On the Complexity of Exact Algorithm for $L(2,1)$-labeling of Graphs

Konstanty Junosza-Szaniawski, Paweł Rzążewski*
{k.szaniawski, p.rzazewski}@mini.pw.edu.pl

Warsaw University of Technology
Faculty of Mathematics and Information Science
Pl. Politechniki 1 , 00-661 Warsaw, Poland

Abstract

$L(2,1)$-labeling is a graph coloring model inspired by a frequency assignment in telecommunication. It asks for such a labeling of vertices with nonnegative integers that adjacent vertices get labels that differ by at least 2 and vertices in distance 2 get different labels. It is known that for any $k \geq 4$ it is NP-complete to determine if a graph has a $L(2,1)$-labeling with no label greater than k.
In this paper we present a new bound on complexity of an algorithm for finding an optimal $L(2,1)$-labeling (i.e. an $L(2,1)$-labeling in which the largest label is the least possible). We improve the upper complexity bound of the algorithm from $O^*(3.5615..^n)$ to $O^*(3.2360..^n)$. Moreover, we establish a lower complexity bound of the presented algorithm, which is $\Omega^*(3.0731..^n)$.

1 Introduction

The channel assignment problem is the problem of assigning channels of frequency (represented by nonnegative integers) to radio transmitters, so that interfering transmitters are assigned channels whose difference is not

*The authors have been working on this paper when visiting KAM at Charles University in Prague in October 2010.

in a set of forbidden values. Hale [6] formulated this problem in terms of so-called T-coloring of graphs. Roberts [5] proposed a modification of this problem, which is called an $L(2,1)$-labeling problem. It asks for such a labeling of vertices of a graph with nonnegative integer labels, that no vertices in distance 2 in the graph have the same labels and labels of adjacent vertices differ by at least 2.

By $L(2,1)(G)$ we denote the $L(2,1)$-span of G, which is the smallest value of k that guarantees the existence of an $L(2,1)$-labeling with no label exceeding k. For any fixed $k \geq 4$, determining if $L(2,1)(G) \leq k$ was proven to be NP-complete by Fiala *et al.* [4]. It remains NP-complete even for regular graphs (Fiala, Kratochvíl [3]) and planar graphs (Eggeman *et al.* [2]).

Král [9] presented an algorithm for a more general problem, which can determine $L(2,1)$-span of any given graph in time $O^*(4^n)$. Havet *et al.* [7] presented a faster algorithm for computing $L(2,1)(G)$. The complexity of this algorithm is bounded by the number of ordered triples of disjoint sets of vertices such that the first of those sets is a 2-packing, i.e. a set in which all the vertices are in distance at least 3 from each other. Havet *et al.* [7] proved that the number of k-element 2-packings in a connected graph on n vertices does not exceed $2^k \binom{n/2}{k}$. With this bound they showed that their algorithm finds $L(2,1)$-span of any connected graph on n vertices in time $O^*(15^{\frac{n}{2}}) = O^*(3.8729..^n)$.

In [8] we presented an improved version of the algorithm by Havet *et al.* [7] (we remind this algorithm in Section 3). The improvement is a consequence of considering an additional dependency that decreases the number of iterations of the main loop of the algorithm. In the same paper we presented a better bound on the number of k-element 2-packings in a connected graph on n vertices, which is $\binom{n-k+1}{k}$. Those results lead to the time complexity bound of $O^*(3.5615..^n)$.

In this paper, in Section 3, we provide a more careful analysis of the algorithm presented in [8]. We notice that its complexity is in fact determined by the number of triples (U, Y, A) of disjoint sets of vertices such that both sets U and Y are 2-packings and no vertex from U has a neighbor in Y. In Section 4 we present an algorithm generating such triples. It follows from the analysis of this algorithm that the number of the triples and therefore the time complexity of our algorithm for $L(2,1)$-labeling is bounded by $O^*(3.2360..^n)$.

In Section 5 we present a lower bound on the complexity of the algorithm from [8], which is $\Omega^*(3.0731..^n)$.

2 Preliminaries

All graphs appearing in this paper are connected. We can restrict ourselves to this class of graphs because a labeling of a disconnected graph can be obtained by labeling each of its components separately.

Let $dist_G(x, y)$ be the *distance* between vertices x and y in a graph G, which is the length of a shortest path joining x and y.

Formally an $L(2,1)$-*labeling* is defined as a function $f\colon V(G) \to \{0, 1, \ldots\}$, such that $\forall x, y \in V(G)\ dist_G(x, y) = 1 \Rightarrow |f(x) - f(y)| \geq 2$ and $\forall x, y \in V(G)\ dist_G(x, y) = 2 \Rightarrow |f(x) - f(y)| \geq 1$.

By an $L(2,1)$-span of a graph ($L(2,1)(G)$) we denote the smallest value of k guaranteeing the existence of a $L(2,1)$-labeling with no label exceeding k.

A set $X \subseteq V(G)$ is a *2-packing* in a graph G if all its vertices are in distance at least 3 from each other ($\forall x, y \in X\ dist_G(x, y) > 2$).

By the *neighborhood* of a set $X \subseteq V(G)$ we mean the set $N(X) = \{u \in V(G)\colon \exists v \in X\ (u, v) \in E(G)\}$. Moreover we define $N[X] = N(X) \cup X$.

A triple of disjoint subsets of vertices (U, Y, A) is called *legal* in a graph G, if U and Y are 2-packings in G, $Y \cap N_G[U] = \emptyset$ and $A \subseteq V(G) \setminus (U \cup Y)$.

We write $f(n) = O^*(g(n))$ (resp. $f(n) = \Omega^*(g(n))$) if there exists a polynomial $p(n)$ such that $f(n) \leq p(n)g(n)$ (resp. $f(n) \geq p(n)g(n)$) for all n greater than some n_0. We write $f(n) = \Theta^*(g(n))$ if $f(n) = \Omega^*(g(n))$ and $f(n) = O^*(g(n))$.

Let ϕ denote the *golden ratio*, i.e. $\phi = \frac{1+\sqrt{5}}{2}$.

3 Exact algorithm for $L(2,1)$ - labeling

Let us start with reminding an exact algorithm for $L(2,1)$-labeling from [8], which is an improved version of the algorithm presented originally by Havet *et al.* [7].

First, the algorithm marks all 2-packings as subsets that can be labeled with the label 0. Then it tries to extend a partial labeling by adding to it a set of vertices with the next label. More precisely, the algorithm iteratively marks all subsets of $V(G)$ that can be labeled with no label greater than i for $i = 1, \ldots, 2n$. (Note here that $L(2,1)(G)$ is smaller than $2n$, since labeling the vertices by distinct even integers is always a valid $L(2,1)$-labeling.) To do it efficiently, we introduce Boolean variables $Lab[X, Y, i]$ for all pairs of disjoint sets $X, Y \subseteq V(G)$, where Y is a 2-packing in G and $i \in \{0, \ldots, 2n\}$.

The value of $Lab[X, Y, i]$ is set *true* if and only if there exists a partial $L(2,1)$-labeling f with no label greater than i for X (i.e. $f\colon X \to \{0, 1, \ldots, i\}$), such that no vertex in $N(Y) \cap X$ has a label i. The values of $Lab[X, Y, i]$ for every 2-packing Y in G, $X \subseteq V(G) \setminus Y$ and $i \in \{0, \ldots, 2n\}$ are computed by dynamic programming using the following implication:

If $Lab[A, U, i-1]$ is *true* for a 2-packing U and a set $A \subseteq V(G) \setminus U$, then $Lab[U \cup A, Y, i]$ is *true* for any Y, which is a 2-packing in $G - (A \cup N[U])$.

Let us show this implication. If $Lab[A, U, i-1]$ is *true* for a 2-packing U and a set $A \subseteq V(G) \setminus U$, then there exists a partial labeling $f\colon X \to \{0, \ldots, i-1\}$ such that no vertex in $N(U) \cap A$ has the label $i-1$. Hence we can extend the labeling f by setting $f(v) = i$ for every $v \in U$. A labeling obtained in this way fulfills the condition to set the value of $Lab[U \cup A, Y, i]$ to *true* for any 2-packing Y in G, which is disjoint with A and with $N[U]$ (i.e. any 2-packing in $G - (A \cup N[U])$).

Notice that $Lab[V(G), \emptyset, i]$ is *true* if and only if there exists an $L(2,1)$-labeling of the graph G with no label exceeding i. Hence $L(2,1)(G) = \min\{i\colon\ Lab[V(G), \emptyset, i] \text{ is } true\}$.

The description and the proof of correctness of the algorithm **Improved-Find-L(2,1)-Span** can be found in [8].

Algorithm 1: Improved-Find-L(2,1)-Span

1 **foreach** Y *- 2-packing in* G **do**
2   **foreach** $X \subseteq V(G) \setminus Y,\ i = 0, \ldots, 2n$ **do** $Lab[X, Y, i] \leftarrow false$
3 **foreach** X *being a 2-packing in* G **do**
4   **foreach** Y *being a 2-packing in* $G - N[X]$ **do** $Lab[X, Y, 0] \leftarrow true$
5   **if** $X = V(G)$ **then return** "$L(2,1)(G) = 0$"
6 **for** $i \leftarrow 1$ **to** $2n$ **do**
7   **foreach** (U, Y, A) *being a legal triple in* G **do**
8   **if** $Lab[A, U, i-1]$ **then** $Lab[U \cup A, Y, i] \leftarrow true$
9   **if** $U \cup A = V(G)$ **then return** "$L(2,1)(G) = i$"

The following theorem is the main result of this paper:

Theorem 1. *The algorithm **Improved-Find-L(2,1)-Span** finds the $L(2,1)$-span of any connected graph in time $O^*((2\phi)^n) = O^*(3.2360..^n)$ and space $O^*(2.7320..^n)$, where n is the number of vertices of the graph.*

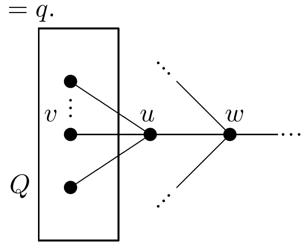
Proof. Notice that the complexity of the algorithm **Improved-Find-L(2,1)-Span** is determined by the operations in the main loop (lines 9-16) and all inner operations of this loop are performed in a polynomial time. Thus the time complexity of the algorithm **Improved-Find-L(2,1)-Span** (in the sense of the O^* notation) is determined by the time needed to generate all the legal triples in G. In Section 4 we show an algorithm that generates all such triples. In Theorem 2 we show that the number of legal triples is bounded by $O((2\phi)^n) = O(3.2360..^n)$ and they can be generated in time $O^*(3.2360..^n)$. Hence the computational complexity of the algorithm **Improved-Find-L(2,1)-Span** is bounded by $O^*((2\phi)^n) = O^*(3.2360..^n)$.

It was proven in [8] that the space complexity of the algorithm **Improved-Find-L(2,1)-Span** is bounded by $O^*(2.7320..^n)$. □

4 Generating the legal triples

In this section we present a recursive algorithm for generating all legal triples in a connected graph.

If the graph has at most 2 vertices, then all legal triples can be easily enumerated. For graphs with more vertices, the algorithm proceeds to its main part. In the beginning, the algorithm finds T – a spanning tree of G and then P, which is the longest path in T (the longest path can be found in polynomial time by the algorithm described in [1]). Let v, u, w denote the three consecutive vertices of P starting from one of the ends of P. Let Q denote the set of neighbors of u different than w. Notice that they are all leaves, because otherwise P would not be the longest path in T. Let $|Q| = q$.



The algorithm constructs the desired triples by branching on all vertices in Q simultaneously.

Suppose that none of the vertices in Q belongs to U or Y. Then each of them belongs or does not belong to A. After selecting vertices in $Q \cap A$ (2^q possibilities), we proceed with the graph $G - Q$ (this graph is connected).

Notice that if any of the vertices in Q belongs to $U \cup Y$, then $u \notin U \cup Y$, since $U \cup Y$ is an independent set.

If exactly one of the vertices (denote it by x) in Q belongs to U or Y ($2q$ possibilities), then we decide which vertices in $(Q \setminus \{x\}) \cup \{u\}$ belong to A (2^q possibilities) and proceed with the graph $G - (Q \cup \{u\})$ (this graph is connected).

If there are exactly two vertices in $Q \cap (U \cup Y)$ (denote them by x, y), one of them must be in U and the other one in Y ($q(q-1)$ possibilities). We decide which vertices in $(Q \setminus \{x, y\}) \cup \{u\}$ belong to A (2^{q-1} possibilities) and proceed with the graph $G - (Q \cup \{u\})$ (this graph is connected). Notice that $|Q \cap (U \cup Y)| \leq 2$, since each of the sets U and Y is a 2-packing in G.

Notice that the algorithm may generate some triples, that are not legal in G (e.g. because not all legal triples in T are legal in G), so it has to be checked before returning each triple. This task can be performed in a polynomial time. Since we have to keep the initial graph G to check if a generated triple is legal in G, in recursive calls we modify a copy $\widehat{G}$ of G.

Algorithm 2: Generate-Triples

Call : Generate-Triples(G, $\widehat{G}$, U, Y, A)
Arguments: Connected graph G, Connected $\widehat{G} \subseteq G$, Set U, Set Y, Set A

if $|V(\widehat{G})| \leq 2$ **then**
    foreach $v \in V(\widehat{G})$ *and* $S \subseteq V(\widehat{G}) \setminus \{v\}$ **do**
        if $(U \cup \{v\}, Y, A \cup S)$ *is a legal triple* **then**
            Print $(U \cup \{v\}, Y, A \cup S)$
        if $(U, Y \cup \{v\}, A \cup S)$ *is a legal triple* **then**
            Print $(U, Y \cup \{v\}, A \cup S)$
    foreach $S \subseteq V(\widehat{G})$ **do**
        if $(U, Y, A \cup S)$ *is a legal triple* **then Print** $(U, Y, A \cup S)$
    return
$T \leftarrow$ a spanning tree of $\widehat{G}$
$P \leftarrow$ a longest path in T
$(v, u, w) \leftarrow$ three consecutive vertices starting from one of the ends of P
$Q \leftarrow N_T(\{u\}) \setminus \{w\}$
foreach $S \subseteq Q$ **do** **Generate-Triples**$(G, \widehat{G} - Q, U, Y, A \cup S)$
foreach $x \in Q$ *and* $S \subseteq Q \cup \{u\} \setminus \{x\}$ **do**
    Generate-Triples$(G, \widehat{G} - (Q \cup \{u\}), U \cup \{x\}, Y, A \cup S)$
    Generate-Triples$(G, \widehat{G} - (Q \cup \{u\}), U, Y \cup \{x\}, A \cup S)$
foreach $x, y \in Q, x \neq y$ *and* $S \subseteq Q \cup \{u\} \setminus \{x, y\}$ **do**
    Generate-Triples$(G, \widehat{G} - (Q \cup \{u\}), U \cup \{x\}, Y \cup \{y\}, A \cup S)$

Lemma 1. *The algorithm call* ***Generate-Triples***$(G, G, \emptyset, \emptyset, \emptyset)$ *generates each legal triple in a connected graph G exactly once.*

Proof. Let G be a connected graph, $\widehat{G}$ a connected subgraph of G and U, Y, A disjoint subsets of $V(G) \setminus V(\widehat{G})$. By $C(G, \widehat{G}, U, Y, A)$ we denote a set of triples $(\widehat{U}, \widehat{Y}, \widehat{A})$ such that $(\widehat{U}, \widehat{Y}, \widehat{A})$ is a legal triple in G, $U \subseteq \widehat{U} \subseteq U \cup V(\widehat{G}), Y \subseteq \widehat{Y} \subseteq Y \cup V(\widehat{G})$ and $A \subseteq \widehat{A} \subseteq A \cup V(\widehat{G})$.

We will prove the following statement by induction on $|V(\widehat{G})|$:

($*$) The algorithm call **Generate-Triples**$(G, \widehat{G}, U, Y, A)$ generates exactly once each triple $(\widehat{U}, \widehat{Y}, \widehat{A}) \in C(G, \widehat{G}, U, Y, A)$.

If $|V(\widehat{G})| \le 2$ the algorithm obviously gives the correct result (lines 1– 9).

Assume that the statement $(*)$ is true for all $G, \widehat{G}, U, Y, A$ defined above, such that $|V(\widehat{G})| < k$.

Notice that if a set U (resp. Y) is not a 2-packing in G, then there is no 2-packing $\widehat{U}$ (resp. $\widehat{Y}$) containing U (resp. Y) and no legal triple can be generated in this recursive call. In such case the algorithm **Generate-Triples** returns no triple, since before returning any triple it checks if it is legal. Hence from now we can assume that the sets U and Y are 2-packings in G.

Consider a connected graph G, a connected subgraph $\widehat{G}$ of G with k vertices and disjoint sets $U, Y, A \subseteq V(G) \setminus V(\widehat{G})$ such that U and Y are 2-packings in G. Let Q, u and w be defined as in the algorithm.

Since at most two vertices from Q can belong to $\widehat{U} \cup \widehat{Y}$, we can partition the set $C(G, \widehat{G}, U, Y, A)$ into three subsets T_i for $i \in \{0, 1, 2\}$ defined as follows:
$T_i = \{(\widehat{U}, \widehat{Y}, \widehat{A}) \in C(G, \widehat{G}, U, Y, A) : \, |Q \cap (\widehat{U} \cup \widehat{Y})| = i\}$.

Notice that any of the vertices in $(Q \cup \{u\}) \setminus (\widehat{U} \cup \widehat{Y})$ belongs or does not belong to A.

Since the graph $\widehat{G} - Q$ is connected and has less than k vertices, by the induction hypothesis all the triples from T_0 are generated in recursive call in line 14, all the triples from T_1 are generated in the recursive call in lines 16-17 and all the triples in T_2 are generated in the recursive call in line 19. □

Theorem 2. *There are $O((2\phi)^n) = O(3.2360..^n)$ legal triples in a connected graph on n vertices. All legal triples can be generated in time $O^*(3.2360..^n)$.*

Proof. Let $f(n)$ be the maximum number of legal triples in a connected graph on n vertices. Consider the call of the algorithm **Generate-Triples** on a connected graph G on n vertices with exactly $f(n)$ legal triples. Notice that

$$f(n) \le 2^q f(n-q) + (2q \cdot 2^q + q(q-1) \cdot 2^{q-1}) f(n-q-1)$$

where q is the cardinality of the set Q defined in the algorithm.

We shall prove by induction on n that $f(n) \le 2 \cdot (2\phi)^n$ for $n \ge 0$. Notice that the inequality holds for all $n \le 2$. Assume that the inequality holds for all values smaller than n. Then:
$f(n) \le 2^q f(n-q) + (2q \cdot 2^q + q(q-1) \cdot 2^{q-1}) f(n-q-1) \le$
$\le 2 \cdot 2^q (2\phi)^{n-q} + 2 \cdot (2q \cdot 2^q + q(q-1) \cdot 2^{q-1})(2\phi)^{n-q-1} =$
$= 2 \cdot 2^n \phi^{n-q-1}(\phi + q + \frac{q(q-1)}{4}) = 2 \cdot (2\phi)^n (\phi + q + \frac{q(q-1)}{4}) \phi^{-(q+1)}$

One can easily verify that the function $h(x) = (\phi + x + \frac{x(x-1)}{4})\phi^{-(x+1)}$ is decreasing for any real $x \geq 1$ and $h(1) = 1$. Hence $f(n) \leq 2 \cdot 2^n \phi^n$.

Since all the local operations are performed in a polynomial time, the time complexity of the algorithm **Generate-Triples** is bounded by $O^*(3.2360..^n)$. $\square$

5 A lower complexity bound

In the analysis of algorithms it is important to know, what the lower bound on their complexity is. In this section we provide a new lower bound on the worst-case time complexity of the algorithm **Improved-Find-L(2,1)-Span** .

Theorem 3. *The lower bound on the complexity of the algorithm **Improved-Find-L(2,1)-Span** is $\Omega^*(3.0731..^n)$, where n denotes the number of vertices in the input graph.*

Proof. Let us consider a path P_n. Let $p(n)$ denote the total number of legal triples (U, Y, A) in P_n and let $r(n)$ denote the number of legal triples (U, Y, A) in P_n such that one arbitrarily chosen end-vertex does not belong to U.

Let v, u and w being three consecutive vertices in one of the ends of P_n (for $n \geq 3$). Define $Z = V(G) \setminus (U \cup Y \cup A)$.

If $v \in U$, then u cannot belong to $U \cup Y$ (since $U \cup Y$ is an independent set), but it can belong to A or to Z (two possibilities). The vertex w cannot belong to U, but it can be in Y, A or Z. Thus the number of legal triples (U, Y, A) in P_n such that $v \in U$ is equal to $2r(n-2)$. Since the case when $v \in Y$ is analogous, the number of legal triples (U, Y, A) such that $v \in U \cup Y$ is equal to $4r(n-2)$.

If $v \notin U \cup Y$, then $v \in A$ or $v \in Z$ (two possibilities) and there are no additional restrictions on the vertices u and w. Thus the number of legal triples (U, Y, A) such that $v \notin U \cup Y$ is equal to $2p(n-1)$.

Hence we get the recursion $p(n) = 2p(n-1) + 4r(n-2)$ for $n > 2$. By a similar reasoning we can receive $r(n) = 2p(n-1) + 2r(n-2)$ for $n > 2$.

By solving this system of recursive equations with initial conditions: $p(1) = 4$; $p(2) = 12$; $r(1) = 3$; $r(2) = 10$ we obtain $p(n) = \Theta(3.0731..^n)$.

Hence the time complexity of the algorithm **Improved-Find-L(2,1)-Span** is bounded from below by $\Omega^*(3.0731..^n)$. $\square$

References

[1] Bulterman, R.W., Sommen F.W. van der, Zwaan G., Verhoeff T., Gasteren A.J.M. van , Feijen W.H.J.: On computing a longest path in a tree. Information Processing Letters, 81, 93–96 (2002)

[2] Eggeman, N., Havet, F., Noble, S.: k-$L(2,1)$-Labelling for Planar Graphs is NP-Complete. Discrete Applied Mathematics 158, 16, 1777-1788 (2010)

[3] Fiala, J., Kratochvíl, J.: On the Computational Complexity of the $L(2,1)$-Labeling Problem for Regular Graphs. ICTCS Proc., LNCS, 228-236 (2005)

[4] Fiala, J., Kratochvíl, J., Kloks, T.: Fixed-parameter complexity of λ-labelings. Discrete Applied Mathematics 113, 1, 59–72 (2001)

[5] Griggs J.R., Yeh R.K.: Labeling graphs with a condition on distance 2. SIAM J. Disc. Math. 5, 586-595 (1992)

[6] Hale, W.K.: Frequency assignment: Theory and applications. Proc. of the IEEE 68, 12, 1497 - 1514 (1980)

[7] Havet, F., Klazar, M., Kratochvíl, J., Kratsch, D., Liedloff, M.: Exact Algorithms for $L(2,1)$-Labeling of Graphs. Algorithmica, published online (2009)

[8] Junosza-Szaniawski K., Rzążewski P.: On improved exact algorithms for $L(2,1)$ labeling of graphs. IWOCA 2010 Proc., LNCS 6460, 34–37 (2011)

[9] Král, D.: An exact algorithm for channel assignment problem. Discrete Applied Mathematics 145, 326-331 (2005)