.16(b), it might also be reasonable to assume that shorts cannot occur between P_1 and P_3 without also causing a short involving P_2. Thus, depending on the criteria used for deciding where and how shorts can occur, we have the opportunity to exclude many pairs of nets from the testing procedure. If it is deemed necessary to test a pair of nets, these are called *critical pairs*; otherwise they are deemed *noncritical*.

The second observation is as follows. Let $G = (V,E)$ be a graph with a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, where each vertex $v_i \in V$ corresponds to a particular net $P_i \in \mathcal{P}$ (for $1 \leq i \leq n$). Also, let each edge $\{v_i, v_j\} \in E$ correspond to a pair of nets P_i, P_j judged to be critical. Now let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a partition of V such that no pair of vertices v_i, v_j in any subset S_l forms a critical pair. From a graph colouring perspective, $\mathcal{S}$ therefore defines a feasible k-colouring of the vertices of G. Now suppose that for the printed circuit board in question, external conductor paths are provided so that all nets in any subset S_l can be made electrically common during testing. There are thus k "supernets" that need to be tested. It can be seen that the printed circuit board contains no short if and only if no pair of "supernets" is seen to be electrically common. Therefore, we only have to perform a maximum of $\binom{k}{2}$ tests as opposed to our original figure of $\binom{n}{2}$ tests. Naturally, it is desirable to reduce k as far as possible in order to minimise the number of tests needed.

In their paper, Garey et al. (1976) propose a number of criteria for deciding whether a pair of nets should be deemed critical, with associated theorems then being presented. We now review some of these.

Theorem 5.13 (Garey et al. (1976)) *Consider a pair of nets $P_i, P_j \in \mathcal{P}$ to be critical if and only if a straight vertical line of sight can be drawn that connects P_i and P_j. Then the corresponding graph $G = (V,E)$ is planar and has a chromatic number $\chi(G) \leq 4$.*

Proof. Given a net pattern $\mathcal{P}$, for each pair of nets for which a vertical line of sight exists, draw such a line. Since each line is vertical, none can intersect. It is now possible to contract each net into a single point, deforming the lines of sight (which may no longer be straight lines) in such a way that they remain nonintersecting. This structure now corresponds to the graph $G = (V,E)$, with each vertex corresponding to a contracted net, and each edge corresponding to the lines of sight. Since G is planar, $\chi(G) \leq 4$ according to the Four Colour Theorem (Theorem 5.8). □

Theorem 5.14 (Garey et al. (1976)) *Consider a pair of nets $P_i, P_j \in \mathcal{P}$ to be critical if and only if a straight vertical line of sight or a straight horizontal line of sight can be drawn that connects P_i and P_j. Then the corresponding graph $G = (V,E)$ has a chromatic number $\chi(G) \leq 12$.*

Proof. It is first necessary to show that any graph G formed in this way has a vertex v with $\deg(v) \leq 11$. Let $G_1 = (V,E_1)$ and $G_2 = (V,E_2)$ be subgraphs of G such that E_1 is the set of edges formed from vertical lines and E_2 is the set of edges formed from horizontal lines. Hence $E = E_1 \cup E_2$. By Theorem (5.13), both G_1 and G_2 are planar. We can also assume without loss of generality that the number of vertices $n > 12$. According to Theorem (5.2), the number of edges in a planar graph with n vertices is less than or equal to $3n-6$. Thus:

$$m \leq |E_1| + |E_2| \leq (3n-6) + (3n-6) = 6n-12.$$

Since each edge contributes to the degree of two distinct vertices, this gives:

$$\sum_{v \in V} \deg(v) = 2m \leq 12n - 24.$$

Hence it follows that some vertex in G must have a degree of 11 or less.

Now consider any subset $V' \subseteq V$ with $|V'| > 12$. The induced subgraph of V' must contain a vertex with a degree at most 11. Consequently, according to Theorem (2.6), $\chi(G) \leq 11 + 1$. □

In their paper, Garey et al. (1976) conjecture that the result of Theorem (5.14) might actually be improved to $\chi(G) \leq 8$ because, in their experiments, they were not able to produce graphs featuring chromatic numbers higher than this. They also go on to consider the maximum length of lines of sight and show that:

- If lines of sight can be both horizontal and vertical but are limited to a maximum length of 1 (where one unit of length corresponds to the distance between a pair of vertically adjacent points or a pair of horizontally adjacent points on the circuit board), then G will be planar, giving $\chi(G) \leq 4$.
- If lines of sight can be both horizontal and vertical but are limited to a maximum length of 2, then G will have a chromatic number $\chi(G) \leq 8$.

Finally, they also note that if arbitrarily long lines of sight travelling in *any* direction are permitted (as opposed to merely horizontal or vertical) then it is possible to form all sorts of different graphs, including complete graphs. Hence arbitrarily high chromatic numbers can occur.

5.6 Graph Colouring with Incomplete Information

In this section we now consider graph colouring problems for which information about a graph is incomplete at the beginning of execution. In the following subsections we discuss three different interpretations, specifically decentralised graph colouring, online graph colouring, and dynamic graph colouring, and give practical examples of each.

5.6.1 Decentralised Graph Colouring

In decentralised graph colouring it is generally assumed that each vertex of a graph is an independent entity responsible for choosing its own colour. Moreover, the only information available to each vertex is who its neighbours are, and what their colours are. In other words, vertices are unaware of the structure of the graph beyond their own neighbouring vertices.

A practical example of this problem might occur in the setting-up of a wireless ad hoc network. Imagine a situation where a network is to be created by randomly dropping a set of wireless devices (equipped with radio transmitters and receivers)

into a particular environment. Imagine further that each device in this network will broadcast information at a particular frequency on the radio spectrum. Finally, also consider the fact that if two devices are close together but using the same frequency (or frequencies that are suitably similar) their transmissions will interfere with one another, inhibiting the ability of other devices to decipher their individual signals.

The above situation is illustrated in Figure 5.17, where each wireless device appears at the centre of a grey circle denoting its transmission range. Figure 5.17(a) shows two devices, u and v, that are situated in each other's transmission ranges. This is sometimes known as a *primary collision* and implies that u and v should not broadcast using the same frequency. Figure 5.17(b), meanwhile, denotes a *secondary* collision . Here, although there is no primary collision between devices v_1 and v_2, it is still necessary that they broadcast at different frequencies in order to allow u to be able to distinguish between them.

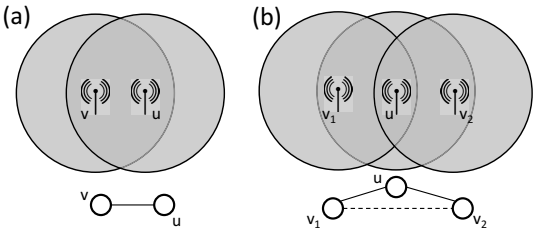

Fig. 5.17 Illustration of a primary collision (a), and (b) a secondary collision (dotted line) in a wireless network

The problem of choosing suitable frequencies for each device in a wireless network can be modelled as a graph colouring problem by relating each device to a vertex, with edges then occurring between any pair of vertices subject to a primary and/or secondary collision. Each frequency then corresponds to a colour and, due to the finite number of frequencies that exist in the radio spectrum, we now wish to colour this graph using the minimal number of colours.

More precisely, let $G = (V,E)$ be a graph with vertex set V and an edge E. The set of edges due to primary collisions, E_1, contains all pairs of devices that are close enough to be able receive each other's transmissions (as with Figure 5.17(a)). The set E_2 then contains pairs of devices subject to secondary collisions: that is, $\{v_i, v_j\} \in E_2$ if and only if the distance between v_i and v_j in the graph $G_1 = (V,E_1)$ is exactly 2 (as is the case in Figure 5.17(b)). If only primary collisions need to be considered when assigning frequencies, we only need to colour the graph G_1; otherwise we will need to colour the graph $G = (V, E = E_1 \cup E_2)$. In either case, this task is a type of decentralised graph colouring problem because each vertex (wireless device) is responsible for choosing its own colour (frequency), while being aware only of its neighbours and their current colours.

One simple but effective algorithm for the decentralised graph colouring problem is due to Finocchi et al. (2005). Let $G = (V,E)$ be a graph with maximal degree $\Delta(G)$. To begin, all vertices in G are set as uncoloured. Each vertex is also allocated

a set of candidate colours, defined $L_v = \{1,2,\ldots,\deg(v)+1\}\ \forall v \in V$. A single iteration of the algorithm then involves the following four steps.

1. In parallel, each uncoloured vertex v selects a *tentative* colour $t_v \in L_v$ at random.
2. In parallel, consider each tentatively coloured vertex v. If no neighbour of v is coloured with t_v, then set t_v as the *final* colour of v.
3. In parallel, consider each remaining tentatively coloured vertex v and
   a. Remove its tentative colour.
   b. Update L_v by deleting all colours from L_v that are assigned as final colours to neighbours of v.
   c. If $L_v = \emptyset$ then let l be the largest colour label assigned as a final colour in v's neighbourhood and set $L_v = \{1,2,\ldots,\min\{l+1,\Delta(G)+1\}\}$.
4. If any uncoloured vertices remain, return to Step 1.

An example run of this algorithm is shown in Figure 5.18. In the first iteration, we see that three of the five vertices are allocated final colours; the remaining two vertices are then allocated final colours in the second iteration.

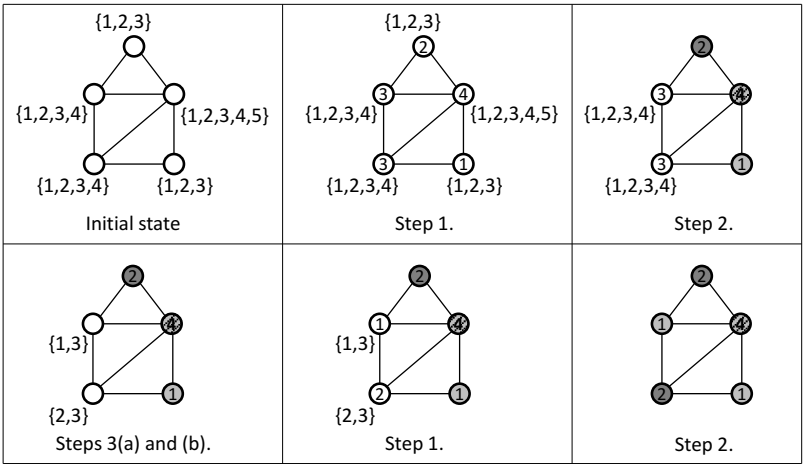

Fig. 5.18 Example run of Finocchi et al.'s algorithm. Here tentatively coloured vertices appear in white, with labels within the vertices indicating tentative colours

Note that in the above algorithm, each vertex v is initially assigned a set of candidate colours $L_v = \{1,2,\ldots,\deg(v)+1\}$. This means that L_v always contains sufficient options to allow each vertex v to be coloured differently from all of its neighbours; hence Step 3(c) will never actually be used. If, however, we desire a solution using fewer colours, we might choose to introduce a *shrinking factor* $s > 1$, which can be used to limit the initial set of candidate colours for each vertex v to

$L_v = \{1, 2, \ldots, \lfloor \frac{\deg(v)+1}{s} \rfloor\}$. In this case Step 3(c) might now be needed if the original contents of L_v prove insufficient. The algorithm may also need to execute for an increased number of iterations in order to achieve a feasible solution.

Finocchi et al. (2005) also suggest an improvement to this algorithm by replacing Step 2 with a more powerful operator. Observe in Step 2 of the first iteration of Figure 5.18 that there are two vertices tentatively coloured with colour 3. Accordingly, neither of these vertices receives a final colour at this iteration, though it is obvious that one of them could indeed receive colour 3 as a final colour at this point. An improvement to Step 2 therefore operates as follows. Let $G(i) = (V(i), E(i))$ be the subgraph induced by all vertices tentatively coloured with colour i. We now identify a maximal independent set for $G(i)$ and assign all vertices in this set to a final colour i. All other vertices in $G(i)$ remain uncoloured. To form this independent set, in parallel each vertex $v \in V(i)$ first generates a random number $r_v \in [0, 1]$. The tentative colour of a vertex v is then selected as its final colour if and only if r_v is less than the random numbers chosen by its neighbours in $G(i)$. This is equivalent to the greedy process of randomly permuting the vertices in $V(i)$ and then adding each vertex $v \in V(i)$ to the independent set if and only if it appears before its neighbours in the permutation.

Decentralised graph colouring problems arise in a number of practical situations, including TDMA slot assignment, wake-up scheduling and data collection (Hernández and Blum, 2014). One noteworthy piece of research is due to Kearns et al. (2006), who have examined the decentralised colouring of graphs representing social networks. In their case each vertex in the graph is a human participant, and two vertices are adjacent only if these people are judged to know one another. The objective of the problem is for each person to choose a colour for himself or herself only by using information regarding the colours of his or her neighbours. Participants are also able to change their colour as often as necessary until, ultimately, a feasible colouring of the entire graph is formed. This problem has real-world implications in situations where it is desirable to distinguish oneself from one's neighbours: for example, selecting a mobile phone ringtone that differs from one's friends, or choosing to develop professional expertise that differs from one's colleagues. In their research, Kearns et al. (2006) carried out a number of experiments on various graph topologies using segregated participants. Under a time limit of five minutes, topologies such as cycle graphs were optimally coloured quite quickly through the collective efforts of the participants. Other, more complex graphs modelling more realistic social network topologies were seen to present more difficulties, however.

5.6.2 Online Graph Colouring

In the online graph colouring problem, a graph is gradually revealed to an algorithm by presenting the vertices one at a time. The algorithm must then assign each vertex to a colour before receiving the next vertex in the sequence. In other words, an online graph colouring algorithm receives vertices in a given ordering $v_1, v_2, \ldots, v_n$ and the

colour for a vertex v_i is determined by only considering the colours of the vertices in the subgraph induced by the set $\{v_1,\ldots,v_i\}$. Once v_i has been coloured, it generally cannot be changed by the algorithm.

Much of the research relating to online graph colouring looks at the worst case behaviour of algorithms on particular topologies. Let A be an online graph colouring algorithm and consider the colourings of a graph G produced by A over all orderings of G's vertices. The maximum number of colours used among these colourings is denoted by $\chi_A(G)$. That is, $\chi_A(G)$ denotes the worst possible performance of A on G.

It has been shown by Lovász et al. (1989) that if G is a bipartite graph with n vertices, then there exists an online algorithm A such that

$$\chi_A(G) \leq 2\log_2 n. \tag{5.2}$$

Kierstead and Trotter (1981) have also shown that if G is an interval graph then there exists an online algorithm A such that

$$\chi_A(G) \leq 3\chi(G) - 2. \tag{5.3}$$

Studies of online colouring have also focussed on the behaviour of the GREEDY algorithm, which, we recall, operates by assigning each vertex to the lowest indexed colour seen to be feasible (see Section 2.1). Bounds noted by Gyárfás and Lehel (1988) include

$$\chi_{\text{GREEDY}}(G) \leq \chi(G) + 1 \tag{5.4}$$

if G is a split graph (i.e., a graph that can be partitioned into one clique and one independent set),

$$\chi_{\text{GREEDY}}(G) \leq \frac{3}{2}\chi(G) + 1 \tag{5.5}$$

if G is the complement of a bipartite graph, and

$$\chi_{\text{GREEDY}}(G) \leq 2\chi(G) - 1 \tag{5.6}$$

if G is the complement of a chordal graph.

Empirical work by Ouerfelli and Bouziri (2011) has also suggested that instead of following the GREEDY algorithm's strategy of assigning vertices to the lowest indexed feasible colour, it is often beneficial to assign vertices to the feasible colour containing the *most* vertices. This is because such a heuristic will often aid the formation of larger independent sets in a solution, ultimately helping to reduce the number of colours used in the final solution.

A real-world application of online graph colouring is presented by Dupont et al. (2009). Here, a military-based frequency assignment problem is considered in which wireless communication devices are introduced one by one into a battlefield environment. From a graph colouring perspective, given a graph $G = (V,E)$, the problem starts with an initial colouring of the subgraph induced by the subset of vertices $\{v_1,\ldots,v_i\}$. The remaining vertices $v_{i+1},\ldots,v_n$ are then introduced one at a time,

with the colour (frequency) of each vertex having to be determined before the next vertex in the sequence is considered. In this application the number of available colours is fixed from the outset, so it is possible that a vertex v_j $(i < j \leq n)$ might be introduced for which no feasible colour is available. In this case a repair operator is invoked that attempts to rearrange the existing colouring so that a feasible colour is created for v_j. Because such rearrangements are considered expensive, the repair operator also attempts to minimise the number of vertices that have their colours changed during this process.

5.6.3 Dynamic Graph Colouring

Dynamic graph colouring differs from decentralised and online graph colouring in that we again possess a global knowledge of the graph we are trying to colour. However, in this case a graph is also able to randomly change over time via the deletion and introduction of vertices, and/or the deletion and introduction of edges. We must therefore re-solve the problem at each stage.

A practical application of dynamic graph colouring might occur in the timetabling of lectures at a university (see Section 1.1.2 and Chapter 8). To begin, a general set of requirements and constraints will be specified by the university and an initial timetable will be produced. However, on viewing this draft timetable, circumstances might dictate that some constraints need to be altered, additional lectures need to be introduced, or other lectures need to be cancelled. This will result in a new timetabling problem that needs to be solved, with the process continuing in this fashion until a finalised solution is agreed upon.

More generally, a dynamic graph colouring problem can be defined by a sequence of graphs $\mathcal{G} = (G_1, G_2, \ldots, G_{|\mathcal{G}|})$, where a solution to each graph $G_i \in \mathcal{G}$ will need to be produced within a limited time frame before the next graph G_{i+1} is considered. One of the main issues to consider here is the level of similarity between two successive graphs G_i and G_{i+1} in this sequence. If G_i and G_{i+1} are seen to be quite alike, then it may be beneficial to use the solution generated for G_i as a starting solution for G_{i+1}. However, if the differences are larger, then it may be more appropriate to simply colour G_{i+1} from scratch.

5.7 List Colouring

The list colouring problem, like the (vertex) graph colouring problem, involves assigning colours to each vertex of a graph such that no pair of adjacent vertices are assigned to the same colour. However, in addition to this, when a problem is specified, individual vertices are also allocated their own list of permissible colours to which they can be assigned.

Defined more precisely, the list colouring problem gives us a graph $G = (V,E)$ and also a set L_v of permissible colours for each vertex $v \in V$. The sets L_v are usually referred to as a "lists", giving the problem its name. The task is to now produce a feasible colouring of G (that is, a colouring that is both proper and complete), with the added restriction that all vertices only be coloured using colours appearing in their corresponding lists: that is, $\forall v \in V, c(v) \in L_v$. If a k-colouring exists for a particular problem instance of the list colouring problem, we say that the graph G is "k-chooseable". The "choice number" $\chi_L(G)$ then refers to the minimum k for which G is k-chooseable. Note that the chromatic number of a graph $\chi(G) \leq \chi_L(G)$.

List colouring problems have obvious applications in areas such as timetabling where, in addition to scheduling events into a minimal number of timeslots (as we saw in Section 1.1.2), we might also face constraints of the form "event v can only be assigned to timeslots x and y", or "event u cannot be assigned to timeslot z". List colouring, stated as a decision problem, is also NP-complete because it generalises the graph colouring problem itself. That is, all k-colouring problems can be easily converted into an equivalent list colouring decision problem of deciding whether G is k-chooseable by simply setting $L_v = \{1,2,\ldots,k\}, \forall v \in V$.

In practice, algorithms for the graph colouring problem can often be used for deciding whether a graph is k-chooseable, beyond those for which $L_v = \{1,2,\ldots,k\}, \forall v \in V$. More specifically, graph colouring algorithms can be used to tackle any list colouring problem for which our chosen $k \geq |L|$, where L is defined as the union of all lists: $L = \bigcup_{v \in V} L_v$. Imagine we have a list colouring problem defined on a graph G for which $k \geq |L|$ is satisfied. First, we create a new graph G' by copying the vertices and edges of G and then adding k additional vertices, which we label $u_1, u_2, \ldots, u_k$. Next we add edges between all $\binom{k}{2}$ pairs of these additional vertices so that they form a complete graph K_k. This implies that any feasible colouring of G' must use at least k different colours. Without loss of generality, we can assume that $c(u_i) = i$ for $1 \leq i \leq k$. Finally we then go through each vertex v in G' that came from the original graph G and consider its colour list, adding an edge between v and u_i if colour $i \notin L_v$. This has the effect of disallowing v from being assigned to colour i as required.

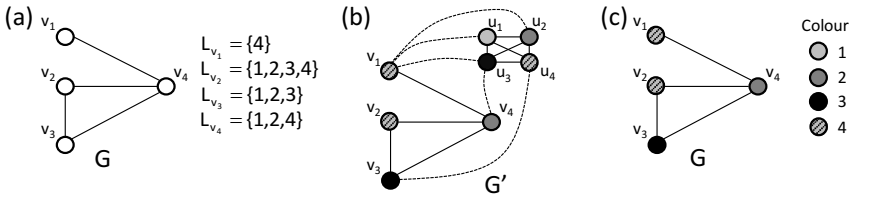

Fig. 5.19 Illustration of how a list colouring problem (a) can be converted into an equivalent graph colouring problem (b), whose colouring then represents a feasible solution to the original list colouring problem (c)

Figure 5.19 demonstrates this process. In this example $k = |L| = 4$, meaning that four additional vertices $u_1, \ldots u_4$ are added (larger values for k are also permitted). The colouring produced for the extended graph G' also uses four colours, which is the chromatic number in this case. However, we also observe that none of the vertices originating from G are actually coloured with colour 1 in this example; hence we deduce that G is actually 3-chooseable, as shown in Figure 5.19(c).

5.8 Equitable Graph Colouring

Another extension to the graph colouring problem is the *equitable* colouring problem, where we seek to establish a feasible colouring of a graph G such that the sizes of the colour classes differ by at most 1. In other words, we seek to establish a feasible k-colouring so that exactly n mod k colour classes contain $\lceil n/k \rceil$ vertices, and the remainder contain exactly $\lfloor n/k \rfloor$ vertices.

Examples of equitable graph colouring problems occur quite naturally as extensions to the general graph colouring problem. In university timetabling for example, it might be desirable to minimise the number of rooms required by balancing the number of events per timeslot (see Section 1.1.2). Another application can be found in the creation of table plans for large parties. Imagine, for example, that we have n guests who are to be seated at k equal-sized tables, but that some guests are known to dislike each other and therefore need to be assigned to different tables. In this case we can model the problem as a graph by using vertices for guests, with edges occurring between pairs of guests who dislike each other. An extension to this application is the subject of Chapter 6.

Let $G = (V, E)$ be a graph with n vertices, a maximal degree $\Delta(G)$, and an independence number $\alpha(G)$.

Definition 5.6 *If V can be partitioned into k colour classes $\mathcal{S} = \{S_1, \ldots, S_k\}$ such that S_i is an independent set and $||S_i| - |S_j|| \leq 1 \; \forall i \neq j$, then $\mathcal{S}$ is said to be an* equitable k-colouring *of G.*

Definition 5.7 *The smallest k for which an equitable k-colouring of G exists is the* equitable chromatic number*, denoted by $\chi_e(G)$.*

Like the graph colouring problem, the equitable graph colouring problem is known to be NP-complete. This follows from the fact that the problem of deciding whether a graph G is k-colourable can be converted into an equitable k-colouring problem by simply adding an appropriate number of isolated vertices to G. Hence the equitable graph colouring problem generalises the standard graph colouring problem. An alternative proof is also due to Furmańczyk (2004), who shows that the problem of deciding whether $\chi_e(G) \leq 3$ is NP-complete, even when G is the line graph of a cubic graph.

Because a feasible equitable k-colouring of a graph G is also a feasible k-colouring of G, it is obvious that $\chi(G) \leq \chi_e(G)$. In some cases however, this bound

can be very poor. Consider star graphs, for example, which comprise a vertex set $V = \{v_1, \ldots, v_n\}$ and an edge set $E = \{\{v_1, v_i\} : i \in \{2, \ldots, n\}\}$. These are a type of bipartite graph and therefore feature a chromatic number of 2. However, it is obvious that v_1 must be assigned a different colour to all other vertices; hence, in an equitable colouring all other colour classes must contain a maximum of two vertices. The equitable chromatic numbers of star graphs are therefore calculated as $\lceil (n-1)/2 \rceil + 1$, as illustrated in Figure 5.20.

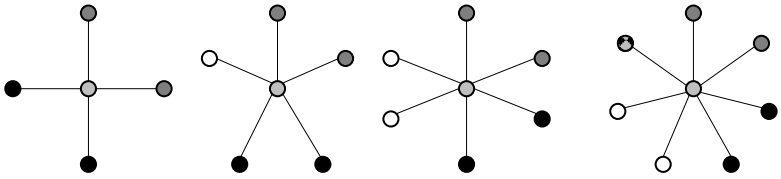

Fig. 5.20 The equitable chromatic numbers for star graphs with $n = 5, 6, 7$, and 8 are 3, 4, 4, and 5 respectively

A better lower bound for equitable chromatic numbers on general graphs is outlined by Furmańczyk (2004). Suppose that G is equitably coloured, with vertex v assigned to colour 1. The number of vertices coloured with colour 1 is therefore at most $\alpha(G - \Gamma(v) - \{v\}) + 1$. Since this colouring is equitable, the number of vertices coloured with any other colour is then at most $\alpha(G - \Gamma(v) - \{v\}) + 2$. Hence:

$$\left\lceil \frac{n+1}{\alpha(G - \Gamma(v) - \{v\}) + 2} \right\rceil \leq \chi_e(G) \tag{5.7}$$

For example, using a star graph with $n = 8$ whose internal vertex v_1 is coloured with colour 1, this gives $\lceil \frac{8+1}{0+2} \rceil = 5$. Note, however, that this bound requires a graph's independence number to be calculated, which is itself an NP-hard problem.

With regard to upper bounds on $\chi_e(G)$, it is known that any graph can be equitably k-coloured when $k \geq \Delta(G) + 1$. Hence:

Theorem 5.15 (Hajnal and Szemerédi (1970)) *Let G be a graph with maximal degree $\Delta(G)$. Then $\chi_e(G) \leq \Delta(G) + 1$.*

This fact was initially conjectured by Erdős (1964), with a formal proof being published six years later by Hajnal and Szemerédi (1970). Shorter proofs of this theorem have also been shown by Kierstead and Kostochka (2008) and Kierstead et al. (2010). The latter publication also presents a polynomial-time algorithm for constructing an equitable $(\Delta(G) + 1)$-colouring. This method involves first removing all edges from G and dividing the n vertices arbitrarily into $\Delta(G)$ equal-sized colour classes. In cases where n is not a multiple of $\Delta(G)$, sufficient isolated vertices are added. The vertices are then considered in turn and, in each iteration i, the edges incident to vertex v_i are added to G. If v_i is seen to be adjacent to another vertex in its colour class, it is moved to a different feasible colour class, leading to a feasible colouring with up to $\Delta(G) + 1$ colours. If this colouring is not equitable, then a

polynomial-length sequence of adjustments are made to reestablish the balance of the colour classes.

It is notable that Theorem 5.15 above is similar to Theorem 2.5 from Chapter 2 which states that for any graph G, $\chi(G) \leq \Delta(G) + 1$. Meyer (1973) has gone one step further to even conjecture a form of Brooks' Theorem (2.7) for equitable graph colouring: every graph G has an equitable colouring using $\Delta(G)$ or fewer colours with the exception of complete graphs and odd cycles. Recall, however, that the problem of determining an equitable k-colouring for an arbitrary k and graph G is still NP-complete, implying the need for approximation algorithms and heuristics in general. One simple approach for achieving approximate equitable k-colourings (for $k \leq \Delta(G)$) can be achieved via a simple modification of the DSATUR algorithm: starting with k empty colour classes, take each vertex in turn according to DSATUR's heuristics and assign it to the feasible colour class containing the fewest vertices, breaking ties randomly.

Figure 5.21 summarises results achieved by this algorithm for random graphs $G_{500,p}$, using $p = 0.1$, 0.5 and 0.9, for a range of suitable k-values. For comparison's sake, the results of a second algorithm are also included here. This operates in the same manner except that vertices are assigned to a randomly chosen feasible colour. The cost here is simply the difference in size between the largest and smaller colour classes in a solution. Hence a cost of 0 or 1 indicates an equitable k-colouring.

Figure 5.21 clearly demonstrates that, for these graphs, the policy of assigning vertices to feasible colour classes with the fewest vertices brings about more equitably coloured solutions. We also see that the algorithm consistently achieves equitable colourings for the majority of k-values with the exception of those close to the chromatic number, and those where k is a divisor of n. For the former case, the low number of available colours restricts the choice of feasible colours for each vertex, often leading to inequitable colourings. On the other hand, when k is a divisor of n the algorithm is seeking a solution with a cost of 0, meaning that the last vertex considered by the algorithm must be assigned to the unique colour class containing one fewer vertex than the remaining colour classes. If this colour turns out to be infeasible (which often seems to be the case), this vertex will then need to be assigned to another colour class, resulting in a solution with a cost of 2. Note, however, that it might be possible to further improve these solutions by, for example, applying a local search algorithm with appropriate neighbourhood operators such as Kempe chain interchanges and pair swaps. An approach along these lines for a related problem is the subject of the case study presented in Chapter 6.

5.9 Weighted Graph Colouring

Further useful extensions of the graph colouring problem can be achieved through the addition of numeric *weights* to a graph. Typically, the term "weighted graph colouring" is used in situations where the vertices of a graph are allocated weights. However, the term is also sometimes used for problems where edges are weighted,

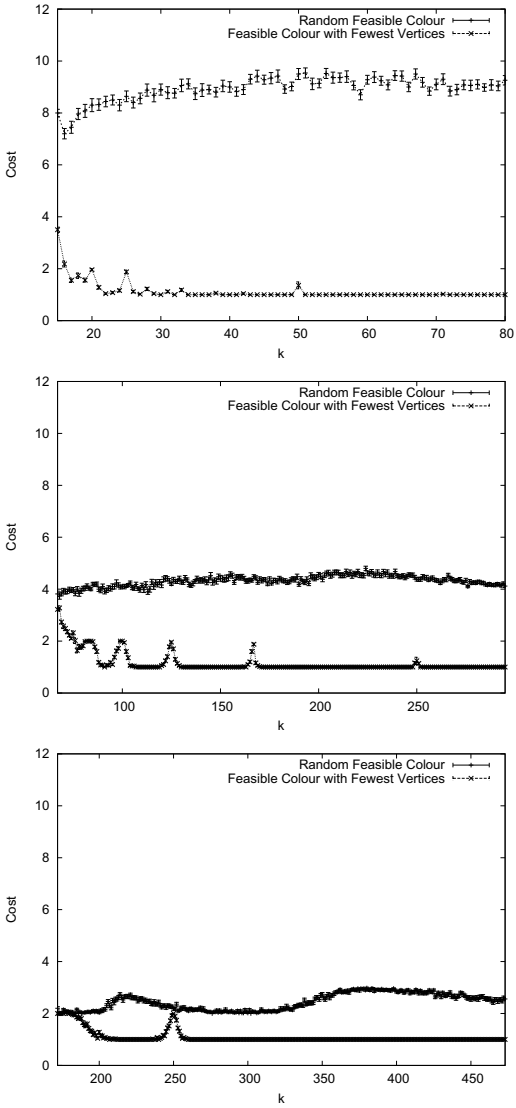

Fig. 5.21 Quality of equitable solutions produced by the modified DSATUR algorithms on random graphs with $n = 500$ for, respectively, $p = 0.1$, 0.5 and 0.9. All figures, are the average of 50 instances per k-value. Error bars show one standard error either side of these means

and for the multicolouring problem. These are considered in turn in the following subsections.

5.9.1 Weighted Vertices

A natural formulation of the weighted graph colouring problem is as follows. Let $G(V,E,w)$ be a graph for which each vertex $v \in V$ is given a nonnegative integer weight w_v. Given a fixed number of colours k, our task is to identify a proper (but perhaps partial) solution $\mathcal{S} = \{S_1, \ldots, S_k\}$ that maximises the objective function:

$$f(\mathcal{S}) = \sum_{i=1}^{k} g(S_i) \tag{5.8}$$

where $g(S_i) = \sum_{v \in S_i} w_v$. For this problem, if $\chi(G) \leq k$ then an optimal solution will obviously feature a cost of $\sum_{v \in V} w_v$, corresponding to a feasible graph colouring solution. On the other hand, if $k < \chi(G)$, then the problem involves finding an optimal subset of vertices $V' \subset V$, where $V' = \bigcup_{i=1}^{k} S_i$.

It is straightforward to show that this formulation is NP-hard by noting that any instance of the graph colouring problem can be transformed into this weighted variant by setting $w_v = 1 \; \forall v \in V$. If, in addition to this, $k = 1$, then the problem is also equivalent to the NP-hard maximum independent set problem.

This formulation of the weighted graph colouring problem arises in practical situations where a limited number of colour classes are available and where the colouring of certain vertices is considered more important than others. For example, in an exam timetabling problem we may be given a fixed number of timeslots (colours), and we might want to prioritise the assignment of larger exams (higher weighted vertices) to the timetable while also making sure that clashing exams (adjacent vertices) are not assigned to the same timeslots. One simple heuristic for this problem is to employ the GREEDY algorithm using a fixed number of colours k and an ordering of the vertices $v_1, v_2, \ldots, v_n$ such that $w_{v_1} \geq w_{v_2} \geq \ldots \geq w_{v_n}$. Algorithms that explore the space of partial proper solutions are also naturally suited. For example, we might make use of the PARTIALCOL algorithm while seeking to minimise the objective function $\sum_{v \in U} w_v$, where $U = V - \bigcup_{i=1}^{k} S_i$ is the set of uncoloured vertices (see Section 4.1.2).

Another well-known formulation of the weighted graph colouring problem involves taking a graph $G(V,E,w)$ as above, and then determining a feasible colouring $\mathcal{S} = \{S_1, \ldots, S_k\}$ that minimises Equation (5.8) using $g(S_i) = \max\{w_v : v \in S_i\}$. A practical example of this occurs in the scheduling of fixed-time jobs to timeslots. Imagine, for example, that we are given a set of jobs V, each with a processing time $w_v \; \forall v \in V$. Imagine further that these jobs are to be scheduled into k timeslots, and that the jobs assigned to a particular timeslot will be carried out simultaneously; hence, the *duration* of a timeslot S_i is set at $\max\{w_v : v \in S_i\}$. Finally, also suppose

that some pairs of jobs u, v are *incompatible* and cannot be assigned to the same timeslot. Such pairs correspond to edges $\{u, v\} \in E$.

In this formulation it is obviously in our interest to try and assign vertices with large weights to the same colour classes. Similarly, it also makes sense in many cases to reduce the number of colours being used by increasing the number of colour classes S_i for which $S_i = \emptyset$. Demange et al. (2007), however, have noted that the optimal number of nonempty colour classes for a particular graph G may well be larger than $\chi(G)$, though it will also always be less than or equal to $\Delta(G)+1$.

As with the previous example, this problem formulation is seen to be NP-hard by observing that any instance for which $w_v = 1$ $(\forall v \in V)$ is equivalent to the standard graph colouring problem. Furthermore, the problem remains NP-hard even for interval graphs (Escoffier et al., 2006) and bipartite graphs (Demange et al., 2007). For the bipartite case, Demange et al. (2007) have provided an algorithm with approximation ratio of $4r_w/(3r_w+2)$ (where $r_w = \frac{\max\{w_v : v \in V\}}{\min\{w_v : v \in V\}}$). They also prove that optimal solutions for bipartite graphs can be found in polynomial time whenever $|\{w_v : v \in V\}| \leq 2$. For general graphs, an approximation algorithm is also suggested that operates as follows. As before, let $G = (V, E, w)$ be a graph with weighted vertices and $g(S_i) = \max\{w_v : v \in S_i\}$.

1. Construct a graph with weighted edges $\bar{G} = (V, E', w')$ where $\bar{G}$ is the complement of G, and for any edge $\{u, v\} \in E'$, $w'_{uv} = w_u + w_v - g(\{w_u, w_v\})$.
2. Compute a maximum weighted matching M^* of $\bar{G}$.
3. For each edge in M^*, colour the end points with a new colour.
4. Colour any remaining vertices with their own new colour.

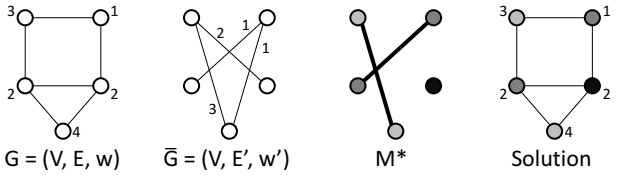

Fig. 5.22 Illustration of the algorithm of Demange et al. (2007)

An example of this process is shown in Figure 5.22. The matching M^* can be determined in polynomial time using methods such as the blossom algorithm (Kolmogorov, 2009). Note that each colour class in the solution is an independent set, but that these are limited to contain a maximum of two vertices. Indeed, in graphs where no independent set contains more than two vertices (such as the complement of a bipartite graph), this algorithm guarantees the optimal. In further work, Hassin and Monnot (2005) have shown that, for any graph, this process produces a solution whose objective function value never exceeds twice the optimum. They also show that the same approximation ratio applies when $g(S_i)$ takes other forms such as $g(S_i) = \min\{w_v : v \in S_i\}$ and $g(S_i) = \frac{1}{|S_i|}\sum_{v \in S_i} w_v$. Malaguti et al. (2009) have also proposed a number of IP-based methods for this problem similar in spirit to

those discussed in Section 3.1.2. In particular, they propose the use of heuristics for building up a large sample of independent sets, and then use an IP model similar to that of Section 3.1.2.1 to select a subset of these. Local search-based methods based on Kempe chain interchanges and pair swaps also seem to be naturally suited to this problem.

5.9.2 Weighted Edges

In many cases it is more convenient to apply weights to the *edges* of a graph as opposed to the vertices, allowing us to express levels of preference for assigning vertices to different (or the same) colours. One interpretation involves taking a graph $G(V,E,w)$ for which each edge $\{u,v\} \in E$ is allocated an integer weight w_{uv}. Given a fixed number of colours k, our task is to then identify a complete (but perhaps improper) solution $\mathcal{S} = \{S_1, \ldots, S_k\}$ that minimises the objective function:

$$f(\mathcal{S}) = \sum_{i=1}^{k} \sum_{u,v \in S_i : \{u,v\} \in E} w_{uv} \tag{5.9}$$

Here, if $\forall \{u,v\} \in E,\ w_{uv} > 0$, then any solution for which $f(\mathcal{S}) = 0$ corresponds to a feasible k-coloured solution.

This sort of formulation is applicable in areas such as university timetabling and social networking. For the former, imagine, as before, that we wish to assign events (vertices) to timeslots (colours), but that there are insufficient timeslots to feasibly accommodate all events. In order to form a complete timetable, this means that some clashes will be necessary; however, some types of clashes may be deemed less critical than others. For example, if two clashing events only have a small number of common students, then we may allow them to both be assigned to the same timeslots (with alternative arrangements then being made for the people affected). On the other hand, if two events contain a large number of common participants, or if the same instructor is required to teach them both, then such a clash would be far less desirable. Appropriate weights added to the corresponding edges can be used to express such preferences.

Note that due to the nature of this problem's requirements, algorithms that search the space of complete improper solutions will often be naturally suitable here. In Chapter 6 an application along these lines will be made to the problem of partitioning members of social networks, where edge weights are used to express a level of "liking" or "disliking" between pairs of individuals.

5.9.3 Multicolouring

Another problem that is sometimes referred to as "weighted graph colouring" is the NP-hard graph *multicolouring* problem. In this case we are given a graph $G(V,E,w)$ for which each vertex $v \in V$ is allocated a weight $w_v \in \{1,2,\ldots\}$. The task is to then assign w_v different colours to each vertex v such that (a) adjacent vertices have no colours in common, and (b) the number of colours used is minimal.

Multicolouring has practical applications in areas such as frequency assignment problems where, in some cases, it is desirable for devices to be able to transmit and receive messages on multiple frequencies as opposed to just one (Aardel et al., 2002). McDiarmid and Reed (2000) have shown that this problem is polynomially solvable for bipartite and perfect graphs, but that it remains NP-hard for triangular lattice graphs and their induced subgraphs, which have practical applications in cellular telephone networks. They also suggest a suitable polynomial-time approximation algorithm for the latter topology.

Note that the graph colouring problem is a special case of the multicolouring problem for which $w_v = 1\ \forall v \in V$. On the other hand, any instance of the multicolouring problem can also be converted into an equivalent graph colouring problem by replacing each vertex $v \in V$ with a clique of size w_v, and then connecting every member of the clique to all neighbours of v in G. This method of conversion allows us to use any graph colouring algorithm (such as those from Chapter 4) with the graph multicolouring problem, though it also increases the number of vertices to colour by a factor of $\sum_{v \in V} w_v/n$. Consequently, graph multicolouring is often studied as a separate computational problem, for which the backtracking algorithm of Caramia and Dell'Olmo (2001) and the IP branch-and-price method of Mehrotra and Trick (2007) are prominent examples.