Chapter 13

Graph Colouring Algorithms

Thore Husfeldt, rev. 69eb646, 2013-06-18

This chapter presents an introduction to graph colouring algorithms. The focus is on vertex colouring algorithms for sequential computation that work on general classes of graphs.

Contents

1. Introduction

A *vertex q-colouring* of a graph is an assignment of q different values, called *colours*, to the vertices such that each pair of adjacent vertices receives different colours. The straightforward algorithm for finding a colouring is to systematically search among all mappings from vertices to colours, a technique often called *exhaustive* or *brute force*:

Algorithm X (*Exhaustive search*). *Given integer $q \geq 1$ and a finite graph G without loops or multiple edges, output a vertex q-colouring if it exists.*

X1. [Main loop.] For every mapping $f : V \to \{0, \ldots, q-1\}$, do step X2.

X2. [Check f.] If every edge $vw \in E$ satisfies $f(v) \neq f(w)$ then output f. ∎

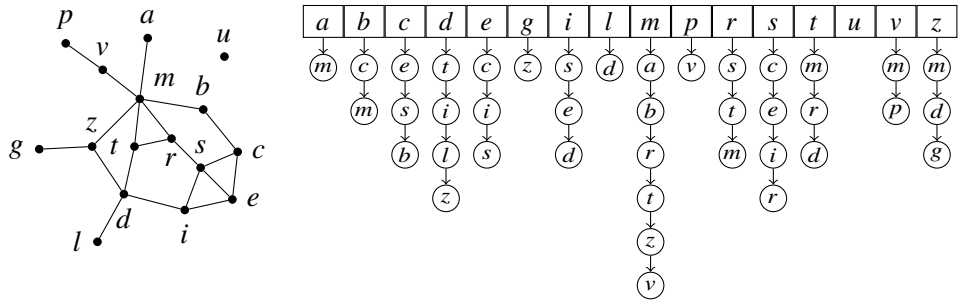


Fig. 1: A graph and its representation as an array of lists.

This algorithm has few redeeming qualities, except for being correct. We consider it here because it serves as an opportunity to make explicit the framework in which we will present more interesting algorithms later.

Model of computation. The number of operations used by algorithm X can be asymptotically bounded by $O(q^n(n + m))$, we call this the *running time* of the algorithm.

To make such a claim, we tacitly assume a computational model that includes primitive operations such as iterating over all mappings from one finite set A to another finite set B in time $O(|B|^{|A|})$ (step X1), or iterating over all edges in time $O(n + m)$ (step X2). For instance, we assume that the input graph is represented by an array of sequences as in figure 1. This allows us to iterate over the neighbours of a vertex in time $O(\deg v)$. (An alternative representation, such as an incidence or adjacency matrix, would not.) Note that it is *not* a primitive operation to detect if two graphs are isomorphic. The convention of expressing computational resources using Landau notation is consistent with our somewhat cavalier attitude towards the details of our computational model. Our assumptions are consistent with the behaviour of a modern computer in a high-level programming language. Nevertheless, we will explain our algorithms in plain English.

Worst case asymptotic analysis. One observes that we could have fixed the colouring of a specific vertex v to $f(v) = 0$, reducing X's running time to $O(q^{n-1}(n+m))$. A moment's thought shows that this reasoning can be extended to cliques of size $r \geq 1$: Search through all $\binom{n}{r}$ induced subgraphs until a clique of size r is found, arbitrarily map these vertices to $\{0, \ldots, r-1\}$ and let algorithm X colour the remaining vertices. This reduces the running time to $O(q^{n-\omega(G)}n^{\omega(G)}(n+m))$, where $\omega(G)$ is the clique size. This may be quite good for some graphs. Another observation is that in the best case, the running time is $O(n + m)$. However, we will

normally not pursue this kind of argument. Instead, we are maximally pessimistic about the input and the algorithm's underspecified choices. In other words, we understand running times as worst case performance guarantees, rather than 'typical' running times or average running times over some distribution.

Sometimes we may even say that algorithm X requires time q^n poly(n), where we leave the polynomial factor unspecified in order to signal the perfunctory attention we extend to these issues.

Overview and notation. Straightforward variants of algorithm X can be used to solve many other graph colouring problems. For instance, to find a list colouring, we restrict the range of values for every $f(v)$ to a given list; to find an edge colouring, we iterate over all mappings $f\colon E \to \{0, \ldots, q-1\}$. Another modification is to count the number of colourings instead of finding one. These extensions provide baseline algorithms for chromatic index, list colouring, the chromatic polynomial, etc. However, for purposes of exposition, we will try to present algorithms in their *least* general form, emphasizing the algorithmic idea rather than its (sometimes quite pedestrian) generalisations. The algorithms are organised by algorithmic technique rather than problem type, graph class, optimality criterion, or computational complexity. These sections are largely independent and can be read in any order, except maybe for algorithm G in section 2. The final section 7 takes a step back and relates the various colouring problems to each other.

Our graphs are finite, undirected, without loops and multiple edges. They have n vertices V and m edges E. The degree of vertex v is $\deg v$, the graph's maximum and minimum degrees are Δ and δ. The neighbours of v are $N(v) = \{\, w \in V\colon vw \in E \,\}$. We use $\chi(G)$ for the chromatic number (the minimum number of colours in a vertex colouring), $\chi'(G)$ for the chromatic index (the minimum number of colours in an edge colouring), and $P(G, q)$ for the number of q-colourings of G.

2. Greedy colouring

The following algorithm, sometimes called *greedy* or *sequential*, considers the vertices one by one and uses the first available colour.

Algorithm G (*Greedy vertex colouring*). *Given an algorithm G of maximum degree Δ with an ordering $v_1, \ldots, v_n$ on its vertices, computes a vertex colouring f with $\max_i |\{\, j < i\colon v_j v_i \in E \,\}| + 1 \le \Delta + 1$ colours.*

G1. [Initialize.] Set $i = 1$.

G2. [Next vertex.] If $i = n + 1$ then terminate with f as the output. Otherwise, increase i.

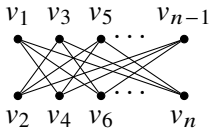
G3. [Find colours of v_i's neighbours.] Compute the set $C = \bigcup_{j<i} f(v_j)$ of colours already assigned to the neighbours of v_i.

G4. [Assign smallest available colour to v_i.] For increasing $c = 0, 1, \ldots$ check if $c \in C$. If not, set $f(v_i) = c$ and return to G2. ∎

For the number of colours, it is clear that l in step G4 is at most $|C|$, which equals the number of neighbours of v_i among $v_1, \ldots, v_{i-1}$. In particular, algorithm G establishes that $\chi(G) \leq \Delta(G) + 1$.

For the running time, note that both steps G3 and G4 take at most $O(1 + \deg v_i)$ operations. Summing over all v, the total time spent in G3 and G4 is asymptotically bounded by $n + (\deg v_1 + \cdots + \deg v_n) = n + 2m$. Thus, algorithm G takes time $O(n + m)$.

Optimal ordering. The size of the colouring computed by Algorithm G strongly depends on the vertex ordering. The worst-case behaviour of G is poor. For instance, it spends $n/2$ colours on the 2-colourable *crown graph*:



On the other hand, for every graph there exists an ordering such that G uses an optimal number of colours; indeed, any ordering that satisfies $f(v_i) \leq f(v_{i+1})$ for an optimal colouring f will have this property. Since there are $O(n!)$ different orderings, this observation is algorithmically quite useless. An ordering is *perfect* for a graph if for every induced subgraph, algorithm G results in an optimal colouring; triangulated graphs and comparability graphs admit such an ordering [12].

Randomness. Algorithm G performs quite well on random graphs, no matter the ordering. For almost all n-vertex graphs, it uses $n/(\log n - 3 \log \log n)$ colours, which is roughly twice the optimum [16].

This suggests a simple randomized algorithm: On input graph G, choose a vertex ordering at random and then execute algorithm G. For many problems, this is a sound algorithmic design strategy, trading good average-case behaviour for good (expected) worst case behaviour. However, for algorithm G the result is quite poor: For every $\epsilon > 0$ there exists graphs with chromatic number n^ϵ for which the randomized algorithm expends $\Omega(n/\log n)$ colours with high probability [24].

Other orderings. The largest-first ordering [34], $\deg v_1 \geq \cdots \geq \deg v_n$, shows that $\chi(G) \leq 1 + \max_i \min\{\deg v_i, i - 1\}$, a bound that is sometimes better than Δ. The smallest-last ordering [30] is given as follows: choose as the last vertex v_n the vertex of minimum degree in G, and proceed recursively with $G \setminus \{v_n\}$. With this ordering, the size of the resulting colouring can be bounded by the Szekeres–Wilf bound [33],

$$\chi(G) \leq \operatorname{dgn}(G) + 1 .$$

Here, the *degeneracy* $\operatorname{dgn}(G)$ is the maximum over all subgraphs H of G of their minimum degree $\delta(H)$. This ordering optimally colours crown graphs and many other classes of graphs, and uses 6 colours on any planar graph.

Other orderings are dynamic in the sense that the ordering is determined during the algorithm's execution instead of before it. For example, Brélaz [7] suggests to choose the vertex among those adjacent to the largest number of different colours. Many other orderings have been investigated. Some of them perform quite well on instances that one may encounter 'in practice,' but attempts at formalising what this means are quixotic. See [23] and [29] for surveys.

2-colourable graphs

Of particular interest are those vertex orderings in which every v_i has an edge to some v_j with $j < i$. Such orderings can be computed in time $O(m + n)$ using basic graph traversal algorithms such as depth-first or breadth-first search. This algorithm is sufficiently important to be made explicit:

Algorithm B (*Bipartition*). *Given a connected graph G, computes a 2-colouring if it exists. Otherwise, finds an odd cycle.*

B1. [Initialize.] Let $f(v_1) = 0$ and let Q (the 'queue') be an empty sequence. For every neighbour w of v_1, set $p(w) = v_1$ (the 'parent' of w) and add w to Q.

B2. [Next vertex.] If Q is empty, terminate. Otherwise, remove the first vertex v from Q. Set $f(v) = 1 - f(p(v))$. For every neighbour w of v, if w is not yet coloured and does not belong to Q then add it to the end of Q. Repeat step B2.

B3. [Verify 2-colouring.] Iterate over all edges vw and verify that $f(v) \neq f(w)$ for all $vw \in E$. If so, return f and terminate.

B4. [Construct odd cycle.] Find vw with $f(v) = f(w)$. Let u denote the nearest common ancestor of v and w in the tree defined by p. Return the path $w, p(w), \ldots, u$, followed by the reversal of the path $v, p(v), \ldots, u$, followed by the edge vw. ∎

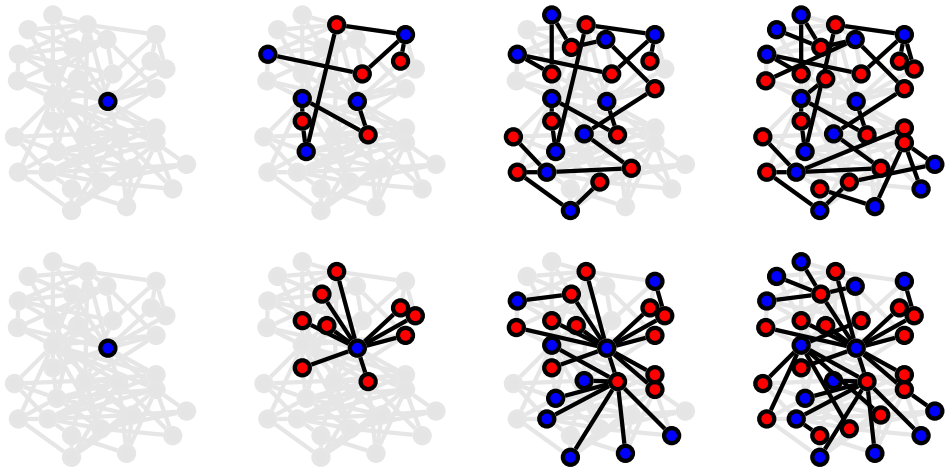


Fig. 2: Algorithm B using depth-first (top) and breadth-first search (bottom).

Algorithm B is an example of a 'certifying' algorithm: An algorithm that produces a witness to certify its correctness, in this case that the graph is not 2-colourable.

To see that the cycle returned in B4 is indeed odd, note that on the two paths $w, p(w), \ldots, u$ and $v, p(v), \ldots, u$, every vertex has a different colour than its parent. Since the respective endpoints of both paths have the same colour, they must contain the same number of edges modulo 2. In particular, their total length is even. With the additional edge vw, the length of the resulting cycle is odd.

Wigderson's algorithm

Algorithm B and G appear together in the following algorithm [35].

Algorithm W (*Wigderson's algorithm*). *Given a graph G with $\chi(G) = 3$, computes a vertex colouring with $O(\sqrt{n})$ colours.*

W1. [Initialize.] Set q, the number of colours used, to 0.

W2. [$\Delta(G) \geq \lceil\sqrt{n}\rceil$.] Consider a vertex v in G with $\deg v \geq \lceil\sqrt{n}\rceil$; if no such vertex exists, proceed to W3. Use algorithm B to 2-colour the neighbourhood $G[N(v)]$ with colours q and $q+1$. Remove $N(v)$ from G, increase q by $\chi(G[N(v)])$. Repeat step W2.

W3. [$\Delta(G) < \lceil\sqrt{n}\rceil$.] Use algorithm G to colour the remaining vertices with the colours $q, \ldots, q + \lceil\sqrt{n}\rceil$. ∎

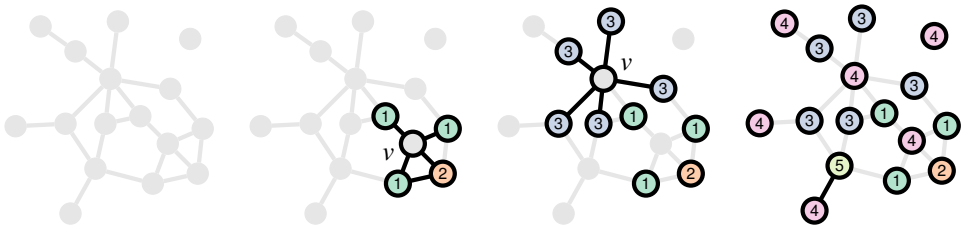


Fig. 3: Algorithm W on a 3-colourable 16-vertex graph, using 5 colours.

The running time is clearly bounded by $O(n + m)$. To analyse the number of colours, we first need to verify that step W2 is in fact correct. Since G is 3-colourable, so is the subgraph induced by $N(v) \cup \{v\}$. Now, if $G[N(v)]$ would require 3 colours then $G[N(v) \cup \{v\}]$ would require 4 colours. Hence $G[N(v)]$ is 2-colourable. Note that W2 can be run at most $O(\sqrt{n})$ times, each using at most 2 colours. Step W3 expends another $\lceil\sqrt{n}\rceil$ colours according to algorithm G.

W extends to graphs with $\chi(G) > 3$ quite naturally. In this case, step W2 calls W recursively to colour $(\chi(G) - 1)$-colourable neighbourhoods. The resulting algorithm uses $O(n^{1-1/(1-\chi(G))})$ colours.

3. Recursion

Recursion is a fundamental algorithmic design technique. The idea is to reduce a problem to one or more simpler instances of the same problem.

Contraction

The oldest recursive construction for graph colouring expresses $P(G, q)$ and $\chi(G)$ in terms of edge contractions. We write $G + vw$ for the graph G with the edge vw added, and $G - vw$ for the graph G with the edge vw removed. The *contraction* of two vertices $v, w \in V$ in G is the graph G/vw where v and w are identified into a single vertex, which is adjacent to all neighbours $N(v) - \{w\}$ and $N(w) - \{v\}$. In particular, the resulting graph has no loops or multiple edges.

Lemma 1. *For vertices v, w with $vw \notin E$ and $q = 0, \ldots, n$,*

$$P(G, q) = P(G + vw, q) + P(G/vw, q)\,. \tag{13.1}$$

Proof. Consider a q-colouring f of G. If $f(v) = f(w)$ then f is also a colouring of G/vw but not of $G + vw$. If $f(v) \neq f(w)$ then f is also a colouring of $G + vw$ but not of G/vw. ∎

A similar expression can be given in for the chromatic number,

$$\chi(G) = \min\{\chi(G + vw), \chi(G/vw)\} \qquad (vw \notin E)\,.$$

The base of these 'addition–contraction' recurrences is reached when the graph is complete. For instance,

$P($  $, q) = P($$, q) + P($$, q)$

$$= P(K_5, q) + P(K_3, q) = q(q-1)(q-2)(1 + (q-3)(q-4))\,.$$

For sparse graphs, it is more useful to give the recurrence on 'deletion–contraction' form. This will recurse on edges until the graph is empty:

$$P(G, q) = P(G/e, q) - P(G - e, q) \qquad (e \in E)\,.$$

Many other graph problems beside colouring can be expressed by a deletion–contraction recurrence. The most general graph invariant that can be defined (and therefore computed) in this fashion is the Tutte polynomial.

The algorithm implied by these recursions is sometimes called Zykov's algorithm [40]. We present the deletion–contraction version.

Algorithm C (*Contraction*). *Given a graph G, computes $P(G, q) = \sum_{i=0}^{n} a_i q^i$ as a sequence of coefficients $(a_0, \ldots, a_n)$.*

C1 [Base.] If G has no edges then return the coefficients $(0, \ldots, 0, 1)$, corresponding to the polynomial $P(G, q) = q^n$.

C2 [Recursion.] Pick an edge e and construct the graphs $G' = G/e$ and $G'' = G - e$. Call C recursively to compute $P(G', q)$ and $P(G'', q)$ as sequences of coefficients $(a'_0, \ldots, a'_n)$ and $(a''_0, \ldots, a''_n)$. Return $(a'_0 - a'_0, \ldots, a'_n - a''_n)$, corresponding to the polynomial $P(G/e, q) - P(G - e, q)$. ∎

To analyse the running time, let $T(r)$ denote the number of executions of C1 for graphs with n vertices and m edges, where $r = n + m$. The two graphs constructed in C2 have sizes $n - 1 + m - 1 = r - 2$ and $n + m - 1 = r - 1$, respectively, so $T(r)$ satisfies $T(r) = T(r-1) + T(r-2)$. This is a well-known recurrence with solution $T(r) = O(\phi^r)$, where $\phi = \frac{1}{2}(1 + \sqrt{5})$ is the golden ratio. Thus, algorithm C requires $O(1.619^{n+m})$ time. A similar analysis for the algorithm implied by the deletion–addition recursion gives $O(1.619^{n+\overline{m}})$, where $\overline{m} = \binom{n}{2} - m$ is the number of nonedges.

These worst-case bounds are often very pessimistic. They do not take into account that recurrences can be stopped as soon as G is a tree (or some other

graph whose solution is known), or that P factors over connected components. Moreover, heuristics for graph isomorphism and memoization can be employed to avoid unnecessary recomputation of isomorphic subproblems. Thus, algorithm C is a more useful algorithm than its exponential running time may indicate.

Vertex partitions

The following result yields a recurrence for $\chi(G)$ in terms of induced subgraphs.

Lemma 2. *There exists a vertex partition* $(S, V \setminus S)$ *into nonempty parts such that*

$$\chi(G) = \chi(G[S]) + \chi(G[V \setminus S])\,.$$

Moreover, we can assume that S *is an independent set in* G*.*

Proof. Let S denote one of the colour classes in an optimal colouring. In particular, S is a nonempty, independent set. ∎

Equation (2) is a tempting 'divide-and-conquer' recurrence, but since we do not know S, no useful algorithm is immediate. However, we can consider *all* partitions. With the assumption that S can be taken to be an independent set, the above lemma gives the following recurrence for the chromatic number:

$$\chi(G) = \min_S\{\,\chi(G[S]) + \chi(G[V \setminus S])\,\} = 1 + \min_S \chi(G[V \setminus S])\,, \tag{13.2}$$

where the minimum is taken over all nonempty independent sets in G. Still, the implied recursive algorithm is too slow to be of interest.

Dynamic programming. The central observation underlying the fundamental algorithmic idea of *dynamic programming* is that the subproblems $\chi(G[W])$ for various W appearing in (13.2) are computed over and over. Therefore it makes sense to store these 2^n values in a table when they are first computed. Subsequent evaluations can then be handled by table lookup instead of recursive computation.

We express the resulting algorithm in a bottom-up fashion:

Algorithm D (*Dynamic programming*). *Given a graph* G*, computes a table with* $T(W) = \chi(G[W])$ *for every* $W \subseteq V$*.*

D1 [Initialize.] Construct a table with (initially undefined) entries $T(W)$ for every $W \subseteq V$. Set $T(\emptyset) = 0$.

D2 [Main loop.] List all vertex subsets $W_1, \ldots, W_{2^n} \subseteq V$ in nondecreasing order of size. Do step D3 for every $W = W_2, \ldots, W_{2^n}$, then terminate.

D3 [Determine $T(W)$.] Set $T(W) = 1 + \min_S T(W \setminus S)$ where the minimum is taken over all nonempty, independent sets S in $G[W]$. ∎

The ordering of subsets in the main loop D2 ensures that every set is handled before any of its supersets. In particular, all values $T(W \setminus S)$ needed in D3 will have been previously computed, so the algorithm is well defined. The minimisation in step D3 is implemented by iterating over all $2^{|W|}$ subsets of W. Thus the total running time of algorithm D is within a polynomial factor of

$$\sum_{W \subseteq V} 2^{|W|} = \sum_{k=0}^{n} \binom{n}{k} 2^k = 3^n \,. \tag{13.3}$$

This rather straightforward application of dynamic programming already provides the nontrivial insight that the chromatic number can be computed in *vertex-exponential* time, rather than depending exponentially on m, $\chi(G)$, or a superlinear function of n.

Maximal independent sets. To pursue this idea a little further we see that S in lemma 2 can be assumed to be a *maximal* independent set, i.e., not a proper subset of another independent set. To see this, let f be an optimal colouring and consider the colour class $S = f^{-1}(0)$. If S is not maximal then repeatedly pick vertices v nonadjacent to S and set $f(v) = 0$.

Clearly, a graph of size k can have $3^{k/3}$ maximal independent sets (consider a disjoint union of $k/3$ triangles). It is known that this is also an upper bound, and that the maximal independent sets can be enumerated within a polynomial factor of that bound (see [9], [32] and [38]).

Lemma 3. *The maximal independent sets of a graph on k vertices can be listed in time $O(3^{k/3})$ and polynomial space.*

We can apply this idea to algorithm D. The minimisation in D3 now takes the form

D3′ [Determine $T(W)$.] Set $T(W) = 1 + \min_S T(W \setminus S)$ where the minimum is taken over all maximal independent sets in $G[W]$.

Using lemma 3 for the minimisation in D3′, the total running time of algorithm D comes within a polynomial factor of

$$\sum_{k=0}^{n} \binom{n}{k} 3^{k/3} = (1 + 3^{1/3})^n = O(2.45^n) \,.$$

This was the fastest algorithm for the chromatic number for many years.

Of particular interest is the 3-colouring case. Here, it makes more sense to let the outer loop iterate over all maximal independent sets and test that the complement is bipartite:

Algorithm L (*Lawler's algorithm*). *Given a graph G, computes a 3-colouring if it exists.*

L1 [Main loop.] For every maximal independent set S of G, do step L2.

L2 [Try $f(S) = 2$.] Use algorithm B to determine if there exists a colouring $f : V \setminus S \to \{0, 1\}$ of $G[V \setminus S]$. If so, extend f to all of V by setting $f(v) = 2$ for all $v \in S$ and terminate with the colouring f. ∎

The running time of algorithm L is dominated by the number of executions of L2, which is $3^{n/3}$ according to lemma 3. Thus, algorithm L decides 3-colourability in time $O(1.45^n)$ and polynomial space.

The use of maximal independent sets goes back to Christofides [11]; algorithms D and L are due to Lawler [26]. A series of improvements to these ideas have further reduced these running times. At the time of writing, the best known time bound for 3-colouring is $O(1.329^n)$ [6] and for 4-colouring is $O(1.76^n)$ [10].

4. Subgraph expansion

The *Whitney expansion* [36]

$$P(G, q) = \sum_{A \subseteq E} (-1)^{|A|} q^{k(A)} ,$$

expresses the chromatic polynomial as an alternating sum of terms each of which depends on the number of connected components $k(A)$ of the edge subset $A \subseteq E$. Determining $k(A)$ is a well-studied algorithmic graph problem, and can be solved in time $O(n + m)$, for example by depth-first search. Thus, the Whitney's subgraph expansion can be evaluated in time $O(2^m(n + m))$.

A more recent expression [2] expands over induced subgraphs instead:

Lemma 4. *For $W \subseteq V$, let $g(W)$ denote the number of nonempty independent sets in $G[W]$. Then G can be q-coloured if and only if*

$$\sum_{W \subseteq V} (-1)^{|V \setminus W|} (g(W))^q > 0 . \tag{13.4}$$

Proof. For every $W \subseteq V$, the term $(g(W))^q$ counts the number of ways to pick q nonempty independent sets $S_1, \ldots, S_q$, where $S_i \subseteq W$. For $U \subseteq V$ let $h(U)$ denote the number of ways to pick nonempty independent sets $S_1, \ldots, S_q$ whose union is U. Then $(g(W))^q = \sum_{U \subseteq W} h(U)$, so

$$\begin{aligned}\sum_{W \subseteq V} (-1)^{|V \setminus W|} (g(W))^q &= \sum_{W \subseteq V} (-1)^{|V \setminus W|} \sum_{U \subseteq W} h(U) \\ &= \sum_{U \subseteq V} h(U) \sum_{W \supseteq U} (-1)^{|V \setminus W|} \\ &= h(V)\,.\end{aligned}$$

To see the last step, the inner sum (over W with $U \subseteq W \subseteq V$) vanishes, except when $U = V$, because there are as many odd-sized and even-sized sets sandwiched between different sets. (This is the principle of inclusion–exclusion.)

If $h(V)$ is nonzero then there exists independent sets $S_1, \ldots, S_q$ whose union is V. These sets correspond to a colouring: associate a colour with the vertices in each set, breaking ties arbitrarily. ∎

For every $W \subseteq V$, the value $g(W)$ can be computed in time $O(2^{|W|}|E|)$ by constructing every nonempty subset of W and testing it for independence. Thus, the total running time for evaluating (13.4) is within a polynomial factor of 3^n, just like in the analysis (13.3) for algorithm D. However, the space requirement is only polynomial. Using dedicated algorithms for evaluating $g(W)$ from the literature, the running time can be reduced to $O(2.2461^n)$ [3].

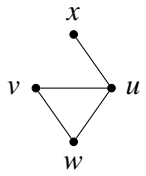
If exponential space is available, we can do even better. First, we introduce a recurrence for $g(S)$.

Lemma 5. *We have* $g(\emptyset) = 0$*, and*

$$g(W) = G(W \setminus \{v\}) + g(W \setminus N[v]) + 1 \qquad (v \in W)\,,$$

where $N[v] = \{v\} \cup \{\, w \in N : vw \in E \,\}$ *denotes the closed neighbourhood of* v*.*

Proof. Fix $v \in W$ and consider the nonempty independent sets $S \subseteq W$. They can be partitioned into two classes: either $v \in S$ or $v \notin S$. The latter sets are counted in $g(W \setminus \{v\})$. It remains to argue that the sets $S \ni v$ are counted in $g(W \setminus N[v]) + 1$. We will do this by counting the equipotent family of sets $S \setminus \{v\}$ instead. Since S contains v and is independent, it cannot contain other nodes in $N[v]$. Thus $S \setminus \{v\}$ is disjoint from $N[v]$ and contained in W. Now, either S is the singleton $\{v\}$ itself, accounted for by the '+1' term, or $S \setminus \{v\}$ is a nonempty independent set and therefore counted in $g(W \setminus N[v])$. ∎



W	g	a_2	a_3	W	g	a_2	a_3
$\emptyset$	0	0	0	$\{v,w\}$	2	4	8
$\{u\}$	1	−1	−1	$\{v,x\}$	3	9	27
$\{v\}$	1	−1	−1	$\{w,x\}$	3	9	27
$\{w\}$	1	−1	−1	$\{u,v,w\}$	3	−9	−27
$\{x\}$	1	−1	−1	$\{u,v,x\}$	4	−16	−64
$\{u,v\}$	2	4	8	$\{u,w,x\}$	4	−16	−64
$\{u,w\}$	2	4	8	$\{v,w,x\}$	5	−25	−125
$\{u,x\}$	2	4	8	V	6	36	216

Fig. 4: Algorithm M on a small graph, with $a_q(W) = (-1)^{|V\setminus W|}(g(W))^q$. The sum of the a_2 is 0, so there is no 2-colouring. The sum of the a_3 is 18, so a 3-colouring exists.

This leads to the following algorithm [3]:

Algorithm I (*Inclusion–exclusion*). *Given a graph G and integer q, determines if G can be q-coloured.*

I1 [Tabulate g.] Set $g(\emptyset) = 0$. For every nonempty subset $W \subseteq V$ in inclusion order, pick $v \in W$ and set $g(W) = g(W \setminus \{v\}) + g(W \setminus N[v]) + 1$.

I2 [Evaluate sum.] If $\sum_{W\subseteq V}(-1)^{|V\setminus W|}(g(W))^q > 0$ return 'yes,' otherwise 'no.' ∎

Both steps I1 and I2 take time 2^n poly(n), and the algorithm requires a table with 2^n entries.

With slight modifications, algorithm I can be made to work for other colouring problems such as the chromatic polynomial and list colouring, also in time and space 2^n poly(n) [3]. Currently, this is the fastest known algorithm for these problems. For the chromatic polynomial, the space requirement can be reduced to $O(1.292^n)$ while keeping the 2^n poly(n) running time [4].

5. Local augmentation

Sometimes, a nonoptimal colouring can be improved by a local change that recolours some vertices. This general idea is the basis of many local search heuristics but also several central theorems.

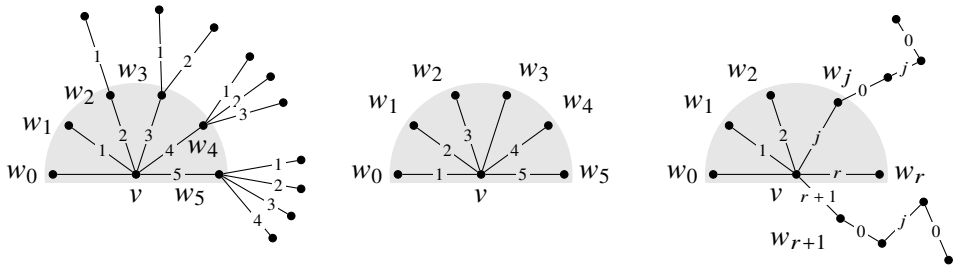


Fig. 5: (a) A fan. (b) Downshifting from 3. (c) Step V7. Colour j is free at w_{r+1}.

Kempe changes

An important example, for edge colouring, establishes Vizing's theorem, $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$.

A colour is *free* at v if it does not appear on v's incident edges. (We will consider an edge colouring with $\Delta + 1$ colours, so every vertex has at least one free colour.) A *fan* around v is a maximal set of edges $vw_0, \ldots, vw_r$, where vw_0 is not yet coloured and the other edges are coloured as follows. For $j = 0, \ldots, r$, no colour is free at both v and w_j. For $j = 1, \ldots, r$, the jth fan edge vw_j has colour j and the colours appearing around w_j include $1, \ldots, j$ but not $j + 1$. Such a fan allows a recolouring by moving colours as follows: remove the colour from vw_j and set $f(vw_0) = 1, \ldots, f(vw_{j-1}) = j$. This is called *downshifting from* j.

Algorithm V (*Vizing's algorithm*). *Given a graph G, finds an edge colouring with at most $\Delta(G) + 1$ colours in time $O(nm)$.*

V1. [Initialize.] Order the edges arbitrarily $e_1, \ldots, e_m$. Let $i = 0$.

V2. [Extend colouring to next edge.] Increment i. Let $vw = e_i$.

V3. [Easy case.] If there is a colour c that is free at both v and w then set $f(vw) = c$ and go back to V2.

V4. [Find w_0 and w_1.] Let $w_0 = w$. Pick a free colour at w_0 and call it 1. Let vw_1 be the edge incident on v coloured 1. (Such an edge exists because 1 is not also free at v.)

V5. [Find w_2.] Pick a free colour at w_1 and call it 2. If 2 is also free at v, then set $f(vw_0) = 1$, $f(cw_1) = 2$ and return to V2. Otherwise, let vw_2 be the edge incident on v coloured 2. Set $r = 2$.

V6. [Extend fan to w_{r+1}.] Pick a free colour at w_r and call it $r + 1$. If $r + 1$ is also free at v then downshift from r, recolour $f(vw_r) = w_{r+1}$ and return to V2.

Otherwise, let vw_{r+1} be the edge incident on v coloured $r+1$. If every colour $1, \ldots, r$ appears around w_{r+1} then increment r and repeat V6.

V7. [Build $\{0, j\}$-path from w_j or from w_{r+1}.] Let $j \in \{1, \ldots, r\}$ be a free colour at w_{r+1} and let $0 \neq j$ be a colour free at v. Construct two maximal $\{0, j\}$-coloured paths P_j and P_{r+1} from w_j and from w_{r+1}, respectively, by following edges of alternating colours $0, j, \ldots$. (They cannot both end in v.) Choose $k \in \{j, r+1\}$ such that P_k does not end in v.

V8. [Flip colours on P_k.] Recolour the edges on P_k by exchanging 0 and j. Downshift from k, recolour $f(vw_k) = 0$, and return to V2. ∎

To see that this algorithm is indeed correct, one needs to check that the recolourings in steps V6 and V8 are legal. A careful analysis can be found in [31].

For the running time, first note that step V6 is repeated at most $\deg v$ times, so the algorithm eventually has to leave that step. The most time-consuming step is V7; a $\{0, j\}$-path can be constructed in time $O(n)$ if we for every vertex maintain a table of incident edges indexed by colour. Thus the total running time of algorithm V is $O(mn)$.

Another example, the proof of Brooks's theorem [8] relies on an algorithm that follows algorithm G but attempts to re-colour the vertices of bichromatic components whenever a fresh colour is about to be introduced.

Random changes

There are many other graph colouring algorithms that fall under the umbrella of local transformations. Of particular interest are local search algorithms that recolour individual vertices at random. This idea defines a random process on the set of colourings called the Glauber or Metropolis dynamics, or the natural Markov chain Monte Carlo method. The aim here is not to find a colouring (since we will have $q > 4\Delta$ this would be easily done by algorithm G), but to find a colouring that is uniformly distributed among all q-colourings.

Algorithm M (*Metropolis*). *Given a graph G with maximum degree Δ and a q-colouring f_0 for $q > 4\Delta$, finds a uniform random q-colouring in polynomial time.*

M1. [Outer loop.] Set $T = \lceil (q - 4\Delta)^{-1} qn \ln 2n \rceil$. Do step M2 for $t = 1, \ldots, T$. Return f_T.

M2. [Recolour random vertex.] Pick vertex $v \in V$ and colour $c \in \{0, \ldots, q-1\}$ uniformly at random. Set $f_t = f_{t-1}$. If c does not appear among v's neighbours, then set $f_t(v) = c$. ∎

An initial colouring can be provided in polynomial time because $q > \Delta + 1$, *e.g.*, by algorithm G. To see that the choice of initial colouring f_0 has no influence on the result f_T, consider two different initial colourings f_0 and f_0' and execute algorithm M on both, using the same random choices for v and c in each step.

Let $d_t = \#\{\, v \in V : f_t(v) \neq f_t'(v) \,\}$ denote the number of *disagreeing* vertices after t steps M2. Every step can change only a single vertex, so $|d_t - d_{t-1}| \in \{-1, 0, +1\}$. We have $d_t = d_{t-1} + 1$ only if $f_{t-1}(v) = f_{t-1}'(v)$ but $f_t(v) \neq f_t'(v)$, so exactly one of the two processes rejected the colour change. In particular, v must have a (disagreeing) neighbour w with $c = f_{t-1}(w) \neq f_{t-1}'(w)$ or $f_{t-1}(w) \neq f_{t-1}'(w) = v$. There are d_{t-1} choices for such a w and therefore $2\Delta d_{t-1}$ choices for picking c and v. Similarly, we have $d_t = d_{t-1} - 1$ only if $f_{t-1}(v) \neq f_{t-1}(v)$ and c does not appear in v's neighbourhood in either f_{t-1} or f_{t-1}'. There are at least $(q - 2\Delta)d_{t-1}$ such choices of c and v.

Thus,

$$\mathbf{E}[d_t] \leq \mathbf{E}[d_{t-1}] + \frac{(q-2\Delta)\mathbf{E}[d_{t-1}]}{qn} - \frac{2\Delta\mathbf{E}[d_{t-1}]}{qn} = \mathbf{E}[d_{t-1}]\Big(1 - \frac{q-4\Delta}{qn}\Big).$$

Iterating this argument and using $d_0 \leq n$, we have

$$\mathbf{E}[d_T] \leq n\Big(1 - \frac{q-4\Delta}{qn}\Big)^T \leq n\exp\Big(-\frac{T(q-4\Delta)}{qn}\Big) \leq n\exp(-\ln 2n) = \tfrac{1}{2}\,.$$

By Markov's inequality, and because d_T is a nonnegative integer, we conclude $\Pr(f_T = f_T') = \Pr(d_T = 0) \geq 1 - \Pr(d_T \geq 1) \geq 1 - \mathbf{E}[d_T] \geq \frac{1}{2}$.

We will contend ourselves with this argument, which shows that the process is 'sufficiently random' in the sense of being memoryless. Informally, we can convince ourselves that f_T is uniformly distributed because we can assume that f_0' in the above argument was sampled according to such a distribution. This inution can be formalised using standard coupling arguments for Markov chains; our calculations above show that the 'mixing time' of algorithm M is $O(n \log n)$.

Algorithm M and its variants have been very well studied, and the analysis can be much improved (see [14] for a survey). Randomized local search has wide appeal across disciplines, including simulations in statistical physics and heuristic methods in combinatorial optimisation.

6. Vector colouring

We turn to a variant of vertex colouring that is particularly interesting from an algorithmic point of view.

Let $S^{d-1} = \{\, \mathbf{x} \in \mathbf{R}^d \colon \|\mathbf{x}\| = 1 \,\}$. A *vector q-colouring* in $d \le n$ dimensions is a mapping $x \colon V \to S^{d-1}$ from the vertices to d-dimensional unit vectors such that neighbouring vectors are 'far apart' in the sense that their scalar product satisfies

$$\langle x(v), x(w) \rangle \le -\frac{1}{q-1}\,, \qquad (vw \in E)\,.$$

The smallest such q is called the *vector chromatic number* $\vec{\chi}(G)$, it need not be an integer. For instance, the vertices of the 3-colourable C_5 can be laid out on the unit circle in the form of a pentagram  . The angles between vectors corresponding to neighbouring vertices are $-1/(\sqrt{5}-1)$ so $\vec{\chi}(C_5) \le \sqrt{5} < 3$.

Lemma 6. *Let $\omega(G)$ denote the size of the largest clique in G. Then, $\omega(G) \le \vec{\chi}(G) \le \chi(G)$.*

Proof. For the first inequality, let W be a clique in G of size $r = \omega(G)$ and consider a vector q-colouring x of G. Let $\mathbf{y} = \sum_{v \in W} x(v)$. Then

$$0 \le \langle \mathbf{y}, \mathbf{y} \rangle \le r \cdot 1 + r(r-1) \cdot \left(-\frac{1}{q-1} \right),$$

which implies $r \le q$. For the second inequality, we place the vertices belonging to each colour class at the corners of a $(q-1)$-dimensional simplex. To be concrete, let $f \colon V \to \{0, \dots, q-1\}$ be an optimal q-colouring. Define $x(v) = (x_1, \dots, x_n)$ by

$$x_i = \begin{cases} ((q-1)/q)^{1/2}\,, & \text{if } i = f(v) - 1\,; \\ -(q(q-1))^{-1/2}\,, & \text{if } i \ne f(v) - 1 \text{ and } i < q\,; \\ 0\,, & \text{if } i \ge q\,. \end{cases}$$

Then we have $\langle x(v), x(v) \rangle = 1 \cdot (q-1)/q + (q-1) \cdot (q(q-1))^{-1} = 1$ and for v and w with $f(v) \ne f(w)$ we have $\langle x(v), x(w) \rangle = 2 \cdot ((q-1)/q)^{1/2}(-(q(q-1))^{1/2}) + (q-2) \cdot (q(q-1))^{-1} = -(q-1)^{-1}$. Thus, x is a vector q-colouring, so $\vec{\chi}(G) \le q$. ∎

What makes vector colourings interesting for us is that they can be found in polynomial time, at least approximately. The details behind this algorithm lie far outside the scope of this presentation, see [22].

Algorithm S (*Semidefinite programming*). *Given a graph G with $\vec{\chi}(G) = q$, finds a vector $(q+\epsilon)$-colouring in in time polynomial in n and* $\log(1/\epsilon)$. ∎

For a graph with $\omega(G) = \chi(G)$, lemma 6 shows that the vector chromatic number equals the chromatic number. In particular, it is an integer and can be determined using algorithm S with $\epsilon < \frac{1}{2}$. This shows that the chromatic number

of perfect graphs can be determined in polynomial time. The theory behind this result counts as one of the highlights of combinatorial optimization, see [17].

How does the vector chromatic number behave for general graphs? For $q = 2$, the vectors have to point in exactly opposite directions. In particular, there can be only 2 vectors for every connected component, so vector 2-colouring is equivalent to 2-colouring.

But already $q = 3$, the situation becomes more interesting. There exist vector 3-colourable graphs that are not 3-colourable. For instance, the Grötzsch graph, the smallest triangle-free graph with chromatic number 4, admits the vector 3-colouring shown in figure 6. More complicated constructions (that we cannot visualize) show that there exist vector 3-colourable graphs with chromatic number at least $n^{0.157}$ (see [13] and [22]).

Even though the gap between $\vec{\chi}$ and χ can be large for general graphs, vector colouring turns out to be a useful starting point for (standard) colouring. The next algorithm translates a vector colouring into a colouring using random hyperplanes.

Algorithm R (*Randomized rounding of vector colouring*). *Given a 3-colourable graph G with maximum degree Δ, computes a q-colouring in polynomial time, where* $\mathbf{E}[q] = O(\Delta^{0.681} \log n)$.

R1 [Vector colour.] Set $\epsilon = 2 \cdot 10^{-5}$ and compute a vector $(3+\epsilon)$-colouring x of G using algorithm S. Let $\alpha \geq \arccos(-1/(2+\epsilon))$ denote the minimum angle in radians between neighbouring vertices.

R2 [Round.] Set

$$r = \lceil \log_{\pi/(\pi-\alpha)}(2\Delta) \rceil$$

and construct r random hyperplanes $H_1, \ldots, H_r$ in R^n. For each vertex v set $f(v)$ to the binary number $b_r \cdots b_1$ where $b_i = 1$ if $x(v)$ is on the positive side of the ith hyperplane H_i.

R3 [Recurse on monochromatic edges.] Iterate over all edges to find the set of monochromatic edges $M = \{\, vw \in E : f(v) = v(w) \,\}$. Recolour these vertices by running algorithm R recursively on $G[M]$. ∎

The algorithm runs in polynomial time because of lemma **??**. We proceed to analyse the size of the final colouring. Let e be an edge whose endpoints received the vector colours $\mathbf{x}$ and $\mathbf{y}$, respectively. Elementary geometric considerations establish the following:

Lemma 7. *Let $\mathbf{x}, \mathbf{y} \in \mathbf{R}^d$ with angle ϕ in radians. A random hyperplane in $\mathbf{R}^d$ fails to separate $\mathbf{x}$ and $\mathbf{y}$ with probability $1 - \phi/\pi$.*

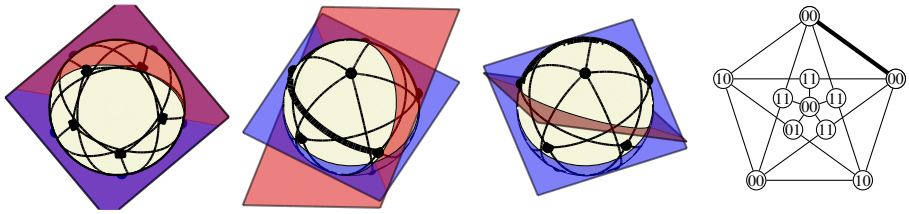


Fig. 6: A vector 3-colouring of the Grötzsch graph visualised as an embedding on the unit sphere. Two hyperplanes separate the vertices into 4 parts. The corresponding vertex colouring with colours from $\{0,1\}^2$ is shown to the right, using a more familiar embedding of the Grötzsch graph in the plane. In this example, the set M of monochromatic edges contains only a single edge, drawn bold.

The vectors $\mathbf{x}$ and $\mathbf{y}$ have angle at most α. (To gain some intuition, if we ignore the error term ϵ, the above lemma shows that $\mathbf{x}$ and $\mathbf{y}$ end up on the same side of a random hyperplane is $1 - \alpha/\pi \leq 1 - \arccos(-\frac{1}{2})/\pi = 1 - 2\pi/3\pi = \frac{1}{3}$.) Edge e is monochromatic if all r independent random hyperplanes $H_1, \ldots, H_r$ fail to separate $\mathbf{x}$ and $\mathbf{y}$ in step R2. Thus,

$$\Pr(e \in M) \leq (1 - \alpha/\pi)^r \leq (\pi/(\pi - \alpha))^{-r} \leq \frac{1}{2\Delta}\,.$$

By linearity of expectation,

$$\mathbf{E}[\#M] = \sum_{e \in E} \Pr(e \in M) \leq \frac{m}{2\Delta} \leq \frac{n}{4}\,.$$

Since every edge is incident on two vertices, the expected number of vertices in the recursive instance $G[M]$ is at most $\frac{1}{2}n$. In general, the expected number of vertices n_i in the ith instance satisfies $n_i \leq \frac{1}{2}n_{i+1}$. In particular, $n_t \leq 1$ after $t = O(\log n)$ rounds, at which point the algorithm terminates. The number of colours expended in every recursive call is

$$2^r \leq (2\Delta)^{-1/\log(\pi/(\pi-\alpha))} < (2\Delta)^{0.631}\,.$$

In terms of Δ, algorithm R is much better than the $\Delta + 1$ bound guaranteed by algorithm G. For an expression in terms of n, we are tempted to bound $2\Delta = O(n)$, but that just shows that the number of colours is $O(n^{0.631} \log n)$, which is worse than the $O(\sqrt{n})$ colours from algorithm W. Instead, we employ a hybrid approach:

Name		Time	What
B	Bipartition	$O(n+m)$	2-colouring
C	Contraction	$O(1.619^n)$	$P(G,q)$
D	Dynamic programming	$3^n \operatorname{poly}(n)$	$\chi(G)$
G	Greedy	$O(n+m)$	$(\Delta(G)+1)$-colouring
I	Inclusion–exclusion	$2^n \operatorname{poly}(n)$	$\chi(G)$
L	Lawler's algorithm	$O(1.45^n)$	3-colouring
M	Metropolis dynamics	$\operatorname{poly}(n)$	Random q-colouring ($q > 4\Delta$)
R	Rounded vector colouring	$\operatorname{poly}(n)$	$O(\Delta^{0.681} \log n)$-colouring for $\chi(G) = 3$
S	Semidefinite programming	$\operatorname{poly}(n)$	vector $\vec{\chi}(G)$-colouring
V	Vizing's algorithm	$O(mn)$	edge $(\Delta(G)+1)$-colouring
W	Wigderson's algorithm	$O(n+m)$	$O(\sqrt{n})$-colouring for $\chi(G) = 3$
X	Exhaustive search	$q^n \operatorname{poly}(n)$	$P(G,q)$

Fig. 7: Overview of algorithms

Run steps W1 and W2 as long as the maximum degree of G is larger than some threshold d. Then, colour the remaining graph with algorithm R. The number of colours used by the combined algorithm is of the order

$$\frac{2n}{d} + (2d)^{0.631} \log n$$

which is minimized around $d = n^{1/1.631}$ with value $O(n^{0.387})$.

Variants of algorithm R for general q-colouring and with intricate rounding schemes have been investigated a lot further, see [25] for a survey. The current best polynomial-time algorithm for colouring a 3-colourable graph, which is based on vector colouring, uses $O(n^{0.22})$ colours [1].

7. Reductions

As shown in figure 7, various graph colouring algorithms differ in their running times, quality guarantees, and which problem they solve.

1. (*Decision*) Given G and q, decide if q can be q-coloured.
2. (*Chromatic number*) Given G, find $\chi(G)$.
3. (*Construction*) Given G and q, construct a colouring
4. (*Counting*) Given G and q, count the number of q-colourings

5. (*Sampling*) Given G and q, construct a random q-colouring.

6. (*Chromatic polynomial*) Given G, compute of the chromatic polynomial, i.e., the coefficients of the integer polynomial $q \mapsto P(G, q)$.

Some of these problems are related using fairly straightforward reductions: The decision problem is easily solved by comparing q to $\chi(G)$; conversely, χ can be determined by solving the decision problem for $q = 1, 2, \ldots$. It is also clear that if we can construct a q-colouring then we can decide that one exists. What is maybe less clear is the other direction. This is seen by a self-reduction that follows the contraction algorithm C.

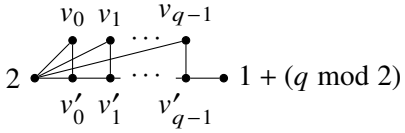
Reduction C (*Construction using decision*). Assume that we have an algorithm that decides if a given graph G can be q-coloured. If $G = K_n$ and $n \leq q$ then give each vertex its own colour and terminate. Otherwise, pick two nonadjacent vertices u, v in G. If $G + vw$ cannot be q-coloured then every q-colouring f of G must have $f(v) = f(w)$. Thus we can identify u and v and recursively find a q-colouring for G/vw. Otherwise, there exists a q-colouring of G with $f(u) \neq f(v)$, so we recurse on $G + vw$. ∎

Some of our algorithms work only for specific, fixed q, such as algorithm B for 2-colourability or algorithm L for 3-colourability. Obviously, they both reduce to the decision problem where q is part of the input. But what about the other direction? The answer turns out to depend strongly on q: The decision problem reduces to 3-colorability, but not to 2-colorability.

Reduction L (*q-colouring using 3-colouring*). Given a graph $G = (V, E)$ and integer q, constructs a graph H that is 3-colourable if and only if G is q-colourable.

First, to fix some colour names, the graph H contains a triangle on the vertices $0, 1, 2$. We assume that vertex i has colour i for $i = 0, 1, 2$.

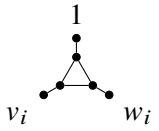
For every vertex $v \in V$, the graph H contains $2q$ vertices $v_0, \ldots, v_{q-1}$ and $v'_0, \ldots, v'_{q-1}$. The intuition is that the v_is act as indicators for a colour in G in the following sense: If v_i has colour 1 in H then vertex v received colour i in G. The vertices are arranged like this:



Here, the rightmost vertex is 1 or 2, depending on the parity of q. The vertices $v_0, \ldots, v_{q-1}$, all being adjacent to 2, must be coloured 0 or 1. Moreover, at least

one of them must be coloured 1. (Otherwise, no valid colouring for $v'_0, \ldots, v'_{q-1}$ remains.)

Now consider an edge vw in G. Let $v_0, \ldots, v_{q-1}$ and $w_1, \ldots, w_{q-1}$ denote the corresponding 'indicator' vertices in H. For every colour $i = 0, \ldots, q-1$, the vertices v_i and w_i are connected using a 'fresh' triangle like this:

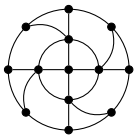


This ensures that v_i and w_i cannot both be 1. ∎

The above reduction, essentially due to Lovász [27], is easily extended to larger, fixed $q > 3$ because G is q-colourable if and only if G with an added 'apex' vertex (adjacent to all other vertices) is $(q+1)$-colourable.

Thus, all q-colouring problems for $q \geq 3$ are in some sense equally difficult. This is consistent with the fact that case $q = 2$ admits a very fast algorithm (B), while none of the others does.

Many constructions have been published that show the computational hardness of colouring for restricted classes of graphs. An interesting example is the restriction of the $q = 3$ case to planar graphs [37]. Draw G in the plane and replace every edge intersection by this subgraph:



This subgraph has the property that in any 3-colouring the top- and bottommost vertices have the same colour, and the left- and rightmost vertices have the same colour. Unlike for nonplanar graphs, this construction cannot be generalized to larger $q > 3$, since the decision problem for planar graphs and every $q \geq 4$ is trivial: The answer is 'yes' because of the 4 Colour Theorem.

Computational complexity

The field of computational complexity relates algorithmic problems from other domains to each other in order to establish a notion of computational hardness. The

chromatic number problem was one of the first problems analysed in this fashion; the reduction (published with minor inaccuracies in [**?**]) is from the CNF-Satisfiability problem.

The input to CNF-Satisfiability is a Boolean formula consisting of m clauses $C_1, \ldots, C_s$. Every clauses C_j consists of a disjunction $C_j = (l_{j1} \vee \ldots \vee l_{jk})$ of literals. Every literal is a variable $x_1, \ldots, x_r$ or its negation $\overline{x_1}, \ldots, \overline{x_r}$. The problem is to find an assignment of the variables to 'true' and 'false' that make all clauses 'true.'

Reduction K (*Satisfiability using chromatic number*). Given an instance $C_1, \ldots, C_s$ to CNF-Satisfiability over the variables $x_1, \ldots, x_r$, constructs a graph G on $1+3r+s$ vertices such that G can be coloured with $r+1$ colours if and only if ϕ is satisfiable.

The graph G contains a complete subgraph on $r+1$ vertices $\{0, 1, \ldots, r\}$. In any colouring, these vertices receive different colours, say $f(i) = i$. The intuition is that the colour 0 represents 'false,' while the other colours represent 'true.' For every variable x_i $(1 \leq i \leq r)$ the graph contains two 'literal' vertices v_i and $\overline{v_i}$. These vertices are adjacent to each other and to all vertices j for $j \neq i$. Thus, one of the two vertices $v_i, \overline{v_i}$ must be assigned the 'true' colour i, the other must be coloured 0. The construction is completed with 'clause' vertices w_j, one for every clause C_j $(1 \leq j \leq s)$. The vertex w_j is adjacent to 0 and every literal vertex, except for those corresponding to literals appearing in C_j. Thus, the only colours from $\{1, \ldots, r\}$ available to C_j are those chosen by its literals. ∎

This reduction shows that computing the chromatic number is hard for the complexity class NP. Similar arguments show that under the stronger exponential time hypothesis ETH, chromatic number requires time $\exp(\Omega(n))$ and that the counting variant (chromatic polynomial) is hard for the class #P.

Edge colouring. The *line graph* $L(G)$ of $G = (V, E)$ is the graph whose vertices are the edges of G and where e_1e_2 is an edge in $L(G)$ if e_1 and e_2 are incident on the same vertex in G. A mapping $f : E \to \{0, \ldots, q\}$ is a vertex colouring of $L(G)$ if and only if it is an edge colouring of G. In particular, every vertex colouring algorithm can be used as an edge colouring algorithm by running it on $L(G)$. For instance, algorithm I computes the chromatic index in time $2^m \operatorname{poly}(n)$, which is the fastest currently known algorithm. Similarly, algorithm G computes a $(2\Delta - 1)$-edge colouring, but this is worse than algorithm V. In fact, since $\Delta \leq \chi'(G) \leq \Delta+1$, algorithm V determines the chromatic index within an additive error of 1. However, deciding which of the two candidate values for $\chi'(G)$ is correct is an NP-hard problem (see [19] for $\chi'(G) = 3$ and [28] for $\chi'(G) > 3$).

Approximating the chromatic number. Algorithm V shows that the chromatic index can be very well approximated. In contrast, approximate *vertex* colouring is much harder. In particular, it is NP-hard to 4-colour a 3-colourable graph [18]. This rules out an algorithm as good as algorithm V, but is far from explaining why the considerable firepower behind, say, algorithm R only results in colouring of size n^c for 3-colourable graphs. The best current exponent is $c = 0.204$ [**?**]. For large enough fixed q, it is NP-hard to find a $\exp(\Omega(q^{1/3}))$-colouring for a q-colourable graph.

If q is not fixed, even stronger hardness results are known. We saw in section 6 that the polynomial-time computable function $\vec{\chi}(G)$ is a lower bound on $\chi(G)$, even though the gap can sometimes be large, say $\chi(G) \geq n^{0.157}\vec{\chi}(G)$ for some graphs. Can we guarantee a corresponding upper bound for $\vec{\chi}$? If not for $\vec{\chi}$, maybe there is some other polynomial-time computable function g so that we would guarantee, *e.g.*, $g(G) \leq \chi(G) \leq n^{0.999}g(G)$? The answer turns out to be 'no' under standard complexity-theoretic assumptions: For every $\epsilon > 0$, it is NP-hard to approximate $\chi(G)$ within a factor $n^{1-\epsilon}$ [39].

Counting. The counting problem is solved by evaluating $P(G, q)$. Conversely, because the chromatic polynomial has degree n it can be interpolated using Lagrangian interpolation from the values of the counting problem at $q = 0, \ldots, n$. And of course $\chi(G) \geq q$ if and only if $P(G, q) > 0$, so it is NP-hard to count the number of q-colourings simply because the decision problem is known to be hard. In fact, the counting problem is hard for Valiant's counting class #P.

On the other hand, an important result in counting complexity [**?**] relates the estimation of the size of a finite set to the problem of uniformly samling from it. In particular, a uniform sampler such as algorithm M serves as a 'fully polynomial randomized approximation scheme' (FPRAS) for the number of colours. Thus, provided that $q > 4\Delta$, algorithm M can be used to compute a value $g(G)$ such that $(1 - \epsilon)g(G) \leq P(G, q) \leq (1 + \epsilon)g(G)$ with high probability in time polynomial in n and $1/\epsilon$ for any $\epsilon > 0$. Much better bounds on q are known [14]. Without some bound on q, such an FPRAS is unlikely to exist because, setting $\epsilon = \frac{1}{2}$, it would constitute a randomised algorithm for the decision problem and therefore imply NP=RP.

Bibliography

1. S. Arora, E. Chlamtac, New approximation guarantee for chromatic number, *Proc. 38th STOC* (2006), 215–224.

2. A. Björklund, T. Husfeldt, Exact algorithms for exact satisfiability and number of perfect matchings. *Algorithmica* **52** (2008) 226–249.
3. A. Björklund, T. Husfeldt, M. Koivisto, Set partitioning via inclusion–exclusion, *SIAM J. Comput.* **39** (2009), 546–563.
4. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto, Covering and packing in linear space, *Inf. Process. Lett.* **111** (2011), 1033–1036.
5. A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto, Computing the Tutte polynomial in vertex-exponential time, *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS) 2008.*
6. R. Beigel, D. Eppstein, 3-coloring in time $O(1.3289^n)$, *J. Algorithms* **111** (2005), 168–204.
7. D. Brélaz, New methods to color the vertices of a graph, *Comm. ACM* **22** (1979), 251–256.
8. R. L. Brooks, On colouring the nodes of a network, *Proc. Cam. Phil. Soc., Math. Phys. Sci.* **37** (1941) 194–197.
9. C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, *C*ommun. ACM **16(9)** (1973) 575–577.
10. J. M. Byskov, Enumerating maximal independent sets with applications to graph colouring, *Oper. Res. Lett.* **32** (2004), 547–556.
11. N. Christofides, An algorithm for the chromatic number of a graph. *Comput. J.* **14(1)** (1971), 38–39.
12. V. Chvátal, Perfectly ordered graphs, in *Topics on Perfect Graphs* (eds. C. Berge and V. Chvátal), *Ann. Disc. Math.* **21** (1984), 63–68.
13. U. Feige, M. Langberg, G. Schechtman, Graphs with tiny vector chromatic numbers and huge chromatic numbers, *SIAM J. Comput.* **33** (2004), 1338–1368.
14. A. Frieze, E. Vigoda, A survey on the use of Markov chains to randomly sample colorings, in *Combinatorics, Complexity and Chance*, Oxford University Press, 2007.
15. B. Gärtner, J. Matoušek, *Approximation algorithms and semidefinite programming*, Springer, 2012.
16. G. Grimmett, C. McDiarmid, On colouring random graphs, *Math. Proc. Cam. Phil. Soc.* **77** (1975), 313–324.
17. L. Lovász, M. Grötschel, A. Schrijver, *Geometric algorithms and combinatorial optimization*, Springer, 1988.
18. V. Guruswami, S. Khanna, On the hardness of 4-coloring a 3-colorable graph, *Proceedings of the 15th Annual IEEE Conference on Computational Complexity* (2000) 188–197.
19. I. Holyer, The NP-completeness of edge-coloring, *SIAM J. Comput.* **10** (1981) 718–720.
20. S. Huang, Improved hardness of approximating chromatic number, *16th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems* (2013).
21. B. Jackson, Chromatic Polynomials, *this volume*.

22. D. R. Karger, R. Motwani, M. Sudan, Approximate graph coloring by semidefinite programming, *J. Assoc. Comput. Mach.* **45** (1998), 246–265.
23. A. Kosowski, K. Manuszewski, Classical coloring of graphs, In *Graph Colorings* (ed. M. Kubale), *AMS Contemporary Math.* **352** (2004), 1–20.
24. L. Kučera, The greedy coloring is a bad probabilistic algorithm, *J. Algorithms* **12** (1991), 674–684.
25. M. Langberg, Graph coloring, in *Encyclopedia of Algorithms* (ed. M. Kao), Springer (2008).
26. E. L. Lawler, A note on the complexity of the chromatic number problem, *Inf. Process. Lett.* **5(3)** (1976), 66–67.
27. L. Lovász, Coverings and coloring of hypergraphs, *Proceedings of the Fourth Southeastern Conference on Combinatorics, Graph Theory, and Computing*, Utilitas Math., Winnipeg, Man., 1973, pp. 3–12.
28. D. Leven and Z. Galil, NP completeness of finding the chromatic index of regular graphs, *J. Algorithms* **4** (1983), 35–44.
29. F. Maffray, On the coloration of perfect graphs.
30. D. W. Matula, A min–max theorem for graphs with applications to graph coloring, *SIAM Rev.* **10** (1968), 481–482.
31. J. Misra, D. Gries , A constructive proof of Vizing's Theorem, *Inf. Proc. Let.* **41** (1992) 131–133.
32. J. W. Moon, L. Moser, On cliques in graphs, *Israel J. Math.* **3** (1965) 23–28.
33. G. Szekeres, H. Wilf, An inequality for the chromatic number of a graph, *J. Comb. Theory* **4** (1968) 1–3.
34. D. J. Welsh, M. B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, *Comput. J.* **10** (1967), 85–86.
35. A. Wigderson, Improving the performance guarantee for approximate graph coloring, *J. Assoc. Comput. Mach.* **30** (1983), 29–735.
36. H. Whitney, A logical expansion in mathematics, *Bull. Amer. Math. Soc.* **38** (1932), 572–579.
37. L. Stockmeyer, Planar 3-colorability is polynomial complete, *ACM SIGACT News* **5** (3), (1973), 19–25.
38. E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments, *Theor. Comput. Sci.* 363 (1) (2006) 28–42.
39. D. Zuckerman, Linear degree extractors and the inapproximability of Max Clique and Chromatic Number, *Theory of Computing* **3** (2007) 103–128.
40. A. A. Zykov, О некоторых свойствах линейных комплексов (On some properties of linear complexes, in Russian), *Math. Sbornik.* **24** (1949) 163–188.