13

Graph colouring algorithms

THORE HUSFELDT

This chapter presents an introduction to graph colouring algorithms.[1] *The focus is on vertex-colouring algorithms that work for general classes of graphs with worst-case performance guarantees in a sequential model of computation. The presentation aims to demonstrate the breadth of available techniques and is organized by algorithmic paradigm.*

1. Introduction

A straightforward algorithm for finding a vertex-colouring of a graph is to search systematically among all mappings from the set of vertices to the set of colours, a technique often called *exhaustive* or *brute force*:

Algorithm X (*Exhaustive search*) *Given an integer $q \geq 1$ and a graph G with vertex set V, this algorithm finds a vertex-colouring using q colours if one exists.*

X1 [Main loop] For each mapping $f : V \to \{1, 2, \ldots, q\}$, do Step X2.

X2 [Check f] If every edge vw satisfies $f(v) \neq f(w)$, terminate with f as the result. ∎

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

[1]Appears as Thore Husfeldt, *Graph colouring algorithms*, Chapter XIII of *Topics in Chromatic Graph Theory*, L. W. Beineke and Robin J. Wilson (eds.), Encyclopedia of Mathematics and its Applications, Cambridge University Press, ISBN 978-1-107-03350-4, 2015, pp. 277–303.

Model of computation

If G has n vertices and m edges, then the number of operations used by Algorithm X can be asymptotically bounded by $O(q^n(n+m))$, which we call 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 indexed by vertices; the sequence stored at vertex v contains the neighouring vertices $N(v)$, see Fig. 1. This representation allows us to iterate

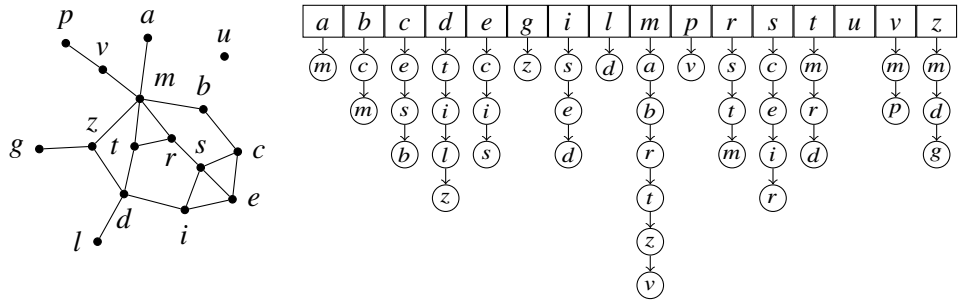


Fig. 1. A graph and its representation as an array of sequences

over the neighbours of a vertex in time $O(\deg v)$. (An alternative representation, such as an incidence or adjacency matrix, would not allow this.) Note that detecting whether two graphs are isomorphic is *not* a primitive operation. The convention of expressing computational resources using asymptotic 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

Note that we could have fixed the colouring of a specific vertex v as $f(v) = 0$, reducing Algorithm X's running time to $O(q^{n-1}(n+m))$. A moment's thought shows that this reasoning can then 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 $\{1,2,\ldots,r\}$ and then 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 useful 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 some other graph colouring problems. For instance, to find a list-colouring, we restrict the range of values for each $f(v)$ to a given list; to find an edge-colouring, we iterate over all mappings $f : E \to \{1, 2, \ldots, q\}$.

Another modification is to count the number of colourings instead of finding just one. These extensions provide baseline algorithms for list-colouring, edge-colouring, the chromatic polynomial, the chromatic index, and so forth. However, for purposes of exposition, we present algorithms in their *least* general form, emphasizing the algorithmic idea rather than its (sometimes quite pedestrian) generalizations. The algorithms are organized 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 perhaps for Algorithm G in Section 2. The final section takes a step back and relates the various colouring problems to each other.

2. Greedy colouring

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

Algorithm G (*Greedy vertex-colouring*) *Given a graph G with maximum degree Δ and an ordering $v_1, v_2, \ldots, v_n$ of its vertices, this algorithm finds a vertex-colouring with $\max_i |\{ j < i : v_j v_i \in E \}| + 1 \le \Delta + 1$ colours.*

G1 [Initialize] Set $i = 0$.

G2 [Next vertex] Increment i. If $i = n + 1$, terminate with f as the result.

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

G4 [Assign the smallest available colour to v_i] For increasing $c = 1, 2, \ldots$, check whether $c \in C$. If not, set $f(v_i) = c$ and return to Step G2. ∎

For the number of colours, it is clear that in Step G4, the value of c is at most $|C|$, which is bounded by the number of neighbours of v_i among $v_1, v_2, \ldots, v_{i-1}$. 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 i, the total time spent in Steps G3 and G4 is asymptotically bounded by $n + (\deg v_1 + \deg v_2 + \cdots + \deg v_n) = n + 2m$. Thus, Algorithm G takes time $O(n + m)$.

Optimal ordering The size of the colouring computed by Algorithm G depends heavily on the vertex ordering. Its worst-case behaviour is poor. For instance, it spends $\frac{1}{2}n$ colours on the 2-colourable *crown graph* shown in Fig. 2.

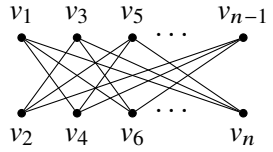


Fig. 2. The crown graph

On the other hand, for every graph there exists an ordering for which Algorithm 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 has this property. Since there are $n!$ 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 always admit such an ordering, as shown by Chvátal [11].

Randomness

Algorithm G performs quite well on random graphs, whatever the vertex ordering. For almost all n-vertex graphs, it uses $n/(\log n - 3 \log\log n)$ colours, which is roughly twice the optimum value (see [15]).

This suggests the following randomized algorithm. For a graph G, choose a vertex ordering at random and then execute Algorithm G. For many problems, it is a sound algorithmic design strategy to trade good average-case behaviour for good (expected) worst-case behaviour in this way. However, for Algorithm G the result is quite poor: for every $\varepsilon > 0$ there exist graphs with chromatic number n^{ε} for which

the randomized algorithm uses $\Omega(n/\log n)$ colours with high probability, as shown by Kučera [26].

Other orderings

In the *largest-first* vertex-degree ordering introduced by Welsh and Powell [38], the vertices are ordered such that $\deg v_1 \geq \deg v_2 \geq \cdots \geq \deg v_n$. This establishes the bound $\chi(G) \leq 1 + \max_i \min\{\deg v_i, i-1\}$, which is sometimes better than $1+\Delta$, such as in Fig. 3.

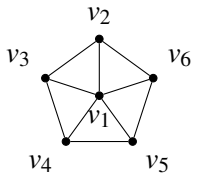


Fig. 3.

Closely related in spirit is Matula's *smallest-last* ordering [32], given as follows: choose as the last vertex v_n a vertex of minimum degree in G, and proceed recursively with $G - v_n$, see Fig. 4. With this ordering, the size of the resulting colouring is be bounded by the Szekeres–Wilf bound [36],

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

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

Fig. 4.

Other orderings are dynamic in the sense that the ordering is determined during the execution of the algorithm, rather than in advance. For example, Brélaz [6] suggests choosing the next vertex from among those adjacent to the largest number of different colours. Many other orderings have been investigated (see the surveys of Kosowski and Manuszewski [25] and Maffray [31]). Many of them perform quite well on instances that one may encounter 'in practice', but attempts at formalizing what this means are quixotic.

2-colourable graphs

Of particular interest are those vertex orderings in which every vertex v_i is adjacent to some vertex v_j with $j < i$. Such orderings can be computed in time $O(m + n)$ using basic graph-traversal algorithms. This algorithm is sufficiently important to be made explicit.

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

B1 [Initialize] Let $f(v_1) = 1$ and let Q (the 'queue') be an empty sequence. For each 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, go to Step B3. Otherwise, remove the first vertex v from Q and set $f(v)$ to the colour not already assigned to $p(v)$. For each neighbour w of v, if w is not yet coloured and does not belong to Q, then set $p(w) = v$ and add w to the end of Q. Repeat Step B2.

B3 [Verify 2-colouring] Iterate over all edges to verify that $f(v) \neq f(w)$ for every edge vw. If so, terminate with f as the result.

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

Fig. 5 shows an execution of Algorithm B finding a 2-colouring.

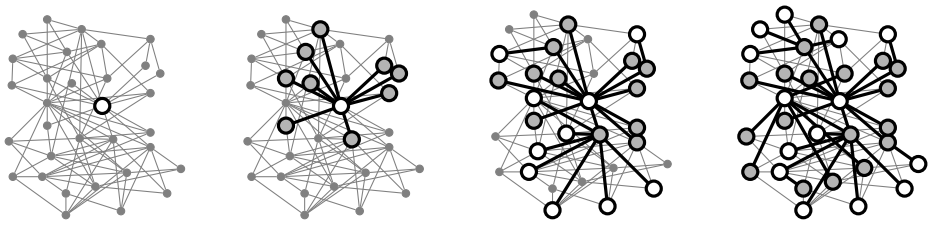


Fig. 5. Execution of Algorithm B

Algorithm B is an example of a 'certifying' algorithm: an algorithm that produces a witness to certify its correctness, in this case an odd cycle if the graph is not 2-colourable. To see that the cycle constructed in Step B4 has odd length, note that on the two paths $w, p(w), p(p(w)), \ldots, u$ and $v, p(v), p(p(v)), \ldots, u$, each vertex has a different colour from its predecessor. 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.

The order in which the vertices are considered by Algorithm B depends on the first-in first-out behaviour of the queue Q. The resulting ordering is called *breadth-first*. An important variant uses a last-in first-out 'stack' instead of a queue; the resulting ordering is called *depth-first*. Fig. 6 shows the resulting behaviour on the graph from Fig. 5.

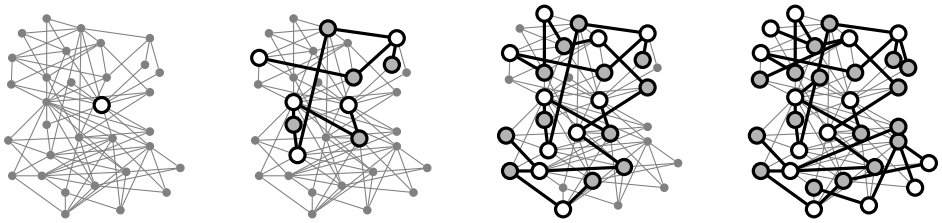


Fig. 6. Execution of Algorithm B using depth-first search

Algorithm B works also for the list-colouring problem, provided that for each vertex v, the available list of colours $L(v)$ has size at most 2. This observation leads to a simple, randomized, exponential-time algorithm for 3-colouring due to Beigel and Eppstein [1].

Algorithm P (*Palette restriction*) *Given a graph, this algorithm finds a 3-colouring if one exists.*

P1 [Forbid one colour at each vertex] For each vertex v, select a list $L(v)$ of colours available at v uniformly and independently at random from the three lists $\{1,2\}$, $\{2,3\}$, and $\{1,3\}$.

P2 [Attempt 2-colouring] Try to solve the list-colouring instance given by L using Algorithm B, setting $f(v_1) = \min L(v_1)$ in Step B1. If successful, terminate with the resulting colouring. Otherwise, return to Step P1. ∎

To analyse the running time, consider a 3-colouring f. For each vertex v, colour $f(v)$ belongs to $L(v)$ with probablity $\frac{2}{3}$. Thus, with probability at least $(\frac{2}{3})^n$, the list colouring instance constructed in step P1 has a solution. It follows that the expected number of repetitions is $(\frac{3}{2})^n$, each of which takes polynomial time.

Wigderson's algorithm

Algorithms B and G appear together in Wigderson's algorithm [40]:

Algorithm W (*Wigderson's algorithm*) *Given a* 3*-chromatic graph* G*, this algorithm finds a vertex-colouring with* $O(\sqrt{n})$ *colours.*

W1 [Initialize] Let $c = 1$.

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, go to Step W3. Use Algorithm B to 2-colour the neighbourhood $G[N(v)]$ with colours c and $c+1$. Remove $N(v)$ from G and increase c 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 $c, c+1, \ldots, c+\lceil\sqrt{n}\rceil$. ∎

Fig. 7 shows an execution of Algorithm W finding a 5-colouring of the 16-vertex instance from Fig. 1.

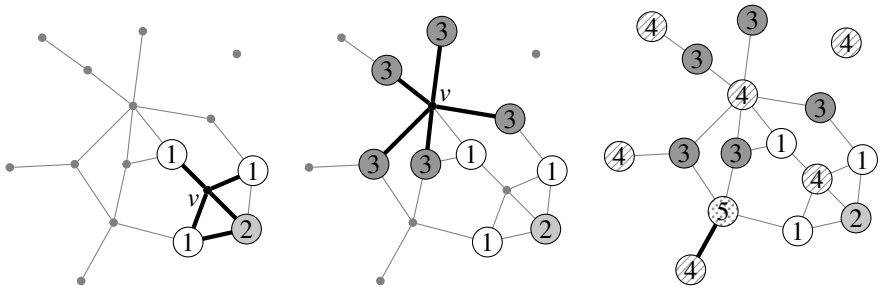


Fig. 7. Execution of Algorithm W

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

Algorithm W naturally extends to graphs with $\chi(G) > 3$. In this case, Step W2 calls Algorithm 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 the chromatic polynomial $P(G,q)$ and the chromatic number $\chi(G)$ in terms of edge-contractions: For non-adjacent vertices v, w and integer $q = 0, 1, \ldots, n$,

$$P(G,q) = P(G + vw, q) + P(G/vw, q)\,,$$
$$\chi(G) = \min\{\chi(G + vw), \chi(G/vw)\}\,,$$

see Chapter 3, Section 2.1. These 'addition–contraction' recurrences immediately imply a recursive algorithm. For instance,

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

$$= P(K_4, q) + P(K_3, q) = q(q-1)(q-2)\big((q-3)(q-4)+1\big)\,.$$

Note that the graphs at the end of the recursion are complete.

For sparse graphs, it is more useful to express the same idea as a 'deletion–contraction' recurrence, which deletes and contracts 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 in this fashion is the Tutte polynomial (see [5] and [18] for its algorithmic aspects).

The algorithm implied by these recursions is sometimes called *Zykov's algorithm* [42]. Here is the deletion–contraction version.

Algorithm C (*Contraction*) *Given a graph G, this algorithm returns the sequence of coefficients $(a_0, a_1, \ldots, a_n)$ of the chromatic polynomial $P(G,q) = \sum_{i=0}^{n} a_i q^i$.*

C1 [Base] If G has no edges then return the coefficients $(0, 0, \ldots, 0, 1)$, 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 Algorithm C recursively to compute $P(G',q)$ and $P(G'',q)$ as sequences of coefficients $(a'_0, a'_1, \ldots, a'_n)$ and $(a''_0, a''_1, \ldots, a''_n)$. Return $(a'_0 - a''_0, a'_1 - a''_1, \ldots, a'_n - a''_n)$, corresponding to the polynomial $P(G/e,q) - P(G-e,q)$. ∎

To analyse the running time, let $T(r)$ be the number of executions of Step C1 for graphs with n vertices and m edges, where $r = n + m$. The two graphs constructed in Step C2 have size $n - 1 + m - 1 = r - 2$ and $n + m - 1 = r - 1$, respectively, so T satisfies $T(r) = T(r-1) + T(r-2)$. This is a well-known recurrence with

solution $T(r) = O(\varphi^r)$, where $\varphi = \frac{1}{2}(1+\sqrt{5})$ is the golden ratio. Thus, Algorithm C requires $\varphi^{n+m}\,\mathrm{poly}(n) = O(1.619^{n+m})$ time. A similar analysis for the algorithm implied by the deletion–addition recursion gives $\varphi^{n+\overline{m}}\,\mathrm{poly}(n)$, where $\overline{m} = \binom{n}{2} - m$ is the number of edges in the complement of G.

These worst-case bounds are often very pessimistic. They do not take into account that recurrences can be stopped as soon as the graph is a tree (or some other easily recognized graph whose chromatic polynomial is known as a closed formula), or that P factorizes over connected components. Moreover, we can use graph isomorphism heuristics and tabulation to avoid some unnecessary recomputation of isomorphic subproblems (see [18]). Thus, Algorithm C is a more useful algorithm than its exponential running time may indicate.

Vertex partitions and dynamic programming

We turn to a different recurrence, which expresses $\chi(G)$ in terms of induced subgraphs of G. By taking S to be a colour class of an optimal colouring of G, we observe that every graph has an independent set of vertices S for which $\chi(G) = 1 + \chi(G - S)$. Thus, we have

$$\chi(G) = 1 + \min \chi(G - S)\,, \tag{1}$$

where the minimum is taken over all non-empty independent sets S in G.

The recursive algorithm implied by (1) is too slow to be of interest. We expedite it using the fundamental algorithmic idea of *dynamic programming*. The central observation is that the subproblems $\chi(G-S)$ for various vertex-subsets S appearing in (1) are computed over and over again. It thus makes sense to store these 2^n values in a table when they are first computed. Subsequent evaluations can then be handled by consulting the table.

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

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

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

D2 [Main loop] List all vertex-subsets $W_1, W_2, \ldots, W_{2^n} \subseteq V$ in non-decreasing order of their size. Do Step D3 for $W = W_2, W_3, \ldots, W_{2^n}$, then terminate.

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

The ordering of subsets in the main loop D2 ensures that each set is handled before any of its supersets. In particular, all values $T(W \setminus S)$ needed in Step D3 will have been previously computed, so the algorithm is well defined. The minimization 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

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

This rather straightforward application of dynamic programming already provides the non-trivial insight that the chromatic number can be computed in time exponential in the number of vertices, 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 notice that S in (1) can be assumed to be a *maximal* independent set – that is, not a proper subset of another independent set. To see this, let f be an optimal colouring and consider the colour class $S = f^{-1}(1)$. If S is not maximal, then repeatedly pick a vertex v that is not adjacent to S, and set $f(v) = 1$.

By considering the disjoint union of $\frac{1}{3}k$ triangles, we see that there exist k-vertex graphs with $3^{k/3}$ maximal independent sets. 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 [7], [34] and [37]). We therefore have the following result:

Theorem 3.1 *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 minimization in Step D3 now takes the following form:

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

Using Theorem 3.1 with $k = |W|$ for the minimization in Step D3′, the total running time of Algorithm D comes within a polynomial factor of

$$\sum_{k=0}^{n} \binom{n}{k} 3^{k/3} = (1 + 3^{1/3})^n = O(2.443^n) \,.$$

For many years, this was the fastest known algorithm for the chromatic number.

3-colouring

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 check whether the complement is bipartite.

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

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

L2 [Try $f(S) = 3$] Use Algorithm B to find a colouring $f : V \setminus S \to \{1,2\}$ of $G - S$ if one exists. In that case, extend f to all of V by setting $f(v) = 3$ for each $v \in S$, and terminate with f as the result. ∎

The running time of Algorithm L is dominated by the number of executions of L2, which, according to Theorem 3.1, is $3^{n/3}$. Thus, Algorithm L decides 3-colourability in time $3^{n/3}\,\mathrm{poly}(n) = O(1.442^n)$ and polynomial space.

The use of maximal independent sets goes back to Christofides [10], while Algorithms D and L are due to Lawler [28]. 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 $O(1.329^n)$ by Beigel and Eppstein [1].

4. Subgraph expansion

The *Whitney expansion* [39] of the chromatic polynomial is

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

see Chapter 3, Section 2 for a proof. It 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, which can be solved in time $O(n + m)$ (for example, by depth-first search). Thus, the Whitney expansion can be evaluated in time $O(2^m(n + m))$.

A more recent expression (see [2]) provides an expansion over *induced* subgraphs:

Theorem 4.1 *For $W \subseteq V$, let $g(W)$ be the number of non-empty independent sets in $G[W]$. Then G can be q-coloured if and only if*

$$\sum_{W \subseteq V} (-1)^{|V \setminus W|} (g(W))^q > 0 \,. \tag{3}$$

Proof. For each $W \subseteq V$, the term $(g(W))^q$ counts the number of ways of selecting q non-empty independent sets $S_1, S_2, \ldots, S_q$, where $S_i \subseteq W$. For $U \subseteq V$, let $h(U)$ be the number of ways of selecting q non-empty independent sets whose union is U. Then $(g(W))^q = \sum_{U \subseteq W} h(U)$, so

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

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

If $h(V)$ is non-zero, then there exist independent sets $S_1, S_2, \ldots, S_q$ whose union is V. These sets correspond to a colouring: associate a colour with the vertices in each set, breaking ties arbitrarily. ∎

For each $W \subseteq V$, we can compute the value $g(W)$ in time $O(2^{|W|}m)$ by constructing each non-empty subset of W and testing it for independence. Thus, the total running time for evaluating (3) is within a polynomial factor of 3^n, just as in the analysis (2) for Algorithm D; however, the space requirement here is only polynomial. We can further reduce the running time to $O(2.247^n)$ by using dedicated algorithms for evaluating $g(W)$ from the literature (see [3]).

If exponential space is available, we can do even better. To that end, we first introduce a recurrence for g.

Theorem 4.2 *Let $W \subseteq V$. We have $g(\varnothing) = 0$, and, for every $v \in W$,*

$$g(W) = g(W \setminus \{v\}) + g(W \setminus N[v]) + 1 \,. \tag{4}$$

Proof. Fix $v \in W$. The non-empty independent sets $S \subseteq W$ can be partitioned into two classes with $v \notin S$ and $v \in S$. In the first case, S is a non-empty independent set with $S \subseteq W \setminus \{v\}$ and thus accounted for by the first term of (4). Consider the second case. Since S contains v and is independent, it contains no vertex from $N(v)$. Thus, S is a non-empty independent set with $\{v\} \subseteq S \subseteq W \setminus N(v)$. The number of such sets is the same as the number of (not necessarily non-empty) independent sets

S' with $S' \subseteq W \setminus N[v]$, because of the bijective mapping $S \mapsto S'$ where $S' = S \setminus \{v\}$. By induction, the number of such sets is $g(W \setminus N[v]) + 1$, where the '+1' term accounts for the empty set. ∎

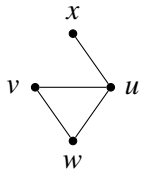
This leads to the following algorithm, due to Björklund *et al.* [3]:

Algorithm I (*Inclusion–exclusion*) *Given a graph G and an integer $q \geq 1$, this algorithm determines whether G can be q-coloured.*

I1 [Tabulate g] Set $g(\varnothing) = 0$. For each non-empty 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 (3)] If $\sum_{W \subseteq V} (-1)^{|V \setminus W|} (g(W))^q > 0$ output 'yes', otherwise 'no'. ∎

Both Steps I1 and I2 take time $2^n \operatorname{poly}(n)$, and the algorithm requires a table with 2^n entries. Fig. 8 shows the computations of Algorithm I on a small graph for $q = 2$ and $q = 3$, with $a_q(W) = (-1)^{|V \setminus W|} (g(W))^q$. The sum of the entries in column a_2 is 0, so there is no 2-colouring. The sum of the entries in column a_3 is 18, so a 3-colouring exists.



W	g	a_2	a_3	W	g	a_2	a_3
$\varnothing$	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. 8. Execution of Algorithm I

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 $2^n \operatorname{poly}(n)$ (see [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 maintaining the $2^n \operatorname{poly}(n)$ running time (see [4]).

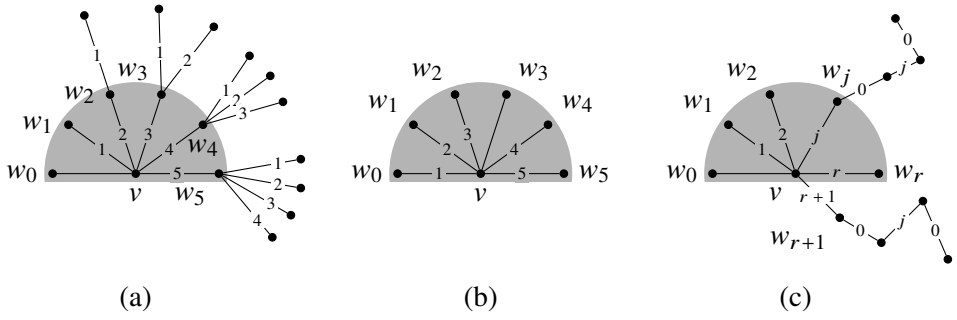
5. Local augmentation

Sometimes, a non-optimal colouring can be improved by a local change that re-colours some vertices. This general idea is the basis of many local search heuristics and also several central theorems.

Kempe changes

An important example, for edge-colouring, establishes Vizing's theorem, $\Delta(G) \le \chi'(G) \le \Delta(G) + 1$. Chapter 5 gives a modern and more general presentation of the underlying idea, and our focus in the present chapter is to make the algorithm explicit.

A colour is *free* at v if it does not appear on an edge at v. (We consider an edge-colouring with $\Delta(G) + 1$ colours, so every vertex has at least one free colour.) A (Vizing) *fan* around v is a maximal set of edges $vw_0, vw_1, \ldots, vw_r$, where vw_0 is not yet coloured and the other edges are coloured as follows. For $j = 0, 1, \ldots, r$, no colour is free at both v and w_j. For $j = 1, 2, \ldots, r$, the jth fan edge vw_j has colour j and the colours appearing around w_j include $1, 2, \ldots, j$ but not $j + 1$ (see Fig. 9(a)). Such a fan allows a recolouring by moving colours as follows: remove the colour from vw_j and set $f(vw_0) = 1, f(vw_1) = 2, \ldots, f(vw_{j-1}) = j$. This is called *downshifting from* j (see Fig. 9(b)).



(a) (b) (c)

Fig. 9. (a) A fan (b) Downshifting from 3 (c) Step V7: colour j is free at w_{r+1}

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

V1 [Initialize] Order the edges arbitrarily $e_1, e_2, \ldots, e_m$. Let $i = 0$.

V2 [Extend colouring to next edge] Increment i. If $i = m + 1$ then terminate. Otherwise, let $vw = e_i$.

V3 [Easy case] If a colour c is free at both v and w, then set $f(vw) = c$ and return to Step 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 with 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 $f(vw_0) = 1$, $f(vw_1) = 2$, and return to Step V2. Otherwise, let vw_2 be the edge incident with 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 $f(vw_r) = c_{r+1}$ and return to Step V2. Otherwise, let vw_{r+1} be the edge incident with v coloured $r + 1$. If each colour $1, 2, \ldots, r$ appears around w_{r+1}, then increment r and repeat Step V6.

V7 [Build a $\{0, j\}$-path from w_j or from w_{r+1}] Let $j \in \{1, 2, \ldots, r\}$ be a free colour at w_{r+1} and let 0 be a colour free at v and different from j. Construct two maximal $\{0, j\}$-coloured paths P_j and P_{r+1} from w_j and w_{r+1}, respectively, by following edges of alternating colours $0, j, 0, j, \ldots$ (see Fig. 9(c)). (The paths cannot both end in v.) Let $k = j$ or $r + 1$ so 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 Step V2. ∎

To see that this algorithm is correct, one needs to check that the recolourings in Steps V6 and V8 are legal. A careful analysis is given by Misra and Gries [33].

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 Step V7; a $\{0, j\}$-path can be constructed in time $O(n)$ if for each vertext we maintain a table of incident edges indexed by colour. Thus the total running time of Algorithm V is $O(mn)$.

Another example from this class of algorithms appears in the proof of Brooks's theorem (see Chapter 2 and [8]), which 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 merely to find a colouring (since $q > 4\Delta$, 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 $q > 4\Delta$, this algorithm finds a uniform random q-colouring f_T in polynomial time.*

M1 [Outer loop] Set $T = \lceil qn \ln 2n/(q-4\Delta)\rceil$. Do Step M2 for $t = 1,2,\ldots,T$, then terminate.

M2 [Recolour a random vertex] Pick a vertex $v \in V$ and a colour $c \in \{1,2,\ldots,q\}$ 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 f_0 can be provided in polynomial time because $q > \Delta + 1$ – for example, by Algorithm G. To see that the choice of initial colouring f_0 has no influence on the result f_T, we 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 $d_t = |\{\, v\colon f_t(v) \neq f_t'(v)\,\}|$ be the number of *disagreeing* vertices after t executions of Step M2. Each step can change only a single vertex, so $|d_t - d_{t-1}| = 1$, 0, or -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 rejects the colour change. In particular, v must have a (disagreeing) neighbour w with $c = f_{t-1}(w) \neq f_{t-1}'(w)$ or $f_{t-1}(w) \neq f_{t-1}'(w) = c$. There are d_{t-1} choices for w and therefore $2\Delta d_{t-1}$ choices for 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 for c and v.

Thus, the expected value of d_t can be bounded as follows:

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

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

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

By Markov's inequality, and because d_T is a non-negative integer, we conclude that

$$\Pr(f_T = f_T') = \Pr(d_T = 0) \ge 1 - \Pr(d_T \ge 1) \ge 1 - \mathbf{E}[d_T] \ge \tfrac{1}{2}\,.$$

We content 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 intuition can be formalized 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 well studied, and the analysis can be much improved (see the survey of Frieze and Vigoda [13]). Randomized local search has wide appeal across disciplines, including simulations in statistical physics and heuristic methods in combinatorial optimization.

6. Vector colouring

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

Vector chromatic number

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

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

The smallest such number q is called the *vector chromatic number* $\vec{\chi}(G)$, which need not be an integer. For instance, the vertices of the 3-chromatic cycle graph C_5 can be laid out on the unit circle in the form of a pentagram ☆. Then the angle between vectors corresponding to neighbouring vertices is $\frac{4}{5}\pi$, corresponding to the scalar product $-1/(\sqrt{5}-1)$, so $\vec{\chi}(C_5) \le \sqrt{5} < 3$.

Theorem 6.1 *If G has clique number $\omega(G)$, 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 that $r \le q$.

For the second inequality, place the vertices belonging to each colour class at the corners of a $(q-1)$-dimensional simplex. To be specific, let $f : V \to \{1, 2, \ldots, q\}$ be an optimal q-colouring and define $x(v) = (x_1, x_2, \ldots, x_n)$ by

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

Then we have

$$\langle x(v), x(v) \rangle = \frac{q-1}{q} + \frac{q-1}{q(q-1)} = 1\,,$$

and for v and w with $f(v) \ne f(w)$ we have

$$\langle x(v), x(w) \rangle = 2\left(\frac{q-1}{q}\right)^{1/2}\left(-\left(\frac{q}{q-1}\right)^{1/2}\right) + \frac{q-2}{q(q-1)} = -\frac{1}{q-1}\,.$$

Thus, x is a vector q-colouring, so $\vec{\chi}(G)$ is at most q. ∎

What makes vector colourings interesting from the algorithmic point of view is that they can be found in polynomial time, at least approximately, using algorithms based on semidefinite programming. The details behind those constructions lie far outside the scope of this chapter (see Gärtner and Matoušek [14]).

Theorem 6.2 *Given a graph G with $\vec{\chi}(G) = q$, a vector $(q + \varepsilon)$-colouring of G can be found in time polynomial in n and $\log(1/\varepsilon)$.*

For a graph with $\omega(G) = \chi(G)$, Theorem 6.1 shows that the vector chromatic number equals the chromatic number. In particular, it is an integer, and can be determined in polynomial time using Theorem 6.2 with $\varepsilon < \frac{1}{2}$. This shows that the chromatic numbers of perfect graphs can be determined in polynomial time. The theory behind this result counts as one of the highlights of combinatorial optimization (see Grötschel, Lovász and Schrijver [16]).

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 two vectors for each connected component, so vector 2-colouring is equivalent to 2-colouring.

But already for $q = 3$, the situation becomes more interesting, since 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 Fig. 10 as an embedding on the unit sphere. More complicated

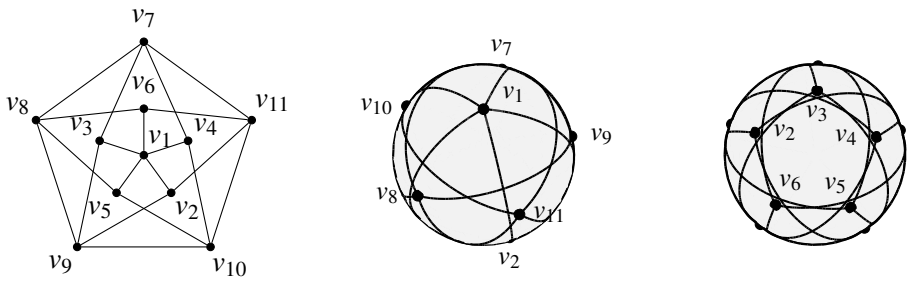


Fig. 10. Left: the Grötzsch graph Middle and right: a vector 3-colouring

constructions (that we cannot visualize) show that there exist vector 3-colourable graphs with chromatic number at least $n^{0.157}$ (see [12] and [22]).

Randomized rounding

Even though the gap between $\vec{\chi}$ and χ can be large for graphs in general, vector colouring turns out to be a useful starting point for (standard) colouring. The next algorithm, due to Karger, Motwani and Sudan [22], translates a vector colouring into a (standard) vertex-colouring using random hyperplanes.

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

R1 [Vector colour] Set $\varepsilon = 2 \cdot 10^{-5}$ and compute a vector $(3+\varepsilon)$-colouring x of G using semidefinite programming. Let $\alpha \geq \arccos(-1/(2+\varepsilon))$ be 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 $H_1, H_2, \ldots, H_r$ in $\mathbb{R}^n$. For each vertex v, let $f(v)$ be the binary number $b_r b_{r-1} \cdots b_1$, where $b_i = 1$ if and only if $x(v)$ is on the positive side of the ith hyperplane H_i.

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

Figure 11 illustrates the behaviour of Algorithm R on the vector 3-colouring of the Grötzsch graph from Fig. 10. Two hyperplanes separate the vertices into four parts. The resulting vertex-colouring with colours from $\{0,1\}^2$ is shown to the right. In this example, the set M of monochromatic edges determined in Step M3 contains only the single edge $v_{10}v_{11}$, drawn bold in the figure.

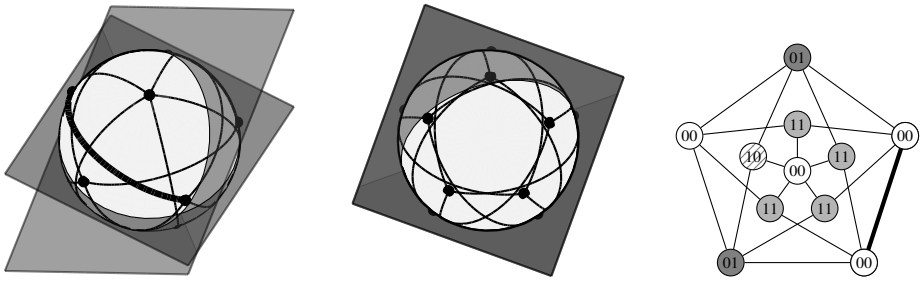


Fig. 11. Left and middle: two hyperplanes Right: the corresponding colouring

Algorithm R algorithm runs in polynomial time, because Theorem 6.2 ensures that Step R1 can be performed in polynomial time.

We proceed to analyze the size of the final colouring. Step R2 uses the colours $\{0, 1, \ldots, 2^{r-1}\}$, so the number of colours used in each Step R2 is

$$2^r \leq (2\Delta)^{-1/\log(\pi/(\pi-\alpha))} < (2\Delta)^{0.631}\,, \tag{5}$$

what is more difficult is to bound the total number of recursive invocations. To this end, we need to understand how fast the instance size, determined by the size of M in Step R3, shrinks.

Let e be an edge whose endpoints received the vector colours $\mathbf{x}$ and $\mathbf{y}$. Elementary geometrical considerations establish the following result.

Theorem 6.3 *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ with angle φ (in radians). A random hyperplane in $\mathbb{R}^d$ fails to separate $\mathbf{x}$ and $\mathbf{y}$ with probability $1 - \varphi/\pi$.*

The angle between the vectors $\mathbf{x}$ and $\mathbf{y}$ is at most α. (To gain some intuition of this, if we ignore the error term ε, Theorem 6.3 shows that $\mathbf{x}$ and $\mathbf{y}$ end up on the same side of a random hyperplane with probability $1-\alpha/\pi \leq 1-\arccos(-\frac{1}{2})/\pi = 1-2\pi/3\pi = \frac{1}{3}$.) The edge e is monochromatic if all r independent random hyperplanes fail to separate $\mathbf{x}$ and $\mathbf{y}$ in Step R2. Thus,

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

By linearity of expectation, the expected size of M is

$$\mathbf{E}[|M|] = \sum_{e \in E} \Pr(e \in M) \leq m/2\Delta \leq \tfrac{1}{4}n\,.$$

Since each edge has two vertices, the expected number of vertices in the recursive instance $G[M]$ is at most $\frac{1}{2}n$, and in general, for $i > 2$, 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. With the bound (5) on the number of colours used per round, we conclude that the total number of colours used is $O(\Delta^{0.631} \log n)$ in expectation.

In terms of Δ, Algorithm R is much better than the bound of $\Delta + 1$ guaranteed by Algorithm G. For an expression in terms of n, we are tempted to bound Δ by $O(n)$, 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. Run Steps W1 and W2 as long as the maximum degree of the graph G is larger than some threshold d, and then colour the remaining graph using Algorithm R. The number of colours used by the combined algorithm is of the order of $(2n/d) + (2d)^{0.631} \log n$, 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 further (see Langberg's survey [27]). The current best polynomial-time algorithm for colouring a 3-chromatic graph based on vector colouring uses $O(n^{0.208})$ colours, due to Chlamtac [9].

7. Reductions

The algorithms in this chapter are summarized in Table 1.

Algorithm	Time	Problem
B Bipartition	$O(n+m)$	2-colouring
C Contraction	$O(1.619^{n+m})$	$P(G,q)$
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	$O(1.443^n)$	3-colouring
M Metropolis dynamics	$\operatorname{poly}(n)$	random q-colouring ($q > 4\Delta$)
P Palette restriction	$1.5^n \operatorname{poly}(n)$	3-colouring
R Rounded vector colouring	$\operatorname{poly}(n)$	$O(\Delta^{0.681} \log n)$-colouring for $\chi(G) = 3$
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	$q^n \operatorname{poly}(n)$	$P(G,q)$

Table 1. Algorithms discussed in this survey

Not only do these algorithms achieve different running times and quality guarantees, they also differ in which specific problem they consider. Let us now be more precise about the variants of the graph colouring problem:

Decision Given a graph G and an integer q, decide whether q can be q-coloured.

Chromatic number Given a graph G, compute the chromatic number $\chi(G)$.

Construction Given a graph G and an integer q, construct a q-colouring of G.

Counting Given a graph G and an integer q, compute the number $P(G,q)$ of q-colourings of G.

Sampling Given a graph G and an integer q, construct a random q-colouring of G.

Chromatic polynomial Given a graph G, compute the chromatic polynomial – that is, the coefficients of the integer polynomial $q \mapsto P(G,q)$.

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

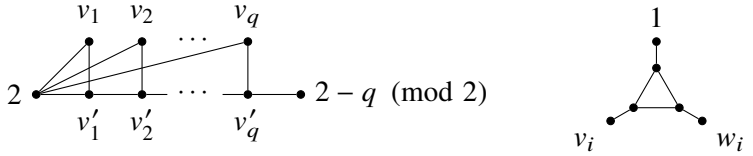
Reduction C (*Constructing a colouring using a decision algorithm*). Suppose that we have an algorithm that decides whether a given graph G can be q-coloured. If $G = K_n$ and $n \leq q$, give each vertex its own colour and terminate. Otherwise, select two non-adjacent vertices v and w in G. If $G +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

Some of our algorithms work only for a specific fixed q, such as Algorithm B for 2-colourability or Algorithm L for 3-colourability. Clearly, 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 an integer q, this reduction constructs a graph H that is 3-colourable with colours $\{0, 1, 2\}$ if and only if G is q-colourable with colours $\{1, 2, \ldots, q\}$.

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

For each vertex $v \in V$, the graph H contains $2q$ vertices $v_1, v_2, \ldots, v_q$ and $v'_1, v'_2, \ldots, v'_q$. Our 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 v has colour i in G. The vertices are arranged as in Fig. 12(a); the right-most vertex is 1 or 2, depending on the parity of q. The vertices $v_1, v_2, \ldots, v_q$ are all adjacent to 2, and so must be coloured 0 or 1.



(a) (b)

Fig. 12.

Moreover, at least one of them must be coloured 1, since otherwise, the colours for $v'_1, v'_2, \ldots, v'_q$ are forced to alternate as $1, 2, 1, \ldots$, conflicting with the colour of the right-most vertex.

Now consider an edge vw in G. Let $v_1, v_2, \ldots, v_q$ and $w_1, w_2, \ldots, w_q$ be the corresponding 'indicator' vertices in H. For each colour $i = 1, 2, \ldots, q$, the vertices v_i and w_i are connected by a 'fresh' triangle as shown in Fig. 12(b). This ensures that v_i and w_i cannot both be 1. In other words, v and w cannot have received the same colour. ∎

The above reduction, essentially due to Lovász [30], can easily be extended to a 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. For instance, 4-colourability is not easier than 3-colourability for general graphs.

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

Many constructions have been published that show the computational difficulty of colouring for restricted classes of graphs. We will sketch an interesting example due to Stockmeyer [35]: the restriction of the case $q = 3$ to planar graphs. Consider the subgraph in Fig. 13(a), called a *planarity gadget*. One can check that this

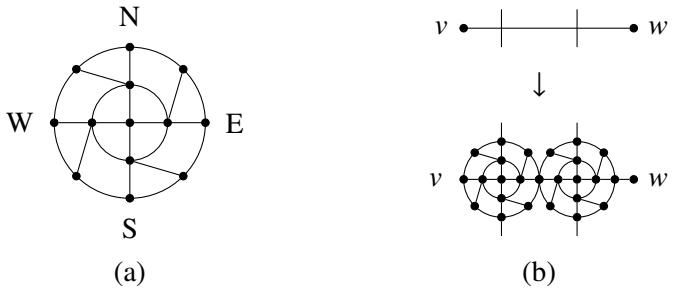


Fig. 13. A planarity gadget

subgraph has the property that every 3-colouring f satisfies $f(\mathrm{E}) = f(\mathrm{W})$ and $f(\mathrm{N}) = f(\mathrm{S})$. Moreover, every partial assignment f to $\{\mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W}\}$ that satisfies $f(\mathrm{E}) = f(\mathrm{W})$ and $f(\mathrm{N}) = f(\mathrm{S})$ can be extended to a 3-colouring of the entire subgraph.

The gadget is used to transform a given (non-planar) graph G as follows. Draw G in the plane and for each edge vw replace each edge intersection by the planarity gadget. The outer vertices of neighbouring gadgets are identified, and v is identified

with W in its neighbouring gadget (see Fig. 13(b)). The resulting graph is planar, and it can be checked that it is 3-chromatic if and only if G is 3-chromatic. Thus, the restriction to planar instances does not make 3-colourability computationally easier. Unlike the case for non-planar graphs, this construction cannot be generalized to larger $q > 3$, since the decision problem for planar graphs and every $q \geq 4$ has answer 'yes' because of the four-colour theorem.

Computational complexity

The field of computational complexity relates algorithmic problems from various domains to one another in order to establish a notion of computational difficulty. The chromatic number problem was one of the first to be analysed in this fashion. The following reduction, essentially from the seminal paper of Karp [23], shows that computing the chromatic number is 'hard for the complexity class NP' by reducing from the NP-hard satisfiability problem for Boolean formulas on conjunctive normal form (CNF). This implies that all other problem in the class NP reduce to the chromatic number.

The input to *CNF-Satisfiability* is a Boolean formula consisting of s clauses $C_1, C_2, \ldots, C_s$. Each clause C_j consists of a disjunction $C_j = (l_{j1} \vee l_{j2} \vee \cdots \vee l_{jk})$ of literals. Every literal is a variable $x_1, x_2, \ldots, x_r$ or its negation $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_r$. The problem is to find an assignment of the variables to 'true' and 'false' that makes all clauses true.

Reduction K (*Satisfiability using chromatic number*). Given an instance $C_1, C_2, \ldots, C_s$ of CNF-Satisfiability over the variables $x_1, x_2, \ldots, x_r$, this reduction constructs a graph G on $3r + s + 1$ vertices such that G can be coloured with $r + 1$ colours if and only the instance 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 each variable x_i $(1 \leq i \leq r)$ the graph contains two adjacent 'literal' vertices v_i and $\overline{v}_i$, both adjacent to all 'true colour' vertices $\{1, 2, \ldots, r\}$ except i. Thus, one of the two vertices $v_i, \overline{v}_i$ must be assigned the 'true' colour i, and the other must be coloured 0. The construction is completed with 'clause' vertices w_j, one for each clause C_j $(1 \leq j \leq s)$. Let $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ be the variables appearing (positively or negatively) in C_j. Then w_j is adjacent to $\{0, 1, \ldots, r\} \setminus \{i_1, i_2, \ldots, i_k\}$. This ensures that only the 'true' colours $\{i_1, i_2, \ldots, i_k\}$ are available at w_j. Furthermore, if x_i appears positive in C_j, then w_j is adjacent to $\overline{v}_i$; if x_i appears negated in C_j, then w_j is adjacent to v_i. Figure 14 shows the reduction for a small instance consisting of just the clause $C_1 = (x_1 \vee \overline{x}_2 \vee \overline{x}_3)$ and a valid colouring corresponding to the

assignment $x_1 = x_3 =$ true, $x_2 =$ false; the edges of the clique on $\{0,1,2,3\}$ are not shown. Thus, the only colours available to w_j are those chosen by its literals. ∎

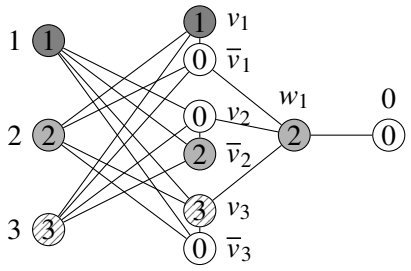


Fig. 14. A 4-colouring instance corresponding to $C_1 = (x_1 \vee \overline{x}_2 \vee \overline{x}_3)$

Edge-colouring

A mapping $f\colon E \to \{1,2,\ldots,q\}$ is an edge-colouring of G if and only if it is a vertex-colouring of the line graph $L(G)$ 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 $2^m\,\mathrm{poly}(n)$, which is the fastest currently known algorithm. Similarly, Algorithm G finds an edge-colouring with $(2\Delta - 1)$ colours, but this is worse than Algorithm V. In fact, since $\Delta \le \chi'(G) \le \Delta + 1$, 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, as shown by Holyer [19] for $\chi'(G) = 3$ and Leven and Galil [29] for $\chi'(G) > 3$.

Approximating the chromatic number

Algorithm V shows that the chromatic index can be very well approximated. In contrast, approximating the chromatic *number* is much harder. In particular, it is NP-hard to 4-colour a 3-chromatic graph (see [17]). This rules out an approximate vertex-colouring algorithm with a performance guarantee as good as Algorithm V, but is far from explaining why the considerable machinery behind, say, Algorithm R results only in a colouring of size n^c for 3-chromatic graphs. The best currently known exponent is $c = 0.204$ (see [24]).

For sufficiently large fixed q, it is NP-hard to find an $\exp(\Omega(q^{1/3}))$-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, maybe there is some other polynomial-time computable function g so that we can guarantee, for example, $g(G) \leq \chi(G) \leq n^{0.999}g(G)$? The answer turns out to be 'no' under standard complexity-theoretic assumptions: For every $\varepsilon > 0$, it is NP-hard to approximate $\chi(G)$ within a factor $n^{1-\varepsilon}$, as shown by Zuckerman [41].

Counting

The problem of counting the q-colourings 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 $q = 0, 1, \ldots, n$. Moreover, note that $\chi(G) \geq q$ 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 [21] relates the estimation of the size of a finite set to the problem of uniformly sampling 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 $g(G)$ for which $(1-\varepsilon)g(G) \leq P(G,q) \leq (1+\varepsilon)g(G)$ with high probability in time polynomial in n and $1/\varepsilon$ for any $\varepsilon > 0$. Much better bounds on q are known (see the survey of Frieze and Vigoda [13]). Without some bound on q, such an FPRAS is unlikely to exist because, with $\varepsilon = \frac{1}{2}$, it would constitute a randomized algorithm for the decision problem and would therefore imply that all of NP can be solved in randomized polynomial time.

Conclusion

Together, the algorithms and reductions presented in this survey give a picture of the computational aspects of graph colouring. For instance, 2-colouring admits a polynomial time algorithm, while 3-colouring does not. In the planar case, 4-colouring is trivial, but 3-colouring is not. An almost optimal edge-colouring can be found in polynomial time, but vertex-colouring is very difficult to approximate. If q is sufficiently large compared to $\Delta(G)$ then the set of colourings can be sampled and approximately counted, but not counted exactly. Finally, even the computationally hard colouring problems admit techniques that are much better than our initial Algorithm X.

None of these insights is obvious from the definition of graph colouring, so the

algorithmic perspective on chromatic graph theory has proved to be a fertile source of questions with interesting answers.

References

1. R. Beigel and D. Eppstein, 3-coloring in time $O(1.3289^n)$, *J. Algorithms* **111** (2005), 168–204.
2. A. Björklund and T. Husfeldt, Exact algorithms for exact satisfiability and number of perfect matchings. *Algorithmica* **52** (2008), 226–249.
3. A. Björklund, T. Husfeldt and M. Koivisto, Set partitioning via inclusion–exclusion, *SIAM J. Comput.* **39** (2009), 546–563.
4. A. Björklund, T. Husfeldt, P. Kaski and M. Koivisto, Covering and packing in linear space, *Inform. Process. Lett.* **111** (2011), 1033–1036.
5. A. Björklund, T. Husfeldt, P. Kaski and M. Koivisto, Computing the Tutte polynomial in vertex-exponential time, *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, IEEE Computer Society (2008), 677–686.
6. D. Brélaz, New methods to color the vertices of a graph, *Comm. Assoc. Comput. Mach.* **22** (1979), 251–256.
7. C. Bron and J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, *Comm. Assoc. Comput. Mach.* **16** (1973), 575–577.
8. R. L. Brooks, On colouring the nodes of a network, *Math. Proc. Cambridge Philos. Soc.* **37** (1941), 194–197.
9. E. Chlamtac, Approximation Algorithms Using Hierarchies of Semidefinite Programming Relaxations, in *Proc. 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*, IEEE Computer Society (2007), 691–701.
10. N. Christofides, An algorithm for the chromatic number of a graph. *Comput. J.* **14** (1971), 38–39.
11. V. Chvátal, Perfectly ordered graphs, *Topics on Perfect Graphs* (eds. C. Berge and V. Chvátal), *Ann. Discrete Math.* **21** (1984), 63–68.
12. U. Feige, M. Langberg and G. Schechtman, Graphs with tiny vector chromatic numbers and huge chromatic numbers, *SIAM J. Comput.* **33** (2004), 1338–1368.
13. A. Frieze and E. Vigoda, A survey on the use of Markov chains to randomly sample colorings, *Combinatorics, Complexity and Chance*, Oxford University Press, 2007.
14. B. Gärtner and J. Matoušek, *Approximation Algorithms and Semidefinite Programming*, Springer, 2012.
15. G. Grimmett and C. McDiarmid, On colouring random graphs, *Math. Proc. Cambridge Philos. Soc.* **77** (1975), 313–324.
16. M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer, 1988.
17. V. Guruswami and S. Khanna, On the hardness of 4-coloring a 3-colorable graph, *SIAM J. Discrete Math.* **18** (2004), 30–40.

18. G. Haggard, D. J. Pearce and G. Royle, Computing Tutte polynomials, *Assoc. Comput. Mach. Math. Software* **37** (2010), Article 24.
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, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, Springer Lect. Notes Comput. Sci. **8096** (2013), 233–243.
21. M. Jerrum, L. G. Valiant and V. V. Vazirani, Random generation of combinatorial structures from a uniform distribution, *Theor. Comput. Sci.* **43** (1986), 169–188.
22. D. R. Karger, R. Motwani and M. Sudan, Approximate graph coloring by semidefinite programming, *J. Assoc. Comput. Mach.* **45** (1998), 246–265.
23. R. M. Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations* (eds. R. E. Miller and J. W. Thatcher), Plenum (1972), 85–103.
24. K. Kawarabayashi and M. Thorup, Combinatorial coloring of 3-colorable graphs, *Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, IEEE Computer Society (2012), 68–75.
25. A. Kosowski and K. Manuszewski, Classical coloring of graphs, *Graph Colorings* (ed. M. Kubale), *Amer. Math. Soc. Contemp. Math.* **352** (2004), 1–20.
26. L. Kučera, The greedy coloring is a bad probabilistic algorithm, *J. Algorithms* **12** (1991), 674–684.
27. M. Langberg, Graph coloring, *Encyclopedia of Algorithms* (ed. M. Kao), Springer (2008), 368–371.
28. E. L. Lawler, A note on the complexity of the chromatic number problem, *Inform. Process. Lett.* **5** (1976), 66–67.
29. D. Leven and Z. Galil, NP completeness of finding the chromatic index of regular graphs, *J. Algorithms* **4** (1983), 35–44.
30. L. Lovász, Coverings and coloring of hypergraphs, *Proc. Fourth Southeastern Conference on Combinatorics, Graph Theory, and Computing*, Boca Raton, *Congr. Numer.* **8** (1973), 3–12.
31. F. Maffray, On the coloration of perfect graphs, *Recent Advances in Algorithms and Combinatorics*, *CMS Books Math.*, Springer (2003), 65–84.
32. D. W. Matula, A min–max theorem for graphs with applications to graph coloring, *SIAM Rev.* **10** (1968), 481–482.
33. J. Misra and D. Gries , A constructive proof of Vizing's Theorem, *Inform. Proc. Lett.* **41** (1992), 131–133.
34. J. W. Moon and L. Moser, On cliques in graphs, *Israel J. Math.* **3** (1965), 23–28.
35. L. Stockmeyer, Planar 3-colorability is polynomial complete, *Assoc. Comput. Mach. SIGACT News* **5** (1973), 19–25.
36. G. Szekeres and H. S. Wilf, An inequality for the chromatic number of a graph, *J. Combin. Theory* **4** (1968), 1–3.
37. E. Tomita, A. Tanaka and H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments, *Theor. Comput. Sci.* **363** (2006), 28–42.

38. D. J. A. Welsh, and 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.
39. H. Whitney, A logical expansion in mathematics, *Bull. Amer. Math. Soc.* **38** (1932), 572–579.
40. A. Wigderson, Improving the performance guarantee for approximate graph coloring, *J. Assoc. Comput. Mach.* **30** (1983), 729–735.
41. D. Zuckerman, Linear degree extractors and the inapproximability of Max Clique and Chromatic Number, *Theory of Computing* **3** (2007), 103–128.
42. A. A. Zykov, On some properties of linear complexes (Russian), *Math. Sbornik* **24** (1949), 163–188.