# Fast Exact Algorithm for $L(2,1)$-Labeling of Graphs[☆]

Konstanty Junosza-Szaniawski[a], Jan Kratochvíl[b], Mathieu Liedloff[c], Peter Rossmanith[d], Paweł Rzążewski[a]

[a] *Warsaw University of Technology, Faculty of Mathematics and Information Science, Pl. Politechniki 1, 00-661 Warszawa, Poland*
[b] *Department of Applied Mathematics, and Institute for Theoretical Computer Science, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic*
[c] *Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, 45067 Orléans Cedex 2, France*
[d] *Department of Computer Science, RWTH Aachen University, Germany*

**Abstract**

An $L(2,1)$-labeling of a graph is a mapping from its vertex set into nonnegative integers such that the labels assigned to adjacent vertices differ by at least 2, and labels assigned to vertices of distance 2 are different. The span of such a labeling is the maximum label used, and the $L(2,1)$-span of a graph is the minimum possible span of its $L(2,1)$-labelings. We show how to compute the $L(2,1)$-span of a connected graph in time $O^*(2.6488^n)$. Previously published exact exponential time algorithms were gradually improving the base of the exponential function from 4 to the so far best known 3, with 3 itself seemingly having been the Holy Grail for quite a while. As concerns special graph classes, we are able to solve the problem in time $O^*(2.5944^n)$ for claw-free graphs, and in time $O^*(2^{n-r}(2+\frac{n}{r})^r)$ for graphs having a dominating set of size $r$.

*Keywords:* graphs, $L(2,1)$-labeling, exponential-time algorithm,
*2010 MSC:* 05C85,
*2010 MSC:* 68R10,
*2010 MSC:* 05C15,
*2010 MSC:* 05C78

[☆]The extended abstract of this paper has been presented at the conference TAMC 2011 [21]. J. Kratochvíl acknowledges partial support of Czech research project 1M0545. Part of the research was done when K. Junosza-Szaniawski and P. Rzążewski visited Charles University supported by Czech project MSM0021620838, and part of the research was done during Dagstuhl seminar 10441 on *Exact Complexity of NP-hard Problems*. M. Liedloff acknowledges the support of the French Agence Nationale de la Recherche (ANR AGAPE ANR-09-BLAN-0159-03)

*Email addresses:* k.szaniawski@mini.pw.edu.pl (Konstanty Junosza-Szaniawski), honza@kam.mff.cuni.cz (Jan Kratochvíl), mathieu.liedloff@univ-orleans.fr (Mathieu Liedloff), rossmani@cs.rwth-aachen.de (Peter Rossmanith), p.rzazewski@mini.pw.edu.pl (Paweł Rzążewski)

## 1. Introduction

An $L(2,1)$-*labeling* of a graph is a mapping from its vertex set into non-negative integers such that the labels assigned to adjacent vertices differ by at least 2, and labels assigned to vertices of distance 2 are different. The *span* of such a labeling is the maximum label used and the minimum possible span of an $L(2,1)$-labeling of a graph $G$ is denoted by $\lambda(G)$. This variant of graph coloring is recently receiving considerable attention (see [4, 10, 14, 29] for some surveys on the problem and its generalizations). It is motivated by the Frequency Assignment Problem whose task is to assign frequencies to transmitters in a broadcasting network while avoiding undesired interference. In the $L(2,1)$-labeling model the vertices of the input graph correspond to the transmitters of a network and the edges indicate which pairs of transmitters are too close to each other so that interference could occur even if the broadcasting channels were just one apart. The second condition follows from a requirement that no transmitter should have two or more close neighbors transmitting on the same frequency.

The concept of distance constrained graph labeling was introduced by Hale [16] and, according to [15], Roberts [26] was the first one who suggested to investigate the $L(2,1)$ case in particular. In their seminal paper [15], Griggs and Yeh present first complexity results and several inspiring conjectures. Their conjecture that $\lambda(G) \leq \Delta(G)^2$ initiated intensive research and is still not fully resolved. It is known to be true for many special graph classes and quite recently has been proved for graphs of large maximum degree [18]. Yet it is interesting to note that the Petersen and Hoffmann-Singleton graphs are the only two known graphs that satisfy equality in this bound (for maximum degree greater than 2).

From the complexity point of view, Griggs and Yeh showed that determining $\lambda(G)$ is NP-hard and raised the question of computational complexity of determining $\lambda(G)$ for trees. The latter was answered by Chang and Kuo by providing a polynomial time algorithm in [5]. This has been later improved to a linear time algorithm by Hasunuma et al. in [17]. For general graphs, Fiala et al. [9] proved that deciding $\lambda(G) \leq k$ remains NP-complete for every fixed $k \geq 4$ (for $k \leq 3$ the problem is polynomial). NP-completeness for planar inputs was proved by Bodlaender et al. [2] for $k = 8$, by Janczewski et al. [20] for $k = 4$ and finally by Eggeman et al. [7] for all $k \geq 4$. The fact that distance constrained labeling is a more difficult task than ordinary coloring is probably most strikingly documented by the NP-completeness of deciding $\lambda(G) \leq k$ for series-parallel graphs [8] (here of course $k$ is part of the input).

Recent trend in algorithmic research is designing exact exponential time algorithms for NP-hard problems while trying to minimize the constant which is the base of the exponential running time function. Kratochvíl et al. [25] gave an $O^*(1.3161^n)$ [1] algorithm for $L(2,1)$-labeling of span 4 (and this algorithm was referenced as one of the examples of the Measure and Conquer branching

[1] In the $O^*$, $\Omega^*$ and $\Theta^*$ notations we suppress the polynomial factor.

technique in [13]). A dynamic programming approach can be used to determining the $L(2,1)$-span (or, in other words, to decide $\lambda(G) \leq k$ even when $k$ is part of the input). The development in this area has been quite interesting. An exact algorithm for the so called Channel Assignment Problem of Kráľ [24] implies an $O^*(4^n)$ algorithm for the $L(2,1)$-labeling problem. This has been improved by Havet et al. [19] to an $O^*(3.8739^n)$ algorithm by proving and using a bound on the number of 2-packings in a connected graph. That paper concludes with a conjecture on partitioning graphs into stars which would imply a better running time for the $L(2,1)$-labeling problem when the minimum degree of the input graph is high. This conjecture was later proved by Alon and Wormald [1], however, even for arbitrarily large minimum degree the running time is not better than $O^*(3^n)$. In the meantime, Junosza-Szaniawski and Rzążewski [22, 23] modified the algorithm and refined the running time analysis and proved that their algorithm runs in time $O^*(3.2361^n)$. A lower-bound of $\Omega(3.0731^n)$ on the worst-case running-time of their algorithm is also provided. The magic running time of $O^*(3^n)$ seemed hardly attainable.

Then at Dagstuhl Seminar 10441 Exact Complexity of NP-hard Problems in November 2010, Peter Rossmanith reported on an $O^*(3^n)$-time algorithm for the $L(2,1)$-labeling problem. Next, using the inclusion-exclusion principle and the fast zeta transform, Cygan and Kowalik [6] proposed an $O^*((\ell+1)^n)$-time algorithm for the channel assignment problem, where $\ell$ is the maximum edge weight. This result also implies an $O^*(3^n)$-time algorithm for the $L(2,1)$-labeling problem. Both those results raised a natural question about a possibility of designing an algorithm with time complexity bounded by $O^*(c^n)$ for some $c < 3$. In this paper we provide a breakthrough in this question by proving the following theorem.

**Theorem 1.** *The $L(2,1)$-span of a connected graph can be determined in time $O^*(2.6488^n)$.*

Our algorithm is based on a reduction of the number of operations performed in the recursive step of the dynamic programming algorithm, which is in essence similar to Strassen's algorithm for matrix multiplication [28]. This trick itself achieves running time $O^*(3^n)$. Further improvement is obtained by proving in Section 2 an upper bound on the number of pairs of disjoint subset of the vertex set, where one of the sets is a 2-packing. Such bounds are also provided for claw-free graphs and graphs with a small dominating set, allowing better running-times of our algorithm on these graph classes. We believe that these bounds and the technique which is used for obtaining them are of interest on their own.

## 2. Auxiliary Combinatorial Results

Throughout the paper we consider finite undirected graphs without multiple edges or loops. The vertex set (edge set) of a graph $G$ is denoted by $V(G)$ ($E(G)$, respectively). The number of vertices of a graph $G$ is called the *order* of

$G$ and denoted by $|G|$. The open neighborhood of a vertex $u$ in $G$ is denoted by $N_G(u)$. The set $N_G[u] = N_G(u) \cup \{u\}$ denotes the closed neighborhood of $u$. The neighborhood of a set $X$ of vertices in $G$ is denoted by $N_G(X) = \bigcup_{v \in X} N_G(v)$ and its closed neighborhood is denoted by $N_G[X] = N_G(X) \cup X$. For a subset $X \subseteq V(G)$ we denote the subgraph of $G$ induced by the vertices in $X$ by $G[X]$. A graph $H$ is a spanning subgraph of $G$ if $V(H) = V(G)$ and $E(H) \subseteq E(G)$. The symbol $n$ is reserved for the number of vertices of the input graph, which will always be denoted by $G$. The distance $\mathrm{dist}_G(u, v)$ between two vertices $u$ and $v$ in a graph $G$ is the length of a shortest path joining $u$ and $v$. The diameter of the graph $G$, denoted by $\mathrm{diam}(G)$, is the maximum distance between vertices in $G$, i.e., $\mathrm{diam}(G) = \max_{u,v \in V(G)} \mathrm{dist}(u, v)$.

A subset $S$ of the vertex set of $G$ is called a *2-packing* if the distance of any two distinct vertices of $S$ is at least 3 (i.e., $S$ is an independent set and no two vertices of $S$ have a common neighbor in $G$). A pair $(S, X)$ of subsets of $V(G)$ is called a *proper pair* if $S \cap X = \emptyset$ and $S$ is a 2-packing in $G$. The number of proper pairs in $G$ will be denoted by $pp(G)$ and by the definition, we have

$$pp(G) = \sum_{\substack{S \subseteq V(G) \\ S \text{ is a 2-packing}}} 2^{n-|S|}.$$

Finally, we define

$$pp(n) = \max\ pp(H)$$

where the maximum is taken over all connected graphs $H$ with $n$ vertices.

*2.1. Bounds on the number of proper pairs in arbitrary connected graphs*

In this section we establish an upper-bound and a lower-bound on the maximum number of proper pairs in arbitrary connected graphs. Then in Section 2.2, we will focus on such bounds for connected claw-free graphs. These upper-bounds will be helpful to establish the worst-case running-time of our algorithm given in Section 3.

**Theorem 2.** *The value of $pp(n)$ is bounded above by $O(2.6488^n)$.*

Proof. Let $G = (V, E)$ be a connected graph on $n$ vertices such that $pp(G) = pp(n)$. We observe that if $S$ is a 2-packing of $G$, then for any edge $e$ of $G$, the set $S$ is also a 2-packing of $G = (V, E \setminus \{e\})$. Thus removing an edge does not decrease the number of proper pairs and we can remove edges from the graph as long as it stays connected. Hence without loss of generality, we assume that $G$ is a tree.

(∗) Suppose in $G$ there are two leaves $v_1$ and $v_2$, which have a common neighbor $v_3$. Notice that every proper pair in $G$ is proper in the graph $H$ obtained from $G$ by removing the edge $v_1v_3$ and adding the edge $v_1v_2$ (see Figure 1). Since this operation does not reduce the number of proper pairs, we can assume that there are no two or more leaves with a common neighbor in $G$.

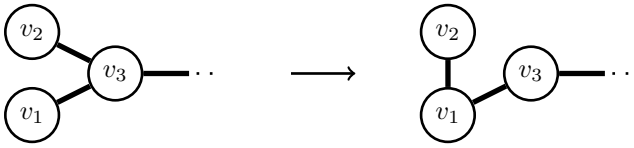



Figure 1: Transformation of two leaves with a common neighbor.

It is easy to observe that $pp(0) = 1$ (as we let that $(\emptyset, \emptyset)$ is the only proper pair of the empty graph), $pp(1) = 3$ and $pp(2) = 8$. Assume that $|V(G)| \geq 3$ and let $P$ be a longest path in $G$. Let $v$ be an end-vertex of the path $P$, $u$ its neighbor on $P$, and $c$ a neighbor of $u$ on $P$ other that $v$ (the third vertex on $P$). By the observation $(*)$ we can assume that $\deg(u) = 2$.

(A) If $\deg(c) \leq 2$, we can partition all proper pairs $(S, X)$ into two subsets: those in which $v \notin S$ and those in which $v \in S$ (see Figure 2).

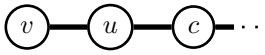



Figure 2: Case (A) with $\deg(c) \leq 2$.

Notice that if $v \notin S$, then $v$ can be in $X$ or outside $S \cup X$. If $v \in S$, then none of the vertices $\{u, c\}$ can belong to $S$. Each of them can be in $X$ or outside $S \cup X$. Since the graphs $G - v$ and $G - \{v, u, c\}$ are connected, we obtain the following recursion:

$$pp(G) \leq 2\, pp(G - v) + 4\, pp(G - \{v, u, c\}) \tag{1}$$

and hence

$$pp(n) \leq 2\, pp(n - 1) + 4\, pp(n - 3). \tag{2}$$

(B) If $\deg(c) > 2$, let $d$ be the neighbor of $c$ on $P$ other than $u$. Let $U = N_G(c) \setminus \{d\}$ and let $W = N_G(U) \setminus \{c\}$. Then all vertices in $W$ are leaves of $G$ (since otherwise $P$ is not the longest path) and all vertices in $U$ except at most one are of degree 2 (from $(*)$). Hence one of the following two cases occurs:

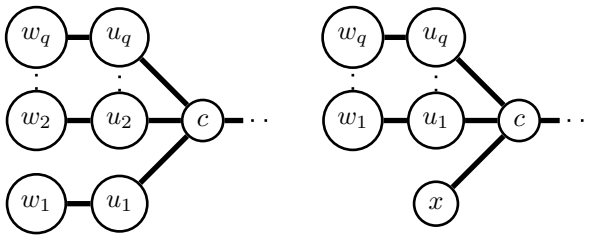
(B0) No vertex from $U$ is a leaf in $G$ (see Figure 3(a)).

(B1) There exists a vertex $x \in U$ which is a leaf in $G$ (there can be at most one such vertex by the observation $(*)$) – (see Figure 3(b)).

We can partition the set of proper pairs $(S, X)$ into those in which $S \cap (W \cup U) = \emptyset$ and the others.

If $S \cap (W \cup U) = \emptyset$, each of the vertices in $W \cup U$ can be in $X$ or outside $S \cup X$.

If $S \cap (W \cup U) = \widehat{S} \neq \emptyset$, $\widehat{S}$ must be a 2-packing in $G$. Notice that the number of proper pairs $(\widehat{S}, \widehat{X})$ in $G[W \cup U \cup \{c\}]$, such that $\widehat{S} \neq \emptyset$ and $c \notin \widehat{S}$ is equal to:




for $q \geq 2$ for $q \geq 1$

(a) Case (B0) with $\deg(c) > 2$ and no neighbor of $c$ is a leaf. (b) Case (B1) with $\deg(c) > 2$ and one neighbor of $c$ is a leaf.

Figure 3: Cases (B0) and (B1)

1. $\underbrace{(3^q - 2^q)2^{q+1}}_{S\cap(W\cup U)=\emptyset} + \underbrace{q \cdot 3^{q-1}2^{q+1}}_{S\cap(W\cup U)\neq\emptyset} = 3^{q-1}2^{q+1}(3+q) - 2^{2q+1}$ for $q \geq 2$ in the case (B0).
2. $\underbrace{(3^q - 2^q)2^{q+2}}_{S\cap(W\cup U)=\emptyset} + \underbrace{q \cdot 3^{q-1}2^{q+2}}_{S\cap(W\cup U\setminus\{x\})\neq\emptyset} + \underbrace{3^q 2^{q+1}}_{x\in S} = 3^{q-1}2^{q+1}(9+2q) - 2^{2q+2}$ for $q \geq 1$ in the case (B1).

Each of the vertices in $(W \cup U \cup \{c\}) \setminus \widehat{S}$ can be in $X$ or outside $S \cup X$.

Since the graphs $G - (W \cup U)$ and $G - (W \cup U \cup \{c\})$ are connected, we obtain the following recursions:

$$pp(n) \leq 2^{2q}\, pp(n-2q) + (3^{q-1}2^{q+1}(3+q) - 2^{2q+1})\, pp(n-2q-1) \quad (3)$$

$$pp(n) \leq 2^{2q+1}\, pp(n-2q-1) + (3^{q-1}2^{q+1}(9+2q) - 2^{2q+2})\, pp(n-2q-2). \quad (4)$$

We shall prove by induction on $n$ that for $n \geq 0$ the following holds:

$$pp(n) \leq 2 \cdot \tau^n \quad (5)$$

where $\tau = 2.6487..$ is the positive root of the equation $\tau^5 = 16\tau + 88$.

It is easy to observe that the inequality (5) holds for $n \leq 2$. Now assume that the inequality holds for all values smaller than $n$.

**Case (A)**

$pp(n) \leq 2\, pp(n-1) + 4\, pp(n-3) \leq 4\tau^{n-1} + 8\tau^{n-3} = 4(\tau^2+2)\tau^{n-3} < 2 \cdot \tau^3 \cdot \tau^{n-3} = 2 \cdot \tau^n$

**Case (B0)**

$pp(n) \leq 2^{2q}\, pp(n-2q) + (3^{q-1}2^{q+1}(3+q) - 2^{2q+1})\, pp(n-2q-1) \leq 2(2^{2q} \cdot \tau^{n-2q} + (3^{q-1}2^{q+1}(3+q) - 2^{2q+1}) \cdot \tau^{n-2q-1}) = 2 \cdot \tau^n (2^{2q} \cdot \tau^{-2q} + (3^{q-1}2^{q+1}(3+q) - 2^{2q+1}) \cdot \tau^{-2q-1}) = 2 \cdot \tau^n ((\frac{2}{\tau})^{2q} - (\frac{2}{\tau})^{2q+1} + \frac{4(3+q)}{\tau^3}(\frac{6}{\tau^2})^{q-1})$

One can easily verify that the function $h_0(x) = (\frac{2}{\tau})^{2x} - (\frac{2}{\tau})^{2x+1} + \frac{4(3+x)}{\tau^3}(\frac{6}{\tau^2})^{x-1}$ is decreasing for all real $x > 2$ and $h_0(2) = 1$.

Hence $pp(n) \leq 2 \cdot \tau^n((\frac{2}{\tau})^{2q} - (\frac{2}{\tau})^{2q+1} + \frac{4(3+q)}{\tau^3}(\frac{6}{\tau^2})^{q-1}) \leq 2 \cdot \tau^n$.

**Case (B1)**

$pp(n) \leq 2^{2q+1}\, pp(n-2q-1) + (3^{q-1}2^{q+1}(9+2q) - 2^{2q+2})\, pp(n-2q-2) \leq 2(2^{2q+1}\tau^{n-2q-1} + (3^{q-1}2^{q+1}(9+2q) - 2^{2q+2})\tau^{n-2q-2}) = 2 \cdot \tau^n((\frac{2}{\tau})^{2q+1} - (\frac{2}{\tau})^{2q+2}) + \frac{4(9+2q)}{\tau^4}(\frac{6}{\tau^2})^{q-1})$

Since the function $h_1(x) = (\frac{2}{\tau})^{2x+1} - (\frac{2}{\tau})^{2x+2} + \frac{4(9+2x)}{\tau^4}(\frac{6}{\tau^2})^{x-1}$ is decreasing for all real $x > 1$ and $h_1(1) < 1$, we obtain:

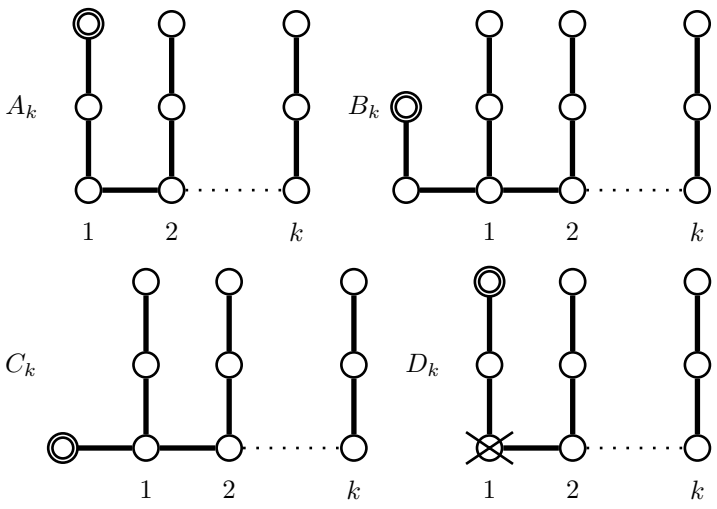
$pp(n) \leq 2 \cdot \tau^n((\frac{2}{\tau})^{2q+1} - (\frac{2}{\tau})^{2q+2} + \frac{4(9+2q)}{\tau^4}(\frac{6}{\tau^2})^{q-1}) < 2 \cdot \tau^n$.

We have shown that regardless of the structure of $G$, the function $2 \cdot \tau^n$ is an upper bound on the number of proper pairs in $G$. Hence $pp(n) = O(\tau^n) = O(2.6488^n)$. □

One may be inclined to conjecture that the worst case is attained in the case of a path $P_n$ on $n$ vertices. A simple calculation shows that $pp(P_n) = \Theta(2.5943..^n)$ is given by the recursion (2): $pp(n) \leq 2\, pp(n-1) + 4\, pp(n-3)$ (see also Section 2.2.2). The following example shows that intuition fails in this case.

**Theorem 3.** *The value of $pp(n)$ is bounded from below by $\Omega(2.6117^n)$.*

Proof. We shall prove the theorem by showing a graph with $\Theta(2.6117..^n)$ proper pairs. Let us consider the following graphs:




Let $a_k, b_k$ and $c_k$ denote the number of proper pairs in the graphs $A_k, B_k$ and $C_k$, respectively. Let $d_k$ denote the number of such proper pairs $(S, X)$ in the graph $D_k$, in which the 2-packing $S$ does not contain the crossed out vertex.

Considering separately the number of proper pairs $(S, X)$, in which $S$ contains or does not contain the marked vertices, we obtain the following system of recursions:

$$\begin{cases} a_k = 2b_{k-1} + 4a_{k-1} \\ b_k = 2c_k + 2d_k \\ c_k = 2a_k + 12d_{k-1} \\ d_k = 4d_{k-1} + 12a_{k-1} \end{cases}$$

Solving this system we obtain the result $a_k = \Theta(x^k)$, where $x = 17.8149..$ is the positive solution of the equation $x^3 = 16x^2 + 576$.

Since $k = n/3$, the graph $A_k$ contains $a_k = \Theta(17.8149..^{n/3}) = \Theta(2.6117..^n)$ proper pairs. □

*2.2. Bounds on the number of proper pairs in connected claw-free graphs*

In graph theory, a complete bipartite graph $K_{1,3}$ is also called a *claw*. A graph is called *claw-free* if it contains no copy of $K_{1,3}$ as an induced subgraph. Claw-free graphs are a well-studied class of graphs. We refer the reader to [3] for properties of this graph class not given in this paper. Let $pp_{cf}(n)$ denote the maximum number of proper pairs in a connected claw-free grap on $n$ vertices. Let $G$ be a claw-free graph on $n$ vertices with $pp_{cf}(n)$ proper pairs.

**Observation 1.** If the diameter of $G$ is at most 2, at most one of the vertices in $V(G)$ can belong to a 2-packing. Hence $pp(G) \leq n \cdot 2^{n-1} + 2^n$.

Assume that $\text{diam}(G) \geq 3$. Notice that removing an edge from a graph does not reduce the number of proper pairs, hence if $H$ is a spanning subgraph of $G$, then $pp(G) \leq pp(H)$.

Let $H$ be a spanning tree of $G$. Let $P$ be a longest path in $H$, $v_1$ be an end-vertex of $P$, $u_1$ be its neighbor on $P$, $w$ be the neighbor of $u_1$ on $P$ other that $v_1$ (the third vertex on $P$) and $x$ be a neighbor of $w$ on $P$ other that $u_1$ (the fourth vertex on $P$).

*2.2.1. Transformations*

Depending on the structure of $G$ near the end-vertex $v_1$ of $P$, we shall apply one of the following transformations to $H$, obtaining another spanning subgraph of $G$. The transformations are applied recursively, in the given order of precedence. The main goal of the transformations is to obtain a spanning subgraph of $G$, in which a longest path ends either with an induced copy of $P_4$, or with a clique with pendant vertices (see Figures 13 and 14). After obtaining such a structure, we can apply one of the branching rules described in Section 2.2.2.

**Case 1**

Suppose $\deg_H u_1 > 2$. In this case all $H$-neighbors of $u_1$ except at most one are leaves in $H$ (otherwise $P$ would not be the longest path in $H$). Let $v_1$ and $v_2$ be two neighbors of $u$ which are leaves in $H$ (see Figure 4).

Notice that since $G$ is claw-free, there exists in $G$ at least one of the edges $v_1w$, $v_2w$ or $v_1v_2$.

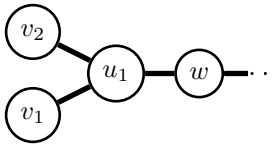



Figure 4: Case 1: $\deg u_1 > 2$

(T1a) If $v_i w \in E(G)$ for some $i \in \{1, 2\}$ (without loss of generality let $i = 1$), add it to $H$ and remove from $H$ the edge $u_1 w$ ($H := (V(G), E(H) \cup \{v_1 w\} \setminus \{u_1 w\})$ (see Figure 5).

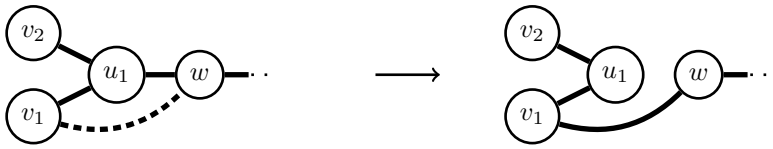



Figure 5: Transformation (T1a)

(T1b) If $v_1 v_2 \in E(G)$, then add it to $H$ and remove from $H$ the edge $v_2 u_1$ (see Figure 6).

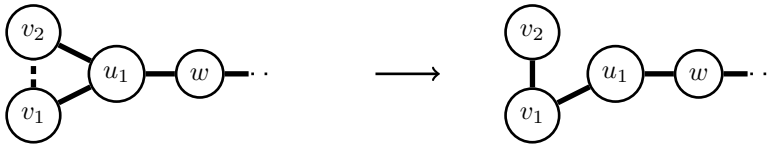



Figure 6: Transformation (T1b)

Notice that if none of the above transformations can be applied to $H$, then $\deg_H u_1 = 2$ (it cannot be smaller than 2 since $\mathrm{diam}(G) \geq 3$).

**Case 2**

If there exists an $H$-neighbor $y \neq x$ of $w$ which is a leaf, then there exists in $G$ at least one of the edges: $u_1 y$, $u_1 x$, $xy$ (since $G$ is claw-free) (see Figure 7).

(T2a) If $u_1 y \in E(G)$ or $xy \in E(G)$, then add it to $H$ and remove the edge $u_1 w$ or $wx$, respectively (see Figure 8).

(T2b) If $u_1 x \in E(G)$, then add it to $H$ and remove the edge $u_1 w$ (see Figure 9).

Notice that if none of above transformations can be applied, then each $H$-neighbor of $w$ except $x$ (let us call them $u_1, \ldots, u_q$) is of degree exactly 2 and their neighbors other than $w$ (let us call them $v_1, \ldots, v_q$ – see Figure 10) are leaves in $H$, since $P$ is the longest path in $H$ and $G$ is claw-free.

**Case 3**

Assume $\deg_H w > 2$.

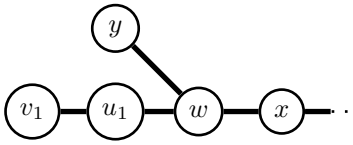



Figure 7: Case 2: one of neighbors of $w$ is a leaf in $H$

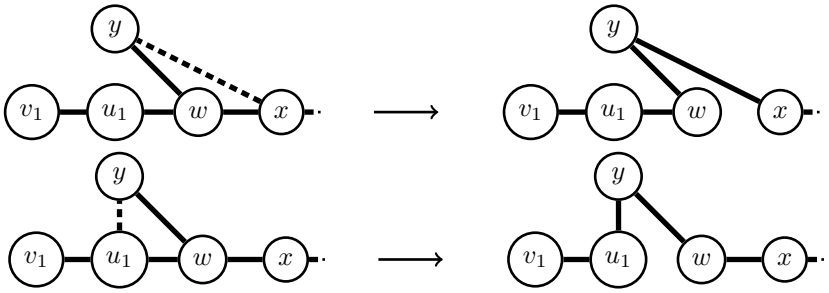



Figure 8: Transformation (T2a)

(T3a) If $u_i x \in E(G)$ for some $i \in \{1, \ldots, q\}$ (without loss of generality let $i = 1$), then add it to $H$ and remove from $H$ the edge $xw$ (see Figure 11).

If no edge $u_i x$ is in $E(G)$ for $i \in \{1, \ldots, q\}$, then the vertices $u_1, \ldots, u_q, w$ induce a $(q+1)$-clique in $G$.

(T3b) Add to $H$ all the edges $u_i u_j$ for $i, j \in \{1, \ldots, q\}$ and $i \neq j$ (see Figure 12). Note that by application of (T3b), $H$ is no longer a tree.

*2.2.2. Branching rules*

Let $H$ be constructed from a spanning tree of $G$ in the given way. If none of the above transformations can be applied to $H$ any more, one of the following cases occurs.

(B1) $\deg_H w = 2$ (see Figure 13)

In this case we can partition all proper pairs $(S, X)$ into two subsets: those in which $v_1 \notin S$ and those in which $v_1 \in S$.

Notice that if $v_1 \notin S$, then $v_1$ can be in $X$ or outside $S \cup X$. If $v_1 \in S$, then none of the vertices $\{u_1, w\}$ can belong to $S$. Each of them can be in $X$ or outside $S \cup X$. Since the graphs $G - v_1$ and $G - \{v_1, u_1, w\}$ are connected and claw-free, we obtain the following recursion:

$$pp(G) \leq 2\, pp(G - v_1) + 4\, pp(G - \{v_1, u_1, w\}) \leq 2\, pp_{cf}(n-1) + 4\, pp_{cf}(n-3). \quad (6)$$

(B2) Transformation (T3b) has been applied to $H$ (see Figure 14).

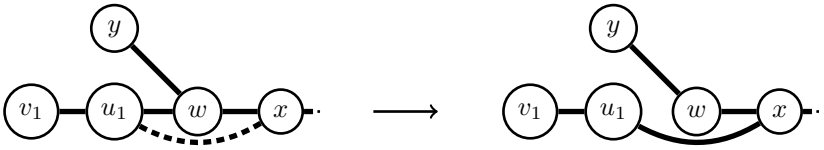



Figure 9: Transformation (T2b)

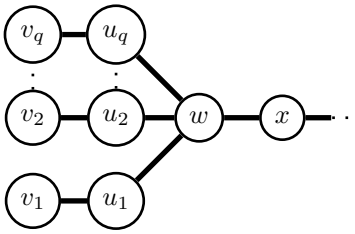



Figure 10: The graph $H$ after applying transformations in Case 1 and Case 2

In this case we can partition all proper pairs $(S, X)$ into two subsets: those in which $\{v_1, \ldots, v_q\} \cap S = \emptyset$ and the remaining ones.

Notice that if $\{v_1, \ldots, v_q\} \cap S = \emptyset$, then each vertex $v_i$ can be in $X$ or outside $S \cup X$. If $\{v_1, \ldots, v_q\} \cap S \neq \emptyset$, then none of the vertices in $\{u_1, \ldots, u_q, w\}$ can belong to $S$. Each of them can be in $X$ or outside $S \cup X$. Since the graphs $G - \{v_1, \ldots, v_q\}$ and $G - \{v_1, \ldots, v_q, u_1, \ldots, u_q, w\}$ are connected and claw-free, we obtain the following recursion:

$$pp(G) \leq 2^q \; pp_{cf}(n-q) + (3^q - 2^q)2^{q+1} \; pp_{cf}(n-2q-1) \tag{7}$$

for $q \geq 2$.

We shall prove by induction on $n$ that for $n \geq 0$ the following holds:

$$pp_{cf}(n) \leq 2 \cdot \tau^n \tag{8}$$

where $\tau \approx 2.5943$ is the positive root of the equation $\tau^3 = 2\tau^2 + 4$.

It is easy to verify that inequality 8 holds for all $n \leq 3$. Moreover, by the Observation 1, if $\mathrm{diam}(H) \leq 2$, then $pp(H) \leq (n+2)2^{n-1} \leq 2 \cdot \tau^n$ for $n \geq 3$. Let us assume that the inequality holds for all values smaller than $n$ and that $\mathrm{diam}(H) \geq 3$. Then one of the cases (B1) or (B2) occurs.

**Case (B1)**

$pp_{cf}(n) \leq 2pp_{cf}(n-1) + 4pp_{cf}(n-3) \leq 2 \cdot \tau^{n-3}(2\tau^2 + 4) = 2 \cdot \tau^n$

**Case (B2)**

$pp_{cf}(n) \leq 2^q \; pp_{cf}(n-q) + (3^q - 2^q)2^{q+1} \; pp_{cf}(n-2q-1) \leq 2 \cdot \tau^n \left( (\frac{2}{\tau})^q + 2\frac{6^q}{\tau^{2q-1}} - (\frac{2}{\tau})^{2q+1} \right)$

One can easily verify that the function $\overline{h}(x) = (\frac{2}{\tau})^x + 2\frac{6^x}{\tau^{2x-1}} - (\frac{2}{\tau})^{2x+1}$ is decreasing for all real $x > 2$ and $\overline{h}(2) < 1$.

Hence $pp_{cf}(n) \leq 2 \cdot \tau^n \left( (\frac{2}{\tau})^q + 2\frac{6^q}{\tau^{2q-1}} - (\frac{2}{\tau})^{2q+1} \right) \leq 2 \cdot \tau^n$.

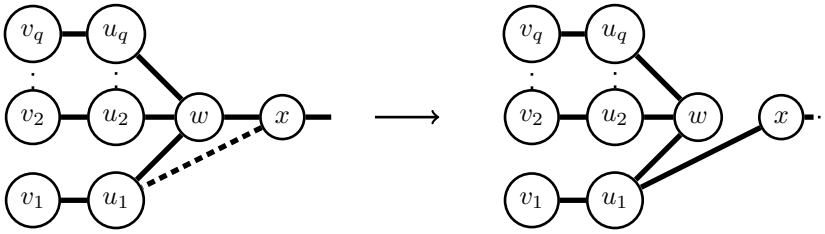



Figure 11: Transformation (T3a)

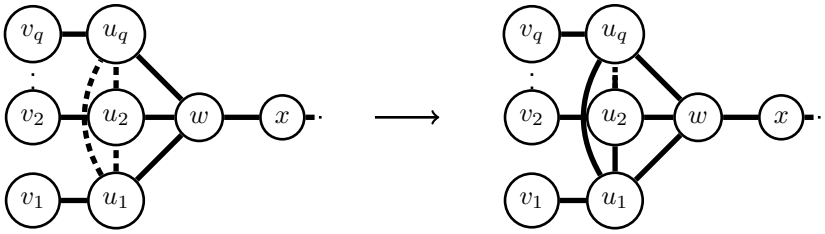



Figure 12: Transformation (T3b)

We have shown that regardless of the structure of $H$, the function $2 \cdot \tau^n$ is an upper bound on the number of proper pairs in $H$. Hence $pp_{cf}(n) = pp(G) \leq pp(H) = O(\tau^n)$.

On the other hand, a path is claw-free and there are exactly $\Theta(\tau^n)$ proper pairs in $P_n$. Hence we have the following Theorem:

**Theorem 4.** *The value $pp_{cf}(n)$ is $\Theta(\tau^n)$, where $\tau \approx 2.5943$.*

*2.3. Covering graphs with connected subgraphs*

The main tool in [19] was partitioning a connected input graph into stars of orders at least 2. Our approach is to divide the computation into connected subgraphs of large constant order. The star graph is an example showing that one cannot always find such a partition. However, we can find a *covering* with a small overlap of the connected subgraphs, as shown by the following result.

**Theorem 5.** *Let $G$ be a connected graph of order $n$ and let $k < n$ be a positive integer. Then there exist connected subgraphs $G_1, G_2, \ldots, G_q$ of $G$ such that*

*(i) every vertex of $G$ belongs to at least one of them,*

*(ii) the order of each of $G_1, G_2, \ldots, G_{q-1}$ is at least $k$ and at most $2k$ (while for $G_q$ we only require $|V(G_q)| \leq 2k$), and*

*(iii) the sum of the numbers of vertices of $G_i's$ is at most $n(1 + \frac{1}{k})$.*

Proof. Assume $G$ is rooted in an arbitrary vertex $r$ and consider a DFS-tree $T$ of $G$. For every vertex $v$ let $T(v)$ be the subtree rooted in $v$. If $|T(r)| \leq 2k$ then add $G$ to the set of desired subgraphs and finish. If there is a vertex $v$ such that $k \leq |T(v)| \leq 2k$ then add $G[V(T(v))]$ to the set of desired subgraphs and

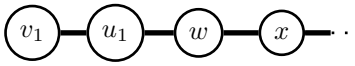



Figure 13: Case (B1)

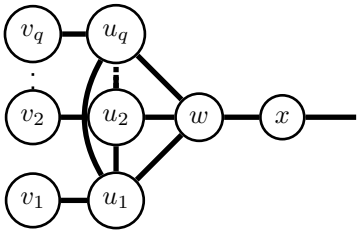



Figure 14: Case (B2)

proceed recursively with $G \setminus V(T(v))$. Otherwise there must be a vertex $v$ such that $|T(v)| > 2k$ and for its every child $u$, $|T(u)| < k$. In such a case find a subset $\{u_1, \ldots, u_j\}$ of children of $v$ such that $k \leq |T(u_1)| + \cdots + |T(u_j)| \leq 2k-1$. Add $G[\{v\} \cup V(T(u_1)) \cup \cdots \cup V(T(u_j))]$ to the set of desired subgraphs and proceed with the graph $G \setminus (V(T(u_1)) \cup \cdots \cup V(T(u_j)))$. This procedure terminates after at most $\frac{n}{k}$ steps and in each of them we have left at most one vertex of the identified connected subgraph in the further processed graph. Notice that from this construction follows that also the graph $G[V(G_i) \cup V(G_2) \cup \cdots \cup V(G_q)]$ is connected for all $i \in \{1, \ldots, q\}$. □

## 3. Exact Algorithm for $L(2,1)$-labeling

One key ingredient in our algorithm are algebraic manipulations very similar to fast matrix multiplication: If we have $2^k \times 2^k$-matrices $A$ and $B$ we can divide them each into four block matrices of the same size. We can then compute $AB$ very easily by eight matrix multiplications of $2^{k-1} \times 2^{k-1}$-matrices. Doing so recursively leads again to a running time of $O(n^3)$ —just as the naive algorithm itself. It is, however, possible to improve on the running time by using only *seven* matrix multiplications to achieve the same result [28]. It turns out that this technique alone does not work in our case, though. We have to use one other trick: We jump between two representations of partial $L(2,1)$-labelings in the course of our dynamic programming algorithm. The idea to use different representations of the same data in dynamic programming is not new and was used in a similar way before [27].

We define the partial function $\oplus$: $\{0, \bar{0}, 1, \bar{1}\} \times \{0,1\} \to \{0,1,\bar{1}\}$ via this table:

| $\oplus$ | 0 | $\bar{0}$ | 1 | $\bar{1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | $\bar{1}$ | – | – | – |

The entry "–" signifies that $\oplus$ is not defined on that input.

We generalize $\oplus$ to vectors via

$$a_1a_2\ldots a_n \oplus b_1b_2\ldots b_n = \begin{cases} (a_1 \oplus b_1)\ldots(a_n \oplus b_n) & \text{if } a_i \oplus b_i \text{ is defined for all } i \in \{1,\ldots,n\}, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

and to sets of vectors $A \subseteq \{0,\bar{0},1,\bar{1}\}^n$, $B \subseteq \{0,1\}^n$ via

$$A \oplus B = \{\, \mathbf{a} \oplus \mathbf{b} \mid \mathbf{a} \in A,\ \mathbf{b} \in B,\ \mathbf{a} \oplus \mathbf{b} \text{ is defined} \,\}.$$

In a nutshell our algorithm proceeds as follows: Given a graph $G = (V, E)$ of order $n$, with $V = \{v_1, \ldots, v_n\}$, it computes tables $T_0, T_1, \ldots, T_{2n} \subseteq \{0,\bar{0},1,\bar{1}\}^n$. Table $T_l$ contains a vector $\mathbf{a} \in \{0,\bar{0},1,\bar{1}\}^n$ if and only if there is a partial labeling $\varphi\colon V \to \{0,\ldots,l\}$ such that :

1. $a_i = 0$ iff $v_i$ is not labeled by $\varphi$ and there is no neighbor $u$ of $v_i$ with $\varphi(u) = l$,
2. $a_i = \bar{0}$ iff $v_i$ is not labeled by $\varphi$ and there is a neighbor $u$ of $v_i$ with $\varphi(u) = l$,
3. $a_i = 1$ iff $\varphi(v_i) < l$, and
4. $a_i = \bar{1}$ iff $\varphi(v_i) = l$.

Once we have all tables $T_l$ it is easy to find the smallest $l$ such that $T_l$ contains at least one vector from $\{1,\bar{1}\}^n$ – such vectors correspond to solutions where all vertices are labeled. We then know that such an $l$ is the $L(2,1)$-span of $G$.

Let $P \subseteq \{0,1\}^n$ be the encodings of all 2-packings of $G$. Formally, $\mathbf{p} \in P$ if and only if there is a 2-packing $S \subseteq V$ such that for all $i$, $1 \le i \le n$, $p_i = 1$ iff $v_i \in S$.

Our strategy is to compute $T_{l+1}$ from $T_l \oplus P$. If $\mathbf{a} \in T_l$ is a vector corresponding to a partial $L(2,1)$-labeling $\varphi$ and $\mathbf{p} \in P$ corresponds to a 2-packing $C$ in $G$, we aim at extending $\varphi$ by setting $\varphi(u_i) = l+1$ for all $u_i \in C$. Such an extension is valid if and only if $p_i = 1$ implies $a_i = 0$, and then $a_i \oplus p_i = \bar{1}$. Thus $T_l \oplus P$ is already almost the same as $T_{l+1}$: $\mathbf{a} \in T_{l+1}$ iff there is an $\mathbf{a}' \in T_l \oplus P$ such that

1. $a_i = 0$ iff $a'_i = 0$ and there is no $v_j \in N(v_i)$ with $a'_j = \bar{1}$
2. $a_i = \bar{0}$ iff $a'_i = 0$ and there is a $v_j \in N(v_i)$ with $a'_j = \bar{1}$
3. $a_i = 1$ iff $a'_i = 1$, and
4. $a_i = \bar{1}$ iff $a'_i = \bar{1}$.

To compute $T_{l+1}$ from $T_l \oplus P$ is therefore easy: Look at each vector in $T_l \oplus P$ and determine for each 0 whether it remains 0 or has to be changed into $\bar{0}$. What remains is to find a method to compute $T_l \oplus P$ fast.

Towards this end let us fix a constant $k$ (whose size will be specified later). Let $G_1, \ldots, G_q$ be a covering of $G$ by connected subgraphs guaranteed by Theorem 5 and let $k'$ be the order of $G_1$. Hence, the relation $k \le k' \le 2k$ holds as long as $q > 1$.

We need one more formalism: If $\mathbf{w}$ is a vector and $A$ is a set of vectors, then

$$A_{\mathbf{w}} = \{\, \mathbf{v} \mid \mathbf{wv} \in A \,\}$$

is the set of all vectors that fall into $A$ after we prefix them with $\mathbf{w}$. Here $\mathbf{wv}$ denotes the concatenation of vectors $\mathbf{w}$ and $\mathbf{v}$.

If $A \subseteq \{0, \bar{0}, 1, \bar{1}\}^n$ and $B \subseteq \{0,1\}^n$, where $n > k'$, we can compute $A \oplus B$ in the following useful, though perhaps at first sight complicated, manner:

$$\begin{aligned} A \oplus B = & \bigcup_{\substack{\mathbf{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \mathbf{v} \in \{0,1\}^{k'} \\ \text{s.t. } \mathbf{u} \oplus \mathbf{v} \text{ defined}}} (\mathbf{u} \oplus \mathbf{v})(A_{\mathbf{u}} \oplus B_{\mathbf{v}}) \\ = & \bigcup_{\substack{\mathbf{v} \in \{0,1\}^{k'} \\ \mathbf{w} \in \{0,1,\bar{1}\}^{k'}}} \mathbf{w}\Big[\Big( \bigcup_{\substack{\mathbf{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \text{s.t. } \mathbf{u} \oplus \mathbf{v} = \mathbf{w}}} A_{\mathbf{u}}\Big) \oplus B_{\mathbf{v}}\Big] \end{aligned}$$

To illustrate the approach, suppose that $k' = 1$, that $\mathcal{A} \subseteq \{0, \bar{0}, 1, \bar{1}\}^n$ is a table containing vectors representing partial labelings of a graph $G$, and that $\mathcal{B} \subseteq \{0,1\}^n$ is a table of vectors representing 2-packings of $G$. The $\oplus$-operation over vectors of $\mathcal{A}$ and $\mathcal{B}$ can be carried out in the following simple way, where terms for which the $\oplus$ operation is not defined are omitted : $\mathcal{A} \oplus \mathcal{B} = (0 \oplus 0)(\mathcal{A}_0 \oplus \mathcal{B}_0) \cup (0 \oplus 1)(\mathcal{A}_0 \oplus \mathcal{B}_1) \cup (\bar{0} \oplus 0)(\mathcal{A}_{\bar{0}} \oplus \mathcal{B}_0) \cup (1 \oplus 0)(\mathcal{A}_1 \oplus \mathcal{B}_0) \cup (\bar{1} \oplus 0)(\mathcal{A}_{\bar{1}} \oplus \mathcal{B}_0)$. As suggested by our previous formula, this can be rewritten as : $\mathcal{A} \oplus \mathcal{B} = 0((\mathcal{A}_0 \cup \mathcal{A}_{\bar{0}}) \oplus \mathcal{B}_0) \cup 1((\mathcal{A}_1 \cup \mathcal{A}_{\bar{1}}) \oplus \mathcal{B}_0) \cup \bar{1}(\mathcal{A}_0 \oplus \mathcal{B}_1)$. When the size of the prefix $k'$ is larger than 1, the computation of the $\oplus$-operation is even more impressive. In the following, we assume that $k \leq k' \leq 2k$, for a fixed constant $k$, and we consider the formula given above to compute $A \oplus B$.

Let us analyze how long it takes to compute $A \oplus B$ in this manner. We are especially interested in the number of $\oplus$-operations on sets with vectors of length $n - k'$. We can omit such a computation if the first set, i.e.,

$$\bigcup_{\substack{\mathbf{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \text{s.t. } \mathbf{u} \oplus \mathbf{v} = \mathbf{w}}} A_{\mathbf{u}} \tag{9}$$

is empty. So how many pairs $\mathbf{v}, \mathbf{w}$ are there such that there is at least one $\mathbf{u}$ with $\mathbf{u} \oplus \mathbf{v} = \mathbf{w}$? If we fix $\mathbf{v}$, then obviously $v_i = 1$ implies $w_i = \bar{1}$. So for a fixed $\mathbf{v}$ there are at most $2^{k'-||\mathbf{v}||}$ many $\mathbf{w}$'s, where $||\mathbf{v}||$ denotes the number of positions $i$ such that $v_i = 1$.

The total number of pairs $\mathbf{v}, \mathbf{w}$ such that $\mathbf{w} = \mathbf{v} \oplus \mathbf{u}$ for some $\mathbf{u}$ and that therefore produce a nonempty contribution in (9) is therefore at most

$$\sum_{\mathbf{v} \in \{0,1\}^{k'}} 2^{k'-||\mathbf{v}||}.$$

Thus, if we draw the $\mathbf{v}$'s from a set of vectors that represent the 2-packings of a connected graph, then we find at most $pp(k')$ such pairs and, hence, we need to make only such many recursive computations of $\oplus$ on sets of vectors of length $n - k'$.

Since we can cover the input graph by induced subgraphs of orders between $k$ and $2k$, this is indeed possible. Theorem 5 implies that the total length of the vectors is $n' \leq n(1 + 1/k)$.

As the length of the vectors is $n' \geq n$, it follows that several coordinates may correspond to the same vertex. However this is not a problem for the algorithm: To be consistent we require that the value of each coordinate corresponding to the same vertex have to be equal, otherwise the algorithm simply removes such a vector from the current table.

In each recursive computation we have to prepare up to $pp(k')$ many pairs of sets of vectors of length $n' - k'$, where $k \leq k' \leq 2k$. Then we recursively compute $\oplus$ on these pairs. From the result we get the next table $T_{l+1}$ in linear time. Preparing the recursive calls and combining their results takes only time linear in the sizes of $A$ and $B$. The size of $B$ is at most $O(n2^{n'})$ bits and the size of $A$ is at most $O(n\, pp(n'))$ bits if we use only our tables $T_l$ for $A$: The $\bar{1}$'s form a 2-packing and for all other nodes there are only two possibilities, 1 or $0/\bar{0}$.

We arrive at the following recurrence for the running time:

$$t_{n'} = O\big(n\, pp(n') + pp(k')t_{n'-k'}\big) \text{ for } k \leq k' \leq 2k$$

We can prove by induction on $n'$ that every solution of this recurrence fulfills $t_{n'} = O\big(nn'pp(n')\big)$: Using the induction hypothesis on $t_{n'-k'}$ yields

$$\begin{aligned} t_{n'} &= O\big(npp(n')\big) + pp(k')\, O\big(n(n'-k')pp(n'-k')\big) \\ &= O\big(npp(n') + n(n'-k')pp(n')\big) = O\big(nn'pp(n')\big). \end{aligned}$$

We arrive at our main result by choosing the constant $k$ so big ($k \geq 61425$ is sufficient) such that $2.64876..^{1+1/k} \leq 2.6488$ since then $t_n \leq t_{n'} = O^*(pp(n(1 + 1/k))) \leq O^*(2.6488^n)$.

Consequently, we have the following Theorem :

**Theorem 6.** *The $L(2,1)$-span of a connected graph can be determined in time $O^*(2.6488^n)$.*

Combining the previously described algorithm (whose pseudocode is also given in Section 4) and the bound provided by Theorem 4, the running-time of our algorithm can be slightly improved for claw-free graphs :

**Corollary 7.** *An $L(2,1)$-span of a claw-free graph can be determined in time $O(2.5944^n)$.*

**Remark 1.** Since line graphs are claw-free and edge version of $L(2,1)$-labeling of a graph $G$ is equivalent to an $L(2,1)$-labeling of a line graph of $G$, we obtain the following corollary.

**Corollary 8.** *An edge $L(2,1)$-labeling problem can be solved in time $O(2.5944^m)$ and exponential space, where $m$ denotes the number of edges of the input graph.*

## 4. Pseudocode of the Algorithm

For the sake of completeness, we provide in this section the pseudocode of the algorithm described in Section 3.

Let $G_1, \ldots, G_q$ be a covering of a given graph $G$ by connected subgraphs as ensured by Theorem 5. Let $d_i$ be the order of $G_i$ for $1 \leq i \leq q$. We denote by $n'$ the sum $d_1 + \cdots + d_q$. Let $A \subseteq \{0, \bar{0}, 1, \bar{1}\}^{n'}$ and $B \subseteq \{0,1\}^{n'}$. We first provide Algorithm MUL which computes $A \oplus B$ using the methods from Section 3.

Algorithm $MUL(A, B, d_1, \ldots, d_q)$:
**if** $q = 1$ **then return** $A \oplus B$ **fi**;
$k' := d_1$;
**for each** $\mathbf{v} \in \{0,1\}^{k'}$ **do**
  $R := \emptyset$;
  **for each** $\mathbf{w} \in \{0, 1, \bar{1}\}^{k'}$ **do**
    $A' := \emptyset$;
    **for each** $\mathbf{u} \in \{0, \bar{0}, 1, \bar{1}\}^{k'}$ **do**
      **if** $\mathbf{w} = \mathbf{u} \oplus \mathbf{v}$ **then** $A' := A' \cup A_{\mathbf{u}}$ **fi**
    **od**;
    **if** $A' \neq \emptyset$ **then** $R := R \cup MUL(A', B_{\mathbf{v}}, d_2, \ldots, d_q)$ **fi**
  **od**
**od**;
**return** $R$

Obviously, the body of the innermost loop is executed exactly $24^{k'}$ times. All operations can be carried out in constant time except set union and the recursive calls to MUL. A set union $X \cup Y$ takes at most $n(|X| + |Y|)$ steps if we implement sets as simple arrays and remember that we can sort them using radix-sort. Not counting recursive calls the running time is therefore $O(n(|A|+|B|))$ if $d_1 = k' = O(1)$. In the boundary case that $q = 1$ then a brute force attack to compute $A \oplus B$ can be carried out in $O(n \cdot |A| \cdot |B|) = O(n8^{k'})$, which is constant if $d_1 = k' = O(1)$.

Let $(v_1, \ldots, v_{n'})$ be the vertices of $G$ (with duplicates allowed) such that

$$(v_1, \ldots, v_{n'}) = (u_{11}, \ldots, u_{1d_1},\ u_{21}, \ldots, u_{2d_2},\ \ldots\ldots,\ u_{q1}, \ldots, u_{qd_q})$$

and $G_i = (u_{i1}, \ldots, u_{id_i})$. Let $k$ be a constant large enough so that $pp(1 + 1/k) = \tau^{1+1/k} < 2.6488$, where $\tau = 2.6487..$ is the positive root of the equation $\tau^5 = 16\tau + 88$ as provided in the proof of Theorem 2. As the proof of Theorem 5 is constructive and provides a polynomial-time algorithm to compute a cover $G_1, \ldots, G_q$, we can arrange the decomposition of $G$ into $G_1, \ldots, G_q$ in such a way that $k \leq d_i \leq 2k$ for any $1 \leq i < q$. In addition, Theorem 5 ensures

that $n' \leq n(1+1/k)$. While the correctness of the following algorithm does not depend on such an arrangement, it is crucial to the running time, which is closely related to all $pp(G[G_i])$'s. We can only guarantee those to be small if the $G[G_i]$'s are connected. We refer the reader to Section 3 for the running-time analysis.

Finally the following Algorithm T computes tables $T_1, \ldots, T_{2n}$.

Algorithm $T(G, n, v_1, \ldots, v_{n'})$
$T_0 := \{0^{n'}\}$;
$P := \emptyset$;
**for each** $\mathbf{x} \in \{0,1\}^{n'}$ **do**
  **if** $\{ v_i \mid x_i = 1 \}$ is a 2-packing **then** $P := P \cup \{\mathbf{x}\}$ **fi**
**od**;
**for** $l = 1, \ldots, 2n$ **do**
  $R := MUL(T_{l-1}, P, d_1, \ldots, d_q)$;
  $T_l := \emptyset$;
  **for each** $\mathbf{x} \in R$ **do**
    **for** $i, j = 1, \ldots, n'$ **do**
      **if** $x_i = 0$, $x_j = \bar{1}$ and $v_i \in N(v_j)$ **then** $x_i := \bar{0}$ **fi**
    **od**;
    $T_l := T_l \cup \{\mathbf{x}\}$
  **od**;
**od**;
**return** $T_1, \ldots, T_{2n}$

## 5. Computing $L(2,1)$-labeling of graphs with a small dominating set

Havet *et al.* [19] presented a refined analysis of their algorithm for graphs that can be partitioned into disjoint stars, each containing at least $d$ vertices. They called such graphs $d$-well partitioned. We can conduct similar analysis for our algorithm. This leads to a better bound for the complexity of the algorithm for more general class of graphs – the graphs with minimum dominating set of size at most $r$ (let us call them $r$-dominated).

Let us start by recalling the definition of a dominating set. Given a graph $G = (V, E)$, a subset $D \subseteq V$ is called *dominating set* if each vertex of $V \setminus D$ has at least one neighbor in $D$. In this section, graphs do not have to be connected. Assume that $D = \{w_1, w_2, \ldots, w_r\}$ is a dominating set in a given graph $G$. Then the set of vertices of $G$ can be easily partitioned into $r$ disjoint stars with centers in vertices of $D$. If some vertex is dominated by more than one vertex from $D$, we include it to only one, arbitrarily chosen, star. Let $S(v)$ for $v \in D$ denote the set of vertices of a star with the center in $v$.

**Lemma 9.** *There are at most* $\binom{r}{k}\left(\frac{n}{r}\right)^k$ *2-packings of size* $k$ *in any* $r$*-dominated graph with* $n$ *vertices.*

Proof. Notice that at most one vertex from every star belongs to a 2-packing. We can choose $k$ stars in $\binom{r}{k}$ ways. Then we can choose the vertices for 2-packing

in $\prod_{i=1}^{k} |S(w_{j_i})|$ ways, where $w_{j_i}$ is the center of $j_i$-th star. It is easy to observe that this product has largest value if the stars have the same order.

Hence the number of sets can be bounded by $\binom{r}{k} \prod_{1 \leq i_1 < i_2 < \cdots < i_k \leq r} |S(w_i)| \leq \binom{r}{k} \prod_{1 \leq i_i < i_2 < \cdots < i_k \leq r} \frac{n}{r} = \binom{r}{k} (\frac{n}{r})^k$. □

If we consider a graph consisting of $r$ disjoint stars, each having the same number of vertices, we notice that the bound presented above is tight.

**Lemma 10.** *There are at most $2^{n-r}(2 + \frac{n}{r})^r$ proper pairs in any $r$-dominated graph on $n$ vertices.*

Proof. Let us calculate the number of proper pairs $(S, X)$. For every fixed 2-packing $X$, every vertex from $V(G) \setminus Y$ can belong to $S$ or not. Hence the bound on number of proper pairs is $\sum_{k=0}^{r} \binom{r}{k} (\frac{n}{r})^k 2^{n-k} = 2^{n-r}(2 + \frac{n}{r})^r$. □

An shown by Fomin *et al.* [12], finding a minimum dominating set in a graph on $n$ vertices can be done in time $O(1.5137^n)$ and exponential space.

**Corollary 11.** *The $L(2,1)$-span of any $r$-dominated graph on $n$ vertices can be found in time $O^*(2^{n-r}(2 + \frac{n}{r})^r)$.*

Simple calculation shows that for $\frac{n}{r} > 3.8912$ the bound from Corollary 11 is better than the bound from Theorem 6. The figure below shows how the complexity of the algorithm depends on $\frac{n}{r}$.

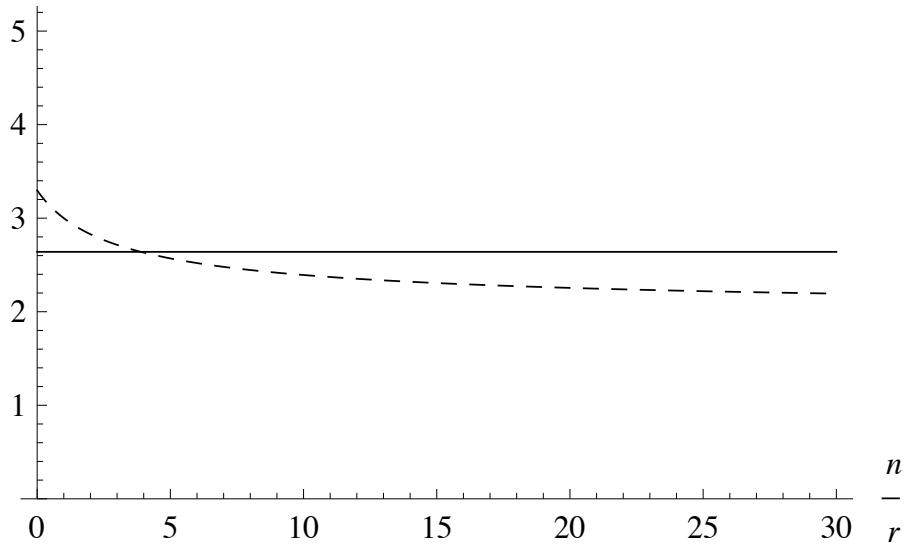



Figure 15: The power bases of complexity bounds from Theorem 6 (plain line) and Corollary 11 (dashed line) in terms of $\frac{n}{r}$

## References


[1] Alon, N., Wormald, N.: High degree graphs contain large-star factors, *"Fete of Combinatorics and Computer Science", Bolyai Soc. Math. Studies (20)* (2010), pp. 9–21.

[2] Bodlaender, H.L., Kloks, T., Tan, R.B., van Leeuwen, J.: Approximations for lambda-Colorings of Graphs. *Computer Journal 47* (2004), pp. 193–204.

[3] Brandstädt, A., Le V.B., Spinrad J.P.: Graph classes: A survey, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1999.

[4] Calamoneri, T.: The $L(h,k)$-Labelling Problem: A Survey and Annotated Bibliography. *Computer Journal 49* (2006), pp. 585–608

[5] Chang, G. J., Kuo, D.: The $L(2,1)$-labeling problem on graphs. *SIAM Journal of Discrete Mathematics 9* (1996), pp. 309–316.

[6] Cygan, M., Kowalik, L.: Channel assignment via fast zeta transform. *Information Processing Letters 111* (2011), pp. 727–730.

[7] Eggeman, N., Havet, F., Noble, S.: $k$-$L(2,1)$-Labelling for Planar Graphs is NP-Complete for $k \geq 4$. *Discrete Applied Mathematics 158* (2010), pp. 1777–1788.

[8] Fiala, J., Golovach, P., Kratochvíl, J.: Distance Constrained Labelings of Graphs of Bounded Treewidth. *Proceedings of ICALP 2005*, LNCS 3580 (2005), pp. 360–372.

[9] Fiala, J., Kloks, T., Kratochvíl, J.: Fixed-parameter complexity of $\lambda$-labelings. *Discrete Applied Mathematics 113* (2001), pp. 59–72.

[10] Fiala, J., Kratochvíl, J.: Locally constrained graph homomorphisms - structure, complexity, and applications. *Computer Science Review 2* (2008), pp. 97–111.

[11] Fomin, F. V., Grandoni, F., Kratsch, D.: Measure and conquer: Domination – A case study. *Proceedings of ICALP 2005*, LNCS 3380 (2005), pp. 192–203.

[12] Fomin F. V., Grandoni, F., Kratsch, D.: A measure and conquer approach for the analysis of exact algorithms. Journal of the ACM 56 (5): 25:1ÂŰ32 (2009)

[13] Fomin, F. V., Kratsch, D.: Exact Exponential Algorithms, Springer, 2010.

[14] Griggs, J. R., Král, D.: Graph labellings with variable weights, a survey. *Discrete Applied Mathematics 157* (2009), pp. 2646–2658.

[15] Griggs, J. R., Yeh, R. K.: Labelling graphs with a condition at distance 2. *SIAM Journal of Discrete Mathematics 5* (1992), pp. 586–595.

[16] Hale, W.K.: Frequency assignemnt: Theory and applications. *Proc. IEEE 68* (1980), pp. 1497–1514

[17] Hasunuma, T., Ishii, T., Ono, H., Uno, Y.: A Linear Time Algorithm for L(2,1)-Labeling of Trees. *Proceedings of ESA 2009*, LNCS 5757 (2009), pp. 35–46

[18] Havet, F., Reed, B., Sereni, J.-S.: $L(2,1)$-labellings of graphs. *Proceedings of SODA 2008* (2008), pp. 621–630.

[19] Havet, F., Klazar, M., Kratochvíl, J., Kratsch, D., Liedloff, M.: Exact algorithms for $L(2,1)$-labeling of graphs. *Algorithmica 59* (2011), pp. 169–194.

[20] Janczewski, R., Kosowski, A., Małafiejski. M.: The complexity of the $L(p,q)$-labeling problem for bipartite planar graphs of small degree. *Discrete Mathematics 309* (2009), pp. 3270–3279

[21] Junosza-Szaniawski K., Kratochvíl J., Liedloff M., Rossmanith P., Rzążewski P.: Fast Exact Algorithm for L(2,1)-Labeling of Graphs. *Proceedings of TAMC 2011*, LNCS 6648 (2011), pp. 82-93

[22] Junosza-Szaniawski K., Rzążewski P.: On Improved Exact Algorithms for $L(2,1)$-Labeling of Graphs. *Proceedings of IWOCA 2010*, LNCS 6460 (2011), pp. 34–37

[23] Junosza-Szaniawski K., Rzążewski P.: On the Complexity of Exact Algorithm for $L(2,1)$-labeling of Graphs. *Information Processing Letters 111* (2011), pp. 697-701

[24] D. Král': Channel assignment problem with variable weights. *SIAM Journal on Discrete Mathematics 20* (2006), pp. 690–704.

[25] Kratochvíl, J., Kratsch, D., Liedloff, M.: Exact algorithms for $L(2,1)$-labeling of graphs. *Proceedings of MFCS 2007*, LNCS 4708 (2007), pp. 513–524.

[26] Roberts, F.S.: private communication to J. Griggs.

[27] van Rooij, J. M. M., Bodlaender, H. L., Rossmanith, P.: Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution, *Proceedings of ESA 2009*, LNCS 5757 (2009), pp. 566–577.

[28] Strassen, V.: Gaussian Elimination is not Optimal. *Numerische Mathematik 13* (1969), pp. 354–356.

[29] Yeh, R.: A survey on labeling graphs with a condition at distance two. *Discrete Mathematics 306* (2006), pp. 1217–1231.