Time optimal d-list colouring of a graph

Nick Gravin

St.Petersburg Department of Steklov Mathematical Institute RAS.
Nanyang Technological University of Singapore.

Abstract. We present the first linear time algorithm for d-list colouring of a graph—i.e. a proper colouring of each vertex v by colours coming from lists $\mathcal{L}(v)$ of sizes at least $deg(v)$. Previously, procedures with such complexity were only known for Δ-list colouring, where for each vertex v one has $|\mathcal{L}(v)| \geq \Delta$, the maximum of the vertex degrees. An implementation of the procedure is available.

1 Introduction

Graph colouring is probably the most popular subject in graph theory. Although even the 3-colourability of a graph is known to be an NP-complete problem, there is a number of positive results dealing with the case where the number of colours is close to Δ — the maximal degree of a graph. The most famous among them is the Brooks's theorem [5] asserting that a connected graph that is neither a clique nor an odd cycle, can be coloured with Δ colours. However, the question becomes much harder even for $\Delta - 1$ colours. Borodin and Kostochka conjectured in [4] that for $\Delta \geq 9$ the graphs with no clique of size Δ are $(\Delta - 1)$-colourable. The conjecture remains open; for $\Delta \geq 10^{14}$ it was affirmed by Reed in [15] by means of the probabilistic method.

Another extension of the Brooks's theorem in the direction of list-colourings was made independently by Vizing [19] and Erdős et al. [7]. Recently this line of research became even more popular than simple colourings. For example it is studied in depth in [9, 18, 13, 11] and many other sources, e.g. as recently as [10]. In 1994 at a graph theory conference held at Oberwolfach Thomassen pointed out that one can also prove a choosability version of Gallai's result [8], which characterized the subgraphs of k-colour-critical graphs induced by the set of all vertices of degree $k - 1$. Surprisingly, as Kostochka in [12] has mentioned, this theorem can be easily derived from the result of Borodin [2, 3] and Erdős et al. [7]. Specifically, Erdős has characterised all d-choosable graphs as follows. The non-d-choosable graphs are exactly the graphs, whose blocks are cliques or odd cycles. So if a graph possesses a block that is neither a clique nor an odd cycle, it can be coloured for every d-list-assignment. This characterisation does not treat the case of colouring of a non-d-choosable graph for a given d-list-assignment. The latter was addressed by Borodin in little-known [2], covering the case when we are given a non-d-choosable graph and a d-list-assignment.

The present paper focused on a linear-time algorithm for the d-list colouring. We also present a proof of the Borodin's result as a corollary of our algorithmic result.

Algorithmic aspects. Although the general list-colouring problem is a hard problem even to approximate, there is a linear time algorithm by Skulrattanakulchai [16] for any Δ-list-assignment, which improves the Lovász's algorithm [14] of such complexity for Δ-colouring. The algorithm of Skulrattanakulchai emerged from a new proof of a theorem of Erdős, Rubin and Taylor [7]. Unlike the latter, we rely upon a simple idea of recursive deletion of vertices presented in the proof by Dirac and Lovász [1] of Brooks Theorem [5]. It allows us to present the first linear time algorithm for d-list colouring problem. Since d-list colouring obeys a simple rule of reduction, our procedure can be easily applied to the problem of partial d-list colouring completion and thus to the completion of a partial Δ-list colouring. So our work generalises, in a number of directions, the result of [16] where the first time optimal algorithm for Δ-list colouring is presented. We describe, as a part of the main algorithm, a linear-time procedure for Δ-colouring adapted for usage in our main algorithm, and which also can be applied to general Δ-colouring problems. It also yields an algorithmic proof for the Brooks's Theorem. Our procedure appears to be quite practical.
An implementation of the entire algorithm together with the procedure can be found in [20]. Finally, we remark that our work can be considered as an algorithmic, and rather nontrivial, counterpart to Borodin's result [2].

2 Preliminaries

For a graph G, a *proper colouring* or just a *colouring* of G is a map $c : V(G) \to \mathbb{N}$ such that $c(v) \neq c(u)$ whenever $uv \in E(G)$. It is called a k-colouring if at most k colours were used. It is natural to consider a more restricted setting, where the allowed colours are prescribed for each vertex. Formally, a *list-assignment* is a function $\mathcal{L}$ assigning to every vertex $v \in V(G)$ a set $\mathcal{L}(v)$ of natural numbers, which are called *admissible colours* for v. An *$\mathcal{L}$-colouring* of G is a proper colouring c of G which assigns to any vertex v a colour $c(v) \in \mathcal{L}(v)$. If $|\mathcal{L}(v)| \geq k$ for every $v \in V(G)$, then $\mathcal{L}$ is a *k-list-assignment*. If $|\mathcal{L}(v)| \geq deg(v)$ for every $v \in V(G)$, then $\mathcal{L}$ is a *d-list-assignment*. A *partial $\mathcal{L}$-colouring* of G of a vertex subset S is an $\mathcal{L}$-colouring of the induced subgraph $G(S)$. An $\mathcal{L}$-colouring $\mathcal{C}$ of G is a *precolouring extension* of a partial colouring $\mathcal{C}'$ on the set S if $\mathcal{C}(v) = \mathcal{C}'(v)$ for all $v \in S$.
Thus, we are given a graph and a list-assignment function $\mathcal{L}$, and the task is either to find a proper $\mathcal{L}$-colouring, or to establish its nonexistence. In a precolouring extension of $\mathcal{L}$-colouring problem we are given, in addition to G and $\mathcal{L}$, a precoloured set $S \subset V$ with a proper colouring on $G(S)$, and the task is to extend this $\mathcal{L}$-colouring to the all vertices of G or to find that it is impossible.
Graph is called *d-choosable* if for any d-list assignment function $\mathcal{L}$ the *$\mathcal{L}$-colouring Problem* has a solution.

2.1 Terminology

For a graph $G = (V, E)$ let $E = E(G)$ denote the set of edges of G and $V = V(G)$ denote the set of vertices of G. For a subset S of vertices, $G(S)$

denotes the induced subgraph of G on S. In this paper we consider only finite simple graphs, i.e. those without loops and multi-edges. We denote by $deg_G(u)$, or $deg(u)$, the degree of u in G, i.e. the number of edges incident with u. Let us from the very beginning assume that G is connected, since the colouring problem can be solved separately for each component of G.
A *cut* vertex is a vertex after deletion of which the graph splits into more than one component of connectivity. A two-connected graph is a graph which remains connected after deletion of any vertex (here we assume that an empty graph is connected). A *block* of G is a maximal by inclusion two-connected induced subgraph of G. Let us recall some facts about blocks and cut vertices, which one can find in [1] or in [6]:

- Two blocks can have at most one common vertex and if they have one then it is a cut vertex of G.
- Any edge of G belongs to some block.
- A bipartite graph with the first part being the set of blocks of G and the second one being the set of cut vertices of G such that every cut vertex is adjacent to the blocks which contain it, is a tree.

We call such a tree of blocks and cut vertices a *block tree*.
Suppose we have a spanning tree T of G with a root r, and an edge orientation towards the root. Such a tree is called *normal* if there is an oriented path in T joining any two $u, v \in V$ such that $(u, v) \in E$. In [6] it is proved that every connected graph contains a normal tree. One can easily check that the depth-first search spanning tree is a normal tree. Using this fact one can easily see that the root of normal spanning tree of two-connected graph is a leaf in this tree, since otherwise by deleting the root we will get more than one component of connectivity.

2.2 Main results

Before discussing the algorithm, let us describe some natural requirements on the input data. In the present paper we consider the most concise graph representation, i.e. the vertices of the graph are represented by integers from 1 to $N = |V|$, and $E(G)$ is given as a set of pairs.

Definition 1. *Let the maximum value of the colour for $\mathcal{L}$ be at most $c|E(G)|$, for a constant c. Then we say that $\mathcal{L}$ satisfies the colour density condition.*

One needs the latter condition to efficiently handle colours. It is not restrictive, since it allows one to describe all possible essentially different situations for $\mathcal{L}$ up to renumbering of the colours. The same condition on the maximal colour value and numeration of vertices was imposed in [16].

Theorem 1. *There is a linear time algorithm which, for a given graph G and d-list-assignment function $\mathcal{L}$ satisfying the colour density condition, either finds an $\mathcal{L}$-colouring of G, or reports its nonexistence.*

Our algorithm is effectively based on the following theorem characterising all the list assignments and graphs which have an admissible assignment.

Theorem 2. [1] *Let $G = (V, E)$ be a graph, with the set of blocks $\mathcal{B}$, and $\mathcal{L}$ a d-list-assignment for G. Then G has no d-list colouring if and only if:*

[1] Due to the space limitation we have placed the proof of theorem 2 in the appendix.

1. *Every $B \in \mathcal{B}$ is either a clique or an odd cycle.*
2. *There exists a map $\mathcal{S}$ which assigns to each pair (v, B), where $v \in B \in \mathcal{B}$, a set of colours $\mathcal{S}(v, B)$ satisfying the following conditions:*
   (a) $\bigcup_{B:\ v \in B \in \mathcal{B}} \mathcal{S}(v, B) = \mathcal{L}(v)$
   (b) $\mathcal{S}(v, B) \bigcap \mathcal{S}(v, B') = \emptyset$ *for any* $B \neq B' \in \mathcal{B}$ *with* $v \in B \cap B'$.
   (c) $|\mathcal{S}(v, B)| = deg_{G(B)}(v)$ *for any* $v \in B$
   (d) $\mathcal{S}(v, B) = \mathcal{S}(u, B)$ *for any* $u, v \in B$

Remark 1. Conditions 2a and 2c of the theorem imply condition 2b; the condition 1 is just the Erdős' characterisation of the d-choosable graphs. Note that the appearance of cliques and odd cycles is not a coincidence, due to Brooks Theorem [5], that says that a connected graph H that is neither a clique nor an odd cycle satisfies $\chi(H) \leq \Delta(H)$.

Remark 2. This theorem is a reformulation of the Borodin's result [2]. One more variant of the Borodin's result one can find in [12].

In other words, the theorem states that non-colourable instances of d-list colouring problem can be described as follows. In the block tree of G we can split every list of a cut vertex v into t disjoint parts and leave only one such part for a block containing v (each part should corresponds to exactly one block) in such a way, that now it is impossible to find a proper colouring of any block using only the lists of admissible colours of its vertices (for each cut vertex in the block we have to use the part of the list we have left for these block and vertex). (See Fig. 1)
Keeping in mind this characterisation while reading the algorithm description may make the latter more transparent. The algorithm for Theorem 1 is described in Section 3.

3 *d*-List Colouring Problem

INPUT:
A graph $G = (V, E)$ and list-assignment $\mathcal{L}$ with $\mathcal{L}(v) \geq deg_G(v)$ for every vertex $v \in V$.

TASK:
Find a proper $\mathcal{L}$-colouring of G or deduce that there is none.

To simplify the presentation, let G be a connected graph. However almost the same algorithm works for not necessarily connected graphs.

3.1 Algorithm

Let us give a few definitions before we start describing an algorithm.

Definition 2. *We call a block a leaf block if it contains at most one cut vertex, i.e. it corresponds to a leaf vertex in the block tree $\mathcal{T}$.*

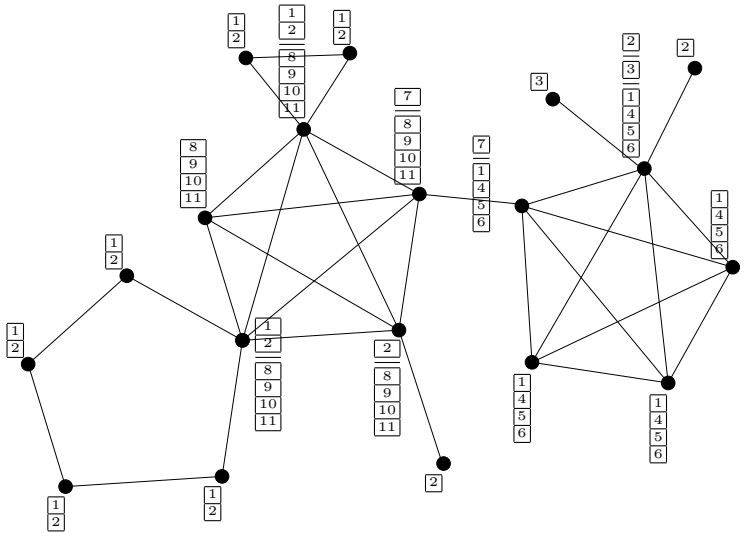

Fig. 1. A graph and $\mathcal{L}$-assignment that does not lead to a $\mathcal{L}$-colouring. The colour list of each cut vertex is split into a number of lists according to the statement of Theorem 2.

Definition 3. *We call an ordered pair of two adjacent vertices (u, v) distinguished by the colour i if $i \in \mathcal{L}(u) \setminus \mathcal{L}(v)$.*

In the main algorithm we will use the following auxiliary procedures:

1. **Recursive deletion of vertices**. It finds a desired d-colouring of a connected graph G, if G has a vertex v with $|\mathcal{L}(v)| > deg_G(v)$.
2. **Search for blocks of** G. It finds the representation of each block by the set of its vertices.
3. **Search for a colour distinguishing the pair** (u, v)**.** It requires us to introduce one boolean array $\mathcal{C}$ of the size $c|E(G)|$, which we reuse by every application of the procedure.
4. **Δ-colouring of a connected graph**. For a connected graph H that is neither a clique nor an odd cycle, the procedure finds a proper Δ-colouring.

Now we are ready to describe the algorithm.

- **STEP 1.** Suppose there exists $v \in V$ with $|\mathcal{L}(v)| > deg_G(v)$. The **recursive deletion of vertices** will give us a colouring we seek and we terminate the algorithm.
- **STEP 2. A)** Find all blocks of G and the corresponding block tree $\mathcal{T}$.
  1. Find the blocks of G using **search for blocks of** G.
  2. For each vertex v of G find a list of the blocks containing v.

3. Construct a bipartite graph H_0 with one part consisting of all blocks and another one consisting of all cut vertices and with $E(H_0) = \{(v, B) \mid v \in B\}$.
4. Construct a block tree $\mathcal{T}$ as the *breadth-first search tree* of H_0, i.e. find the corresponding order $\mathcal{B}$ on the blocks and cut vertices.

– **B)** Take the last block B in the order $\mathcal{B}$ and its unique cut vertex v_0. In the next steps we are trying to reduce G by cutting off $B \setminus \{v_0\}$, or find a desired colouring.
– **STEP 3.** Try to find in B a pair of distinguished by a colour vertices (u, v), such that u is not a cut vertex, i.e. $u \neq v_0$.
1. If there is such a pair in B, say (u, v) distinguished by the colour c, then we colour u with c and delete it from G. In the remaining graph, $|\mathcal{L}(v)| > deg(v)$. Also the graph $G' = G \setminus \{v\}$ is connected. We colour G' by the **recursive deletion of vertices** and terminate.
2. Otherwise we proceed to the next step.

– **STEP 4.** Check whether the block B is a clique or an odd cycle. Can be done by pre-computing vertex degrees of G.
1. If B is a clique or an odd cycle, we reduce G by cutting off $B \setminus \{v_0\}$. Since in STEP 3 v_0 and a vertex v adjacent to v_0 are not distinguished by a colour, we have $\mathcal{L}(v_0) \supseteq \mathcal{L}(v)$. We reduce $\mathcal{L}(v_0)$ by $\mathcal{L}(v)$. Then we return to the STEP 2 B) with next after B in $\mathcal{B}$ block. Here we use a property of order $\mathcal{B}$ to reduce efficiently $\mathcal{L}(v_0)$.
2. Other possibilities for B. In this case we necessarily get a desired colouring and terminate.
   - Colour $G(V \setminus V(B))$ by applying the **recursive deletion of vertices** in $G(V \setminus B)$.
   - Delete from $\mathcal{L}(v_0)$ all the colours which are present among neighbours of v_0.
   - Check if v_0 has the same as the other vertices in B set of admissible colours.
   - If v_0 does not, then for the remaining not coloured vertices a pair (v_0, u), where u is any vertex adjacent to v_0 in B, is distinguished by a colour. We complete the colouring of the whole graph as in STEP 3.1 and terminate.
   - Otherwise we apply **Δ-colouring** procedure to B.

3.2 Implementation of procedures

Here we give a description of our auxiliary procedures in details together with their complexity analysis.

The recursive deletion of vertices. Suppose there exists $v \in V$ with $|\mathcal{L}(v)| > deg_G(v)$. Put v in a stack $\mathcal{O}$ and delete it from the graph.
Then the remaining graph again should have a vertex with the same property, since G is a connected graph and $|\mathcal{L}(u)| \geq deg(u)$ for any $u \in V$. We repeat the previous action for this vertex, i.e. put it in $\mathcal{O}$ and delete it from G, and keep doing such deletion until there are no vertices left in G.

Then we take the first vertex from the $\mathcal{O}$ and colour it with any admissible colour, then take the second one and again colour it with any admissible colour which does not occur among its already coloured neighbours and so on (we succeed to colour every such vertex because we have appropriately chosen vertices in the $\mathcal{O}$). Eventually we colour all the vertices of G.

Remark 3. This procedure also works well for graphs which have more than one component of connectivity if in each such component there is a vertex w with $|\mathcal{L}(w)| > deg(w)$.

Proposition 1. *The recursive deletion of vertices in G and subsequent colouring, works in linear in the size of the graph time.*

Proof. We can compute the degrees of all vertices of G in linear time. Indeed, after putting any vertex in $\mathcal{O}$, any of its neighbours can be put in $\mathcal{O}$ immediately after it. So we can make a depth-first search starting with a vertex, which has degree less than its list size, and put in $\mathcal{O}$ corresponding vertex at each step of that search. All of this works in linear time.
To colour all the vertices in $\mathcal{O}$ in linear time it is sufficient to colour each vertex $u \in O$ in $O(|\mathcal{L}(u)|)$ time.
As we have the bound on the maximal colour value we can create in linear time, during the initialisation of the algorithm, an array $\mathcal{C}_0$ of boolean values of size equal to maximum value among all possible colours and with initial False values.
At the colouring step of the procedure for a vertex u we match in $\mathcal{C}_0$ with True the colours of all already coloured neighbours of u and then search in $\mathcal{L}(u)$ for a colour c with False value of $\mathcal{C}_0[c]$. When we have found c, we reset back to False, by one pass along the neighbours of u, all the values of $\mathcal{C}_0$. Thus we return $\mathcal{C}_0$ to the initial state and we can use it again for other vertices. Then we colour u with c. This concludes the proof, since we use only $O(|\mathcal{L}(u)|)$ time to colour u; also in total we use only $c|E(G)|$ of space, since we create the array only once.

Search for blocks of G. The algorithm for finding all blocks in linear time can be found, e.g. in [17].

Search for a colour distinguishing the pair (u, v). As we have the bound on the maximal colour value we can in linear in $|\mathcal{L}|$ time create, during the initialisation of the algorithm, an array $\mathcal{C}$ of boolean values of size equal to the maximum value among all possible colours and with initial False values. Then we can mark with True value all the colours which are in $\mathcal{L}(v)$ and then verify whether there is a colour c in $\mathcal{L}(u)$ for which $\mathcal{C}[c]$ is False, and finally reset all the True values of $\mathcal{C}$ back to False. Thus we either find a distinguishing colour or check that there are no such a colour, in linear in $|\mathcal{L}(u)| + |\mathcal{L}(v)|$ time.

Δ-colouring of a connected graph. We use a linear time algorithm for Δ-colouring by Lovász [14]. But there is another, easier, way to deal with the Δ-colouring and we will describe it in Section 4, which also provides a new constructive proof of the Brooks Theorem.

3.3 Proof of correctness of the algorithm

STEP 1. We compute all vertex degrees in G and check for each vertex whether $|\mathcal{L}(v)| > deg_G(v)$.
STEP 2.
A) Find all blocks of G and the corresponding block tree $\mathcal{T}$. We consider the representation of each block by the set of its vertices; also we construct for each vertex a list of all blocks it contains. To recall the definition of block tree, one can think of a bipartite graph H with one part consisting of all blocks and another consisting of all cut vertices (that is vertices which belong to at least two blocks) and with $E(H) = \{(B, v) \mid v \in B\}$; by construction H is a tree and is called the block tree. For complexity reasons we need to construct $\mathcal{T}$ as the **breadth-first search tree**, i.e. we require that $\mathcal{T}$ has the specific order on blocks and cut vertices.

Remark 4. We have to do the **search for blocks** as the unsolvable instances of the problem are described in terms of certain conditions on blocks [7].

For tree representation it is useful to view an orientation towards the root. So we add an artificial vertex to G and join it to a vertex of G. Thus we add an artificial block to $\mathcal{T}$. Let us take this artificial block B_0 to be a root of $\mathcal{T}$. Let us pick a block B_0 and consider the order $\mathcal{B}$ in which blocks appear in a breadth-first search on $\mathcal{T}$ with the root in B_0.
B) Let B be the last block of $\mathcal{T}$ in $\mathcal{B}$.

Note that B is a leaf block. Let us denote by v_0 the cut vertex of B, i.e. the unique cut vertex in $\mathcal{T}$ joined with B. Since B is the last in $\mathcal{B}$, and $\mathcal{B}$ is the order of a breadth-first search, all the other neighbours of v_0 that appear after v_0 in $\mathcal{B}$, are leaf blocks.
STEP 3.
Case A. Suppose we have found a pair of vertices (u, v) distinguished by the colour c, such that u is not a cut vertex in B.
Then we colour u with c and delete it from G. In the remaining graph, $|\mathcal{L}(v)| > deg(v)$. Also the graph $G' = G \setminus \{v\}$ is connected. So we can colour G' by the recursive deletion of vertices.
Case B. If there are no such pairs then for all non-cut vertices $w, v \in B$ we have $\mathcal{L}(v) = \mathcal{L}(w)$, whereas for the cut vertex v_0 one has $\mathcal{L}(v_0) \supseteq \mathcal{L}(v)$. Consequently, all the vertices of $B \setminus \{v_0\}$ have the same degree. Then we come to the STEP 4.

At the next step of the algorithm we either reduce G by cutting off $B \setminus \{v_0\}$ or find a colouring of G by recursive deletion of vertices.

STEP 4. All lists for non cut vertices of B are the same and that one is contained in the list of cut vertex of B.
According to the Brooks Theorem [5] (cf. Remark 1) there are two cases. (That can be distinguished by pre-computing vertex degrees)

Case I (B is either a clique or an odd cycle)
We need to make the reduction of $\mathcal{L}(v_0)$ very carefully, since $|\mathcal{L}(v_0)|$ can be much bigger than the size of B. We describe it in details in Section 3.4.

Case II (remaining possibilities for B) In this case due to classification by Erdős [7] we know that G should have a proper colouring for any list-assignment function.

3.4 Complexity analysis of the algorithm

Clearly each of the Steps 1 and 2 takes only $O(|G| + |\mathcal{L}|)$ time.

STEP 3. Part 1. Search of a distinguished by a colour pair of vertices in $B \setminus \{v_0\}$.

Proposition 2. *Search for a distinguished by a colour pair of vertices (u, v), such that $u, v \neq v_0$, can be done in linear in the block's size time, where by the size of a block we mean the number of edges plus number of vertices in this block.*

Proof. We can consider a spanning depth-first search tree of B with the root in v_0 and check only pairs of admissible colour sets for adjacent vertices in this tree. Since B is a block, v_0 has unique neighbour in the tree. So we do double check for two adjacent vertices u and v twice, at first for the pair (u, v) then for (v, u).
Now let us show that the total time of all checks is linear in $\sum_{v \in B \setminus \{v_0\}} |\mathcal{L}(v)|$.
By every check of u and v which are not cut vertices we either get that $\mathcal{L}(u) = \mathcal{L}(v)$, or find a pair of distinguished by a colour vertices together with such colour. In the latter case we terminate the search of the pair. In the former case we see that the check works in linear in $min(|\mathcal{L}(u)|, |\mathcal{L}(v)|)$ time. So the total time will be linear in $\sum_{v \in B \setminus \{v_0\}} |\mathcal{L}(v)|$.

Part 2. Check whether a pair (u_0, v_0) is distinguished by a colour in linear in $\mathcal{L}(u_0)$ time, where u_0 is adjacent to v_0 in B.

Remark 5. Since B is a block and we consider the depth-first search spanning tree with the root in v_0, we need only to make one check for v_0 and its unique neighbour u_0 in the tree. But we cannot allow ourselves doing this check even in linear in $|\mathcal{L}(v_0)| + |\mathcal{L}(u_0)|$ time in straightforward manner for all blocks pending at v_0. Since $|\mathcal{L}(v_0)|$ may be much bigger than the size of B and furthermore v_0 may be the cut vertex for many small blocks like B, the total time can be quadratic in input size, e.g.for $G = K_{n-1,1}$. For the same reasons we are not allowed to make a straightforward reduction of $\mathcal{L}(v_0)$ in Step 4.

To avoid multiple checks described in Remark 5 and reductions of $\mathcal{L}(v_0)$ for large $\mathcal{L}(v_0)$ we can remember the True values of $\mathcal{L}(v_0)$ in $\mathcal{C}$ and use it not for one pending at v_0 block but for all such blocks. So strictly speaking we do the following:

- Introduce at the beginning of the algorithm a new array $\mathcal{C}'$ with $|\mathcal{C}'| = |\mathcal{C}|$ of boolean, initially marked with False values.
- If the cut vertex of a leaf block B differs from the previous one, say v_0', then we do the following.
  1. Revert $\mathcal{C}'$ to initial state of only False values in $|\mathcal{L}(v_0')|$ time, just going once along $\mathcal{L}(v_0')$.
  2. Mark in $\mathcal{C}'$ by True values the colours of $\mathcal{L}(v_0)$.
- For a leaf block B with the cut vertex v_0 check the pair (u_0, v_0) in linear in $|\mathcal{L}(u_0)|$ time. One can do it by one passing along $\mathcal{L}(u_0)$ and checking if $\mathcal{C}'[c]$ is True for $c \in \mathcal{L}(u_0)$.

For the search of a distinguishing colour for the pair (u_0, v_0) it suffice to use $\mathcal{C}'$ instead of $\mathcal{L}(v_0)$, since in the search we only check whether a colour $i \in \mathcal{L}(u_0)$ belongs to $\mathcal{L}(v_0)$.

STEP 4.

Effective reduction of $\mathcal{L}(v_0)$. Since we do not need the whole list $\mathcal{L}(v_0)$, while v_0 is a cut vertex, at any reduction we can efficiently maintain only $\mathcal{C}'$ instead of $\mathcal{L}(v_0)$. Thus at every reduction of $\mathcal{L}(v_0)$ we just mark back with False all the colours of $\mathcal{L}(u_0)$ in $\mathcal{C}'$ and do not change $\mathcal{L}(v_0)$ yet. We will really change $\mathcal{L}(v_0)$ only when we cut off all the leaf blocks pending at v_0 and when v_0 becomes a non-cut vertex. By definition of $\mathcal{B}$ we will do such change only once for v_0, so the total time of reducing lists for all cut vertices will take a linear time. After such "fake" reduction of $\mathcal{L}(v_0)$ we colour all vertices in $B \setminus \{v_0\}$ and delete them from G. Then we go back to Step 2 B) and find other leaf block in the reduced graph.

Thus Step 3 and Step 4 work in $O(|B|)$ time for each particular block B and to maintain $\mathcal{C}'$ for each cut vertex in total takes $O(|\mathcal{L}|)$ time. Adding up all above together, we see that the algorithm works in $O(|G|+|\mathcal{L}|)$ time.

4 Constructive proof of the Brooks theorem

Δ-Colouring of a two-connected graph which is neither a clique nor an odd cycle. We describe an algorithm different from Step 4 III. Here we may suppose that all vertices in $B \setminus \{v_0\}$ (v_0 is the unique cut vertex in B) have the same list of admissible colours. Hence all the vertices in $B \setminus v_0$ have the same degree.
To this moment we know that B is a block, i.e. two-connected graph. Moreover we have found a depth-first search spanning tree for B (it is just a part of depth-first search tree for G). And also we have described a procedure of *recursive deletion of vertices*. So using all of this we can now describe a quite simple algorithm for colouring of G. The only difficult place in this algorithm is the number of combinatorial cases we are considering. So the description of the algorithm will consist of working out these cases, as follows.

- Suppose we have found two non-adjacent vertices, say u and v, in $B \setminus \{v_0\}$ such that if we do the following:
  - colour u and v with the same colour
  - delete this colour from $\mathcal{L}(w)$ for each neighbour w of v or u
  - delete both u and v from G,

  then the remaining graph have a *recursive deletion of vertices*. We call such vertices u and v a *good pair*. Thus if we have found a good pair then we can easily colour G in linear time.
- If $deg_G(u) = 2$ for an $u \in B \setminus \{v_0\}$, then B is a cycle and we know that B is not an odd cycle. So it is even. Let us colour $B \setminus \{v_0\}$ then delete from $\mathcal{L}(v_0)$ the colour of its two neighbours in B and then colour the remaining graph, since it has *recursive deletion of vertices*.
- Two descendant neighbours of a vertex u in the depth-first search spanning tree, i.e. normal tree, with v_0 as a root, are a *good pair*. Indeed, one can easily derive that these two descendants are not adjacent from the fact that a tree with root v_0 is normal. Also one can see that if we delete from B these two descendants, then in the remaining graph there always will be a path from u to any other vertex. These two things show the goodness of the pair.

– If our depth-first search spanning tree is not a path, then we can efficiently, i.e. in linear in block's size time, find such a pair. In this case we are done.
– So our tree is just a path $w_0, w_1, w_2, \ldots, w_{k-1}, v_0$, and, as above, v_0 is the unique cut vertex of B for G. For convenience let $v_0 = w_k$. Consider the closest to w_0 vertex, say w_t, on the path, which is not adjacent to w_0, and so $deg_G(w_0) \geq t-1$.
  1. The pair (w_0, w_t) is *good* when either there is a vertex among $w_1, w_2, \ldots, w_{t-1}$ which is adjacent to something else than w_0, w_1, $\ldots$, w_t, or $t+1 \leq k$ and w_{t+1} is adjacent to w_0.
     This is true, as w_0 and w_t are not adjacent and there cannot be more than two components of connectivity in the graph $B \setminus \{w_0, w_t\}$, since w_0 is a leaf in the normal tree (the vertices w_1, w_2, $\ldots$, w_{t-1} will be connected in $B \setminus \{w_0, w_t\}$ and so will be v_0 with remaining vertices).
  2. Suppose $deg_G(w_0) \geq t+1$, then w_1 is adjacent to a w_l with $l \geq t+1$, since $deg_G(w_1) = deg_G(w_0) \geq t+1$. By case 1 above, we have found a good pair.
  3. If $t-1 = deg_G(w_0)$, then w_{t-1} is not adjacent to some w_r with $r < t-1$, as $deg_G(w_0) = deg_G(w_{t-1})$ and w_{t-1} is adjacent to w_t. Hence w_r and w_{t-1} are a *good pair*, since w_r and w_{t-1} are both adjacent to w_0 and w_t.
  4. Suppose $t = deg_G(w_0)$ and that we are not in case 1. Hence w_0, w_1, $\ldots$, w_{t-1} and $w_1, \ldots, w_t$ form cliques, as $t = deg_G(w_0) = \cdots = deg_G(w_t)$ and no w_i with $1 \leq i \leq t-1$ is adjacent to w_j with $j \geq t+1$. Also $k \geq t+2$, as w_0 is not adjacent to w_t and w_{t+1}, but there is a vertex w_i adjacent to w_0 with $i > t$. We can assume $i > t+1$, since the possibility, when $i = t+1$, we have considered in the case 1. Thus, as $deg_G(w_t) = deg_G(w_0) > 2$, w_{t-1} and w_{t+1} are a *good pair*.

Remark 6. The same algorithm works for the Δ-colouring of a 2-connected graph H. Indeed, we can take any vertex of H as a cut vertex v_0 and we can easily colour H in linear in the size of H time if there is a vertex u with $deg_H(u) < \Delta(H)$.

Remark 7. This algorithm also provides a proof of Brooks Theorem [5] for two-connected graphs, since if the graph is not a clique or an odd cycle we can apply the algorithm and get the proper colouring.

Acknowledgement. We thank Oleg Borodin and Alexander Kostochka for rather helpful pointers to the literature.

References

1. J. A. Bondy, U. S. R. Murty, *Graph Theory with Applications*, Elsevier, New York, 1976.
2. O. V. Borodin, *A chromatic criteria for degree assignment, IV* USSR conference on the theoretical cybernetics problems, in Novosibirsk, proceeding reports, 127-128 (1977) (in Russian)

3. O. V. Borodin, *Problems of colouring and of covering the vertex set of a graph by induced subgraphs,* Ph.D. Thesis, Novosibirsk State University, Novosibirsk, 1979 (in Russian).
4. O. V. Borodin, A. V. Kostochka, *On an upper bound on a graph's chromatic number, depending on the graphs's degree and density* J. Comb. Th. (B) 23 (1977), p. 247-250.
5. R. L. Brooks, *On colouring the nodes of network*, Proc. Cambridge Phil. Soc. 37 (1941) 194–197.
6. R. Diestel *Graph Theory*, Springer, Berlin Heidelberg New York, 2000.
7. P. Erdős, A. L. Rubin, H. Taylor, *Choosability in graphs*, in: Proc. West-Coast Conf. on Combinatorics, Graph Theory and Computing, Arcata, California, Congr. Numer. XXVI (1979) 125–157.
8. T. Gallai, *Kritische Graphen I,* Publ. Math. Inst. Hung. Acad. Sci. 8 (1963) p. 373-395.
9. T. R. Jensen, B. Toft, *Graph Colouring Problems*, John Wiley & Sons, New York, 1995.
10. Kawarabayashi, Ken-ichi, B. Mohar, *List-color-critical graphs on a fixed surface*, SODA '09: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, 1156–1165, New York 2009.
11. A. Kostochka, M. Stiebitz, *A list version of Dirac's Theorem*, J. Graph Th. 39 (2002).
12. A. Kostochka, M. Stiebitz, B. Wirth, *The colour theorems of Brooks and Gallai extended*, Discrete Math 162 299-303 (1996)
13. J. Kratochvíl, Z. Tuza, M. Voigt, *New trends in the theory of graph colorings: choosability and list coloring*, 183–197, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 49, Amer. Math. Soc., Providence, RI, 1999.
14. L. Lovász, *Three short proofs in graph theory*, J. Comb. Th.(B) 19(1975), 269–271.
15. B. Reed, *A strengthening of Brooks' theorem,* J. Comb. Th. (B) V. 76, (1999) p. 136 - 149.
16. S. Skulrattanakulchai, *Δ-List Vertex Coloring in Linear Time and Space,* Information Processing Letters, Vol. 98 , Issue 3 (May 2006), 101–106.
17. R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput. 1 (2) (1972) 146–160.
18. Z. Tuza, *Graph colorings with local constraints—a survey*, Discussiones Math. Graph Th. 17(2) (1997), 161–228
19. V. Vizing, *Colouring the vertices of a graph in prescribed colours.*, Metody Diskret. Anal. v Teorii Kodov i Schem 29 (1976), 3–10 (In Russian)
20. `http://www1.spms.ntu.edu.sg/~ngravin/code/Dcols.tgz`

Appendix. Proof of theorem 2.

Sufficiency. Suppose we have found such a map $\mathcal{S}$ and by a contradiction G has a $\mathcal{L}$-colouring $\mathcal{C}$. It is clear that G can not be a block, since otherwise by the Brooks Theorem it has no colouring. Let us consider a leaf block B in the tree of blocks and cut vertices. As we have mentioned in step 4 of the algorithm, the cut vertex v_0 of this block could not have a colour from the set $\mathcal{S}(v_0, B)$ in $\mathcal{C}$, otherwise we get a contradiction with Brooks Theorem for $G(B)$. Hence for the graph $G' = G((V \setminus B) \bigcup \{v_0\})$ and assignment $\mathcal{L}'$:

- $\mathcal{L}'(v) = \mathcal{L}(v), v \in V \setminus B$
- $\mathcal{L}'(v_0) = \mathcal{L}(v_0) \setminus \mathcal{S}(v_0, B)$

the map $\mathcal{S}'$ satisfies all condition from the theorem, where $\mathcal{S}'$ is a restriction of $\mathcal{S}$ to G'. Also $\mathcal{C}$ restricted to $V(G')$ is an $\mathcal{L}'$-colouring of G'. Thus by the method of infinite descent we arrive at contradiction.

Necessity. Suppose we have a graph G and a list assignment $\mathcal{L}$ such that G is non-$\mathcal{L}$-colourable. Let us see how the algorithm works for G and $\mathcal{L}$. It should give us a negative answer. So it should work till the graph G becomes empty. It means that we eliminate according to the case II of step 4 one by one all the blocks of G. Thus any block of G is either a clique or an odd cycle, and by setting $\mathcal{S}(v_0, B)$ to be the set of colours deleted from $\mathcal{L}(v_0)$ at the step of deleting B, we get a map in the theorem.