RAIRO-Oper. Res. 43 (2009) 27–33
DOI: 10.1051/ro/2009003

RAIRO Operations Research
www.rairo-ro.org

LOCALLY BOUNDED k-COLORINGS OF TREES

C. Bentz[1] and C. Picouleau[2]

Abstract. Given a tree T with n vertices, we show, by using a dynamic programming approach, that the problem of finding a 3-coloring of T respecting local (*i.e.*, associated with p prespecified subsets of vertices) color bounds can be solved in $O(n^{6p-1}\log n)$ time. We also show that our algorithm can be adapted to the case of k-colorings for fixed k.

Keywords. Bounded graph coloring, tree, dynamic programming.

Mathematics Subject Classification. 05C15, 90C39.

1. Introduction

An important combinatorial problem consists in coloring an undirected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. A *k-coloring* is a mapping $f : V \to \{1, \ldots, k\}$ such that $f(x) \neq f(y)$ for every edge $[x, y] \in E$. In other words, f is a partition of V such that each class $f_i, 1 \leq i \leq k$, is an independent set. It is well known that, for every fixed $k \geq 3$, the problem of deciding if a k-coloring exists for a graph G is NP-complete (see [10]). Note that, for $k = 2$, a 2-coloring exists if and only if G is bipartite, and a 2-coloring of a bipartite graph can be obtained in linear time by using basic algorithmic arguments.

A k-coloring f is *equitable* if the size of each color class f_i satisfies $\lfloor n/k \rfloor \leq |f_i| \leq \lceil n/k \rceil$. The equitable k-coloring problem, *i.e.*, the problem of deciding whether an equitable k-coloring exists, has been studied for a long time and a well known result of Hajnal and Szemerédi is proved in [12]: G has an equitable k-coloring for any $k \geq \Delta(G) + 1$ ($\Delta(G)$ denotes the maximum degree of a vertex of G); this bound is sharp. A related problem is the b-bounded k-coloring problem defined as follows: given G and a nonnegative integer b, decide whether a k-coloring f such that $|f_i| \leq b$ for all colors $i, 1 \leq i \leq k$, exists. This problem was introduced in [13], and in [4] authors prove that the b-bounded 3-coloring problem is NP-complete for bipartite graphs (actually, their proof even shows the NP-completeness of the

Received June 19, 2007. Accepted July 28, 2008.

[1] CEDRIC, CNAM, Paris, France; cedric.bentz@cnam.fr

[2] CEDRIC, CNAM, Paris, France; chp@cnam.fr

Article published by EDP Sciences

equitable 3-coloring problem in bipartite graphs), while it is tractable in this case if b is fixed [4,13]. For trees, both problems, equitable and b-bounded k-coloring, are studied in [1,5–7,14]; a linear-time algorithm that solves the b-bounded k-coloring problem is designed in the first reference. In [3] it is shown that the equitable k-coloring problem is polynomial-time solvable in bounded tree-width graphs but that the precolored version becomes NP-complete, even in trees.

Actually, these two problems are special cases of the following problem MINkC: find a k-coloring f that minimizes $F(|f_1|,\ldots,|f_k|)$, where $F(\cdot)$ is a function that can be computed in polynomial time and $|f_i|$ is the size of the ith color class, for each i. Indeed, the equitable k-coloring problem is equivalent to finding a k-coloring f that minimizes the following function: $F(|f_1|,\ldots,|f_k|)=0$ if $\lfloor n/k\rfloor \leq |f_i| \leq \lceil n/k\rceil$ for each i, $F(|f_1|,\ldots,|f_k|)=1$ otherwise. The b-bounded k-coloring problem consists in finding a k-coloring f that minimizes the following function: $F(|f_1|,\ldots,|f_k|)=0$ if $|f_i|\leq b$ for each i, $F(|f_1|,\ldots,|f_k|)=1$ otherwise. Another subproblem of MINkC consists in finding a k-coloring such that $|f_i|\leq b_i$ for each i, *i.e.*, that minimizes the function: $F(|f_1|,\ldots,|f_k|)=0$ if $|f_i|\leq b_i$ for each i, $F(|f_1|,\ldots,|f_k|)=1$ otherwise. Note that in all these problems the b_i's are part of the input. This corresponds to a variant of the problem studied by Baker and Coffman [1]: in their problem, each color class corresponds to a set of tasks (nodes) to be executed at a given time; therefore, the cardinality of each color class is bounded by the total number of processors. In this new variant, each color class can have a different bound: it emulates the case where some processors may be unavailable at some times (because, for instance, they are in maintenance mode).

Moreover, when k is fixed, MINkC is equivalent to the following problem EQkC: given k positive integers $n_1,\ldots,n_k$, find a k-coloring such that $|f_i|=n_i$ for each i, with $\sum_{i=1}^{k} n_i = n$. Indeed, one can solve an instance of MINkC by testing, for every k-tuple $(n_1,\ldots,n_k)$ with $\sum_{i=1}^{k} n_i = n$, whether there exists a k-coloring f such that $|f_i|=n_i$ for each i (by solving an instance of EQkC), and, for each such coloring, by computing its cost, $F(|f_1|,\ldots,|f_k|)$. The optimal solution is the best solution obtained during this process. A polynomial-time dynamic programming algorithm was designed in [11] to solve EQkC in bounded tree-width graphs when k is fixed, by enumerating all the k-tuples $(n_1,\ldots,n_k)$ such that there exists a k-coloring f that satisfies $|f_i|=n_i$ for each i (the number of such k-tuples is $O(n^{k-1})$, while the number of k-colorings is $O(k^n)$). Also note that the edge-coloring variant of EQkC in trees was solved by de Werra *et al.* in [9].

In this paper, we consider a generalization of EQkC, involving "local" (instead of "global") bounds on the cardinalities of the color classes. More precisely, given a partition $V_1,\ldots,V_p$ of the vertex set V and pk integral bounds $n_{11},\ldots,n_{1k},n_{21},\ldots,n_{2k},\ldots,n_{p1},\ldots,n_{pk}$ such that $\sum_{j=1}^{k} n_{ij} = |V_i|$ for each i, the problem EQpkC consists in deciding whether there exists a k-coloring such that, for each $i\in\{1,\ldots,p\}$ and for each $j\in\{1,\ldots,k\}$, the number of vertices having color j in V_i is n_{ij} (note that EQkC is equivalent to EQ1kC). One can also define the associated variant of MINkC, named MINpkC: when k and p are fixed, EQpkC and MINpkC are equivalent. A natural subproblem of MINpkC (but not

of MINkC) is the following generalization of the problem studied in [1]: given a partition $V_1, \ldots, V_p$ of V and pk bounds $b_{11}, \ldots, b_{pk}$, decide whether there exists a k-coloring such that, for each i and j, the number of vertices having color j in V_i is *at most* b_{ij}. This corresponds to a variant of the problem studied by Baker and Coffman where (i) the number of available processors can vary over time (because, for instance, some of them may break down) and (ii) each processor has an associated type, and can execute only tasks of this type (the number of processors of type i at time j is then b_{ij}).

Our main result is a polynomial-time algorithm to solve EQpkC in trees when both p and k are fixed. For the sake of simplicity we first describe precisely a dynamic programming approach that yields a polynomial-time algorithm for solving EQ13C (and thus MIN13C) in trees. Then, we show briefly how this approach can be generalized to solve EQpkC. Note that, it is not clear at all whether the algorithm given in [11], which is much more complicated than ours when G is a tree, could be adapted to EQpkC.

2. Notations and preliminaries

We consider a tree $T = (V, E)$ with n vertices. We begin by rooting the tree at an arbitrary vertex r. For any vertex v we denote by T_v the subtree rooted at v, so the whole rooted tree is T_r. The *level* of a vertex in T_r is its distance from r. Let d_v denote the number of children of a vertex v in T_r, and let these children be $u_1, \ldots, u_{d_v}$. We also denote by $T_v(i)$ the partial subtree of T_v containing the vertex v and the subtrees $T_{u_1}, \ldots, T_{u_i}$.

In the dynamic programming algorithm, we proceed level by level, starting with the vertices with the highest level. This way, when we deal with a vertex v in level l, we know that all its children, lying in level $l + 1$, have been examined.

Throughout this paper, a feasible 3-coloring corresponds to a triple (a, b, c) where a, b, c are the cardinalities of the three color classes (several 3-colorings can correspond to the same triple): we shall call it a *colorable triple*.

We define $X(v, i)$ as the set of all the possible colorable triples for the subtree $T_v(i)$. For a vertex v, we will compute successively the sets $X(v, 1), X(v, 2), \ldots, X(v, d_v)$. At the end of our algorithm, we will obtain $X(r, d_r)$, *i.e.*, the set of all possible colorable triples of the initial tree T.

Now, for each $i > 1$, given $X(v, 1), \ldots, X(v, i - 1)$, all we need is a way to compute $X(v, i)$. In fact, for each $j \in \{1, \ldots, i - 1\}$ and for each colorable triple in $X(v, j)$, we also need to keep track of the colors in which v may be colored in colorings associated with this colorable triple (there may be several possible colors), which implies that the same information must be available for the possible colorable triples of the subtree T_{u_j}, for each child u_j of v (assume that this data is stored, with the colorings, in the X's). So, assume we know all that is needed (we shall see later how to initialize the induction), and want to compute $X(v, i)$ for some v and $i \geq 2$. More precisely, what is currently known is, on the one hand, the set of all the possible colorable triples for the subtree $T_v(i - 1)$, and, for each

colorable triple, the set of possible colors for v. On the other hand, for each child u_j of v, we know the set of all possible colorable triples for the subtree T_{u_j} (as well as the set of possible colors for u_j in each colorable triple). We want to extend these colorable triples to valid colorable triples for the partial subtree $T_v(i)$. Thus, we have to combine the colorable triples in $X(v, i-1)$ and in $X(u_i, d_{u_i})$: this point is explained in the next section. First, we need a few more definitions.

Given a vertex v and a colorable triple $\xi \in X(v, i)$, we denote by λ_ξ the 3 dimensional vector whose ith component (denoted by $\lambda_\xi(i)$) is equal to 1 if v can be colored with the ith color in ξ and to 0 otherwise. It can be viewed as a characteristic vector: it contains the list of the color classes the vertex v can belong to in colorings associated with this triple.

3. The dynamic programming algorithm

We describe informally our approach. The main idea is that any colorable triple in $X(v, i)$, $i > 1$, can be obtained by pairing the color bounds of a colorable triple in $X(v, i-1)$ and of a colorable triple in $X(u_i, d_{u_i})$, and then merging them. By trying all the possible combinations, we are sure to obtain all the possible colorable triples in $X(v, i)$. However, some combinations will lead to the same resulting triple, so we have to merge them as well (see below for a detailed explanation). Moreover, some combinations are forbidden, because v and u_i belong to two color classes merged together (and thus we do not obtain a valid coloring, since there is an edge $[u_i, v]$). An important remark is that we do not compute $X(v, d_v)$ directly by using $X(u_1, d_{u_1}), \ldots, X(u_{d_v}, d_{u_{d_v}})$, *i.e.*, without computing $X(v, 1), \ldots, X(v, d_v - 1)$ first. It would be interesting to determine whether such an approach is possible. Also note that the computations of $X(v, i)$ and $X(w, j)$ for two distinct vertices v and w on the same level are independent; therefore, one can compute them in parallel (if several processors are available, for instance), and reduce the running time by a multiplicative factor of $n/h(T)$ ($h(T)$ being the height of the tree T).

Formally, the *main step*, *i.e.*, the computation of $X(v, i)$, $i > 1$, can be stated as follows:

Given $X(v, i-1)$ and $X(u_i, d_{u_i})$,

$$
\begin{aligned}
X(v,i) \quad = \quad & \Big\{ (\xi = (a_1 + b_1, a_2 + b_2, a_3 + b_3), \lambda_\xi) \text{ where} \\
& (a_1, a_2, a_3) \in X(v, i-1),\ (b_1, b_2, b_3) \in X(u_i, d_{u_i}) \text{ and} \\
& \forall j \in \{1,2,3\}\ \lambda_\xi(j) = \min(1, \lambda_{(a_1,a_2,a_3)}(j) \times \sum_{h \neq j} \lambda_{(b_1,b_2,b_3)}(h)) \quad (1) \\
& \text{with } \sum_{j=1}^{3} \lambda_\xi(j) \geq 1 \Big\}.
\end{aligned}
$$

Now, let us show how to initialize the induction (*initialization step*). For a leaf v, we have three possible colorable triples: $\xi_1 = (1,0,0) = \lambda_{\xi_1}$, $\xi_2 = (0,1,0) = \lambda_{\xi_2}$ and $\xi_3 = (0,0,1) = \lambda_{\xi_3}$. It remains to describe how we compute $X(v,1)$ for a non-leaf vertex v. Given a colorable triple $\xi = (a,b,c) \in X(u_1, d_{u_1})$, we can define at most three potential colorable triples for $X(v,1)$: $\xi_1 = (a+1,b,c)$, $\xi_2 = (a,b+1,c)$ and $\xi_3 = (a,b,c+1)$. Here are the conditions for these three combinations to define feasible colorable triples:

- if $\lambda_\xi \neq (1,0,0)$ then ξ_1 is feasible, so we add ξ_1 to $X(v,1)$ and set $\lambda_{\xi_1} = (1,0,0)$;
- if $\lambda_\xi \neq (0,1,0)$ then ξ_2 is feasible, so we add ξ_2 to $X(v,1)$ and set $\lambda_{\xi_2} = (0,1,0)$;
- if $\lambda_\xi \neq (0,0,1)$ then ξ_3 is feasible, so we add ξ_3 to $X(v,1)$ and set $\lambda_{\xi_3} = (0,0,1)$.

Note that, in both steps, each time we examine a new combination, we have to check whether it exists or not among the colorable triples already computed in $X(v,i)$ (*i.e.*, we must not add twice a given triple obtained by two different combinations of colorable triples, in order to store only a reasonable number of triples). Therefore, we go through the colorable triples already computed in $X(v,i)$, and if we find a triple ξ identical to our current combination ξ', we set $\lambda_\xi(i) \leftarrow \min(1, \lambda_\xi(i) + \lambda_{\xi'}(i))$ for each i and do not add ξ' in $X(v,i)$.

As an example, consider a vertex u with two possible colorable triples $\xi_1 = (\alpha,\beta,\gamma)$ and $\xi_2 = (\alpha,\beta+1,\gamma-1)$ such that (for instance) $\lambda_{\xi_1} = \lambda_{\xi_2} = (1,1,1)$. If we want to extend these colorable triples for u to colorable triples for its father v, we can, for example, start with ξ_1 and color v with the second color (*i.e.*, the color associated with the color class of cardinality β): this way, we obtain a triple $\xi_3 = (\alpha,\beta+1,\gamma)$ with $\lambda_{\xi_3} = (0,1,0)$. Then, we consider ξ_2 and color v with the third color: we obtain a triple $\xi_4 = (\alpha,\beta+1,\gamma)$ with $\lambda_{\xi_4} = (0,0,1)$. Since $\xi_3 = \xi_4$, we cannot keep both of them: we store only ξ_3 and set $\lambda_{\xi_3} = (0,1,1)$.

4. Validity and complexity

In this section, we show the following theorem:

Theorem 1. *Given a tree T, the algorithm defined in Section 3 enumerates all the triples (a,b,c) such that there exists a 3-coloring f of T verifying $|f_1| = a$, $|f_2| = b$ and $|f_3| = c$. Moreover, it runs in $O(n^5 \log n)$ time.*

Proof. We first consider the initialization step. If $\lambda_\xi = (1,0,0)$, then $\xi_1 = (a+1,b,c)$ cannot be associated with a feasible coloring, because v and u_1 would belong to the same color class in this case (both being in the first one). In the other cases, u_1 can belong to one of the two other color classes, and thus we obtain a feasible coloring. The same argument can be used for ξ_2, and hence for ξ_3. Now, let us consider the main step. Formula (1) ensures that we examine all the possible combinations of a colorable triple in $X(v,i-1)$ and of a colorable triple in $X(u_i, d_{u_i})$. However, to check if a combination corresponds to a feasible

coloring, we must also verify that v can belong to at least one color class (this is why feasible combinations must satisfy $\sum_{j=1}^{3} \lambda_\xi(j) \geq 1$). Eventually, we detail the computation of $\lambda_\xi(j)$. The jth color class in ξ being obtained by merging the jth color class in (a_1, a_2, a_3) and the jth color class in (b_1, b_2, b_3), v can belong to this color class (*i.e.*, $\lambda_\xi(j) = 1$) if and only if v can belong to the jth color class in (a_1, a_2, a_3) (*i.e.*, $\lambda_{(a_1,a_2,a_3)}(j) = 1$) and u_i can belong to a color class different from the jth in (b_1, b_2, b_3) (*i.e.*, $\sum_{h \neq j} \lambda_{(b_1,b_2,b_3)}(h) \geq 1$).

Now, let us examine the running time. Sets $X(v, i)$ are represented by balanced binary trees like Red-Black trees or AVL (see [8]). So, denoting by $n_v(i)$ the number of vertices of $T_v(i)$, and considering that the number of triples (a, b, c) satisfying $a+b+c = n$ is $O(n^2)$, basic operations like search or insertion take $O(\log n_v^2(i)) = O(\log n_v(i))$ time in the worst case. If $\nu_v(i)$ denotes the number of colorable triples in $X(v, i)$, then the computation of $X(v, i)$, $i > 1$, from $X(v, i-1)$ in the main step takes $O(\nu_v(i-1) \cdot \nu_{u_i}(d_{u_i}) \cdot \log \nu_v(i)) = O(n_v^4(i) \cdot \log n_v(i)) = O(n^4 \log n)$ time (since checking if a new triple ξ is already in the current $X(v, i)$ takes $O(\log n_v^2(i))$ time). The initialization step takes $O(\nu_v(1) \cdot \log \nu_v(1)) = O(n_v^2(1) \cdot \log n_v(1))$ time. Note that since both the main and the initialization steps are performed $O(n)$ times, the whole running time of our algorithm is bounded by $O(n^5 \log n)$. □

Corollary 1. *EQ3C can be solved in $O(n^5 \log n)$ time in trees, by using the algorithm given in Section 3.*

5. Generalization

We describe in this section how the algorithm given in Section 3 to solve EQ3C can be adapted to EQpkC when both p and k are fixed. The general outline of the approach is similar to the case where $k = 3$ and $p = 1$; we consider colorable pk-tuples instead of colorable triples, and the size of a characteristic vector λ_ξ is k. For $1 \leq j \leq k$ and $1 \leq i \leq p$, the $(k(i-1)+j)$th component of a given colorable pk-tuple equals the number of vertices of V_i that have color j. The main step is the same as for $p = 1$ and $k = 3$. The initialization step is as follows. For a leaf $v \in V_i$, the jth of the k colorable pk-tuples associated with v has all its components equal to 0, except from its $(k(i-1)+j)$th component, which is equal to 1. For a non-leaf vertex $v \in V_i$, given a colorable pk-tuple $\xi \in X(u_1, d_{u_1})$, the jth of the k potential colorable pk-tuples $\xi_1, \ldots, \xi_k$ for $X(v, 1)$ is obtained by increasing the $(k(i-1)+j)$th component of ξ by one.

It now remains to examine the whole running time. The number of pk-tuples $(a_{11}, \ldots, a_{pk})$ such that $a_{11} + \cdots + a_{pk} = n$ is at most $O(n^{pk-1})$. Moreover, using a Red-Black tree (see Sect. 4), each checking can be implemented within $O\left(\log\left(n^{k-1}\right)\right)$ time. Hence, the whole running time is $O\left(n\left(n^{pk-1}\right)\left(n^{pk-1}\right)\log\left(n^{pk-1}\right)\right) = O\left(n^{2pk-1}\log\left(n^{pk-1}\right)\right)$, *i.e.*, $O(n^{2pk-1}\log n)$ for fixed p, k. An interesting open problem would be to determine whether this complexity may be improved or not.

6. Concluding remarks

It can be noticed that our approach does not solve efficiently the case where k is not fixed, but the problem of enumerating all the valid colorable k-tuples cannot be tractable in this case, since the size of the solution (*i.e.*, the number of k-tuples associated with feasible k-colorings) would not be polynomial in n.

However, it is possible that, for particular problems (such as EQkC), one can obtain faster algorithms. In particular, some of these problems might be solved without enumerating all the colorable k-tuples, and so one may hope to obtain efficient algorithms even for arbitrary values of k.

Since it has been shown that EQkC can be solved in polynomial time in bounded tree-width graphs when k is fixed, it would be interesting to establish whether our approach can be adapted to EQpkC.

References

[1] B.S. Baker and E.G. Coffman, Mutual exclusion scheduling. *Theor. Comput. Sci.* **162** (1996) 225–243.

[2] C. Berge, *Graphes*. Gauthier-Villars, Paris (1983).

[3] H.L. Bodlaender and F.V. Fomin, Equitable colorings of bounded treewidth graphs. *Theor. Comput. Sci.* **349** (2005) 22–30.

[4] H.L. Bodlaender and K. Jansen, Restrictions of graph partition problems. Part I. *Theor. Comput. Sci.* **148** (1995) 93–109.

[5] B. Bollobás and R. Guy, Equitable and proportional coloring of trees. *J. Comb. Theory, Ser. B* **34** (1983) 177–186.

[6] B.-L. Chen and K.-W. Lih, A note on the m-bounded chromatic number of a tree. *Eur. J. Comb.* **14** (1993) 311–312.

[7] B.-L. Chen and K.-W. Lih, Equitable Coloring of Trees. *J. Comb. Theory, Ser. B* **61** (1994) 83–87.

[8] T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to Algorithms*. The MIT press, (2001).

[9] D. de Werra, A. Hertz, D. Kobler and N.V.R. Mahadev, Feasible edge colorings of trees with cardinality constraints. *Discrete Math.* **222** (2000) 61–72.

[10] M.R. Garey and D.S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. Ed. Freeman New York (1979).

[11] S. Gravier, D. Kobler and W. Kubiak, Complexity of list coloring problems with a fixed total number of colors. *Discrete Appl. Math.* **117** (2002) 65–79.

[12] A. Hajnal and E. Szemerédi, Proof of a conjecture of P. Erdős. *Combinatorial Theory and its Applications*, Vol. II, North-Holland, Amsterdam (1970) 601–623.

[13] P. Hansen, A. Hertz and J. Kuplinsky, Bounded vertex colorings of graphs. *Discrete Math.* **111** (1993) 305–312.

[14] M. Jarvis and B. Zhou, Bounded vertex coloring of trees. *Discrete Math.* **232** (2001) 145–151.