Exact algorithms for $L(2,1)$-labeling of graphs *

Frédéric Havet† Martin Klazar ‡ Jan Kratochvíl ‡
Dieter Kratsch § Mathieu Liedloff §

Abstract

The notion of distance constrained graph labelings, motivated by the Frequency Assignment Problem, reads as follows: A mapping from the vertex set of a graph $G = (V, E)$ into an interval of integers $\{0, \dots, k\}$ is an $L(2,1)$-labeling of G of span k if any two adjacent vertices are mapped onto integers that are at least 2 apart, and every two vertices with a common neighbor are mapped onto distinct integers. It is known that for any fixed $k \geq 4$, deciding the existence of such a labeling is an NP-complete problem. We present exact exponential time algorithms that are faster than the naive $O((k+1)^n)$ algorithm that would try all possible mappings. The improvement is best seen in the first NP-complete case of $k = 4$ – here the running time of our algorithm is $O(1.3006^n)$. Furthermore we show that dynamic programming can be used to establish an $O(3.8730^n)$ algorithm to compute an optimal $L(2,1)$-labeling.

---

*An extended abstract of the paper [15] was presented at the conference "Mathematical Foundations of Computer Science (MFCS 2007)", Český Krumlov, Czech Republic, 26–31 August 2007.

†Projet Mascotte I3S (CNRS & UNSA) and INRIA, INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 Sophia-Antipolis Cedex. E-mail: `Frederic.Havet@sophia.inria.fr`. Partially supported by the european project FET-AEOLUS.

‡Department of Applied Mathematics and Institute for Theoretical Computer Science, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic. E-mail: `(klazar|honza)@kam.ms.mff.cuni.cz`. Supported by Research grant 1M0545 of the Czech Ministry of Education.

§Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France. E-mail: `(kratsch|liedloff)@univ-metz.fr`.

1 Introduction

History. The Frequency Assignment Problem (FAP) asks for assigning frequencies to transmitters in a broadcasting network with the aim of avoiding undesired interference. One of the graph theoretical models of FAP which is well elaborated is the notion of *distance constrained labeling* of graphs. An $L(2,1)$-labeling of a graph G is a mapping from the vertex set of G 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. In this model, the vertices of G represent the transmitters and the edges of G express which pairs of transmitters are too close to each other so that an undesired interference may occur, even if the frequencies assigned to them differ by 1. This model was introduced by Roberts [19] and since then the concept has been intensively studied. Undoubtedly, distance constrained graph labelings provide a graph invariant of significant theoretical interest. Let us mention a few of the known results and open problems: Griggs and Yeh [11] proved that determining the minimum possible span of G – denoted by $L_{2,1}(G)$ – is an NP-hard problem. Fiala et al. [4] later proved that deciding $L_{2,1}(G) \leq k$ remains NP-complete for every fixed $k \geq 4$, while Bodlaender et al. [1] proved NP-hardness for planar inputs for $k = 8$. (For $4 \leq k \leq 7$ and planar inputs, the complexity is still open.) When the span k is part of the input, the problem is nontrivial even for trees – though a polynomial time algorithm based on bipartite matching was presented in [2], existence of a linear time algorithm for trees is still open. Moreover, somewhat surprisingly, the problem becomes NP-complete for series-parallel graphs [3], and thus the $L(2,1)$-labeling problem belongs to a handful of problems known to separate graphs of tree-width 1 and 2 by P/NP-completeness dichotomy. From the structural point of view, Griggs and Yeh [11] conjectured that every graph of maximum degree Δ satisfies $L_{2,1}(G) \leq \Delta^2$. Gonçalves [10] proved the upper bound $L_{2,1}(G) \leq \Delta^2 + \Delta - 2$ by analyzing an algorithm of Chang and Kuo [2]. Very recently, using probabilistic arguments, Havet, Reed and Sereni [12] settled Griggs and Yeh conjecture for large enough Δ. However, for $\Delta > 2$, the Moore graphs are the only graphs known to require span Δ^2, and for large Δ the graphs known to require the largest span are incidence graphs of projective planes for which $L_{2,1} \geq \Delta^2 - \Delta$. It is an open problem if there are infinitely many graphs satisfying $L_{2,1}(G) > \Delta^2 - o(\Delta)$.

Generalizations have been considered, both in the direction of taking into account larger distances and in the direction of allowing a more complicated structure of the frequency space. In the latter direction, circular metric was considered by Leese et al. [16] and Liu et al. [18], showing that in certain sense the circular metric is easier than the linear one (e.g., the circular span of a tree is uniquely determined by its maximum degree and can thus be determined in linear time). Fiala and Kratochvíl consider in [5] the utmost generalization for the case when the metric in the frequency space can be described by a graph, say H. They define the notion of an $H(2,1)$-labeling of G, which is a mapping from the vertex set of G into the vertex set of H such that vertices adjacent in G are mapped onto nonadjacent (distinct) vertices of H, and vertices with a common neighbor (in G) are mapped onto distinct vertices of H. They also show that $H(2,1)$-labelings are exactly locally injective homomorphisms from G to $\overline{H}$, the complement of H. In particular, an $L(2,1)$-labeling of span k is a locally injective homomorphism into the complement of the path of length k. (The complement of the path of length 4 is depicted in Figure 1(a).) The complexity of locally injective homomorphisms was considered in [5, 6, 7] where a number of NP-complete cases were identified, but the complete characterization is still open.

Regarding larger distance constraints, the general channel assignment problem was addressed in [17] and [14]. This problem asks, given a graph G and nonnegative integer weights $w : E(G) \to \{0, 1, 2, \dots\}$ on edges, for a labeling $f : V(G) \to \{0, 1, \dots, k\}$ such that $|f(u) - f(v)| \geq w(uv)$ for every edge $uv \in E(G)$; the aim is to minimize k, the span of the assignment. Král [14] shows that, using Dynamic Programming, this problem can be solved in time $O^*((l+2)^n)$, where l is the maximum edge weight and n the number of vertices of G. Since an $L(2,1)$-labeling on G can be expressed as the channel assignment problem on $G^{(2)}$ (the distance power of G) with weights 1 and 2 only – weight 1 for pairs of vertices at distance 2 in G, weight 2 for the edges of G –, an $O^*(4^n)$ time exact algorithm for the $L(2,1)$-labeling problem follows.

Our results. The goal of this paper is to explore exact exponential time algorithms for the $L(2,1)$-labeling problem. Since one cannot hope for polynomial time algorithms (unless P = NP), our aim is to design algorithms with running time $O^*(c^n)$ and minimizing the constant c.[1] First we consider exact algorithms deciding whether the input graph has an $L(2,1)$-labeling of span at most k. We show that it is not difficult to beat the trivial bound $c \leq k+1$ (which follows from merely checking all possible mappings

[1]Here we use the so called O^* notation: $f(n) = O^*(g(n))$ if $f(n) \leq p(n) \cdot g(n)$ for some polynomial $p(n)$.

from $V(G)$ into $\{0, 1, \ldots, k\}$) by presenting an algorithm of running time $O^*((k-2)^n)$ which can also be generalized to the $H(2,1)$-labeling problem. Then we refine the branching algorithm for the case of span $k = 4$ to achieve an algorithm of running time $O^*(1.3161^n)$ (beating $c = k - 2 = 2$). By a refined analysis we establish an improved running time of $O^*(1.3006^n)$ for the same algorithm. We also obtain a lower bound of $\Omega(1.2290^n)$ for the worst-case running time of our algorithm.

Finally we study the problem of computing an optimal $L(2,1)$-labeling, i.e., one of the smallest span. By designing a general branching algorithm similar to the one for span 4, we give a polynomial space algorithm for computing an $L(2,1)$-labeling of span k, if one exists, in time $O^*((k-2.5)^n)$. Then, we show that by a dynamic programming approach the $L_{2,1}$-span of a graph can be computed and an optimal $L(2,1)$-labeling can be constructed in time $O(3.8730^n)$ and exponential space.

Preliminaries. 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 open (closed) neighborhood of a vertex u in G is denoted by $N_G(u)$ ($N_G[u]$, respectively). The symbol n is reserved for the number of vertices of the input graph, which will always be denoted by G. The subgraph of $G = (V, E)$ induced by $S \subseteq V$ is denoted by $G[S]$. A subset $S \subseteq V$ fulfilling $N[u] \cap N[v] = \emptyset$ for all $u, v \in S$, is called a 2-*packing* of G.

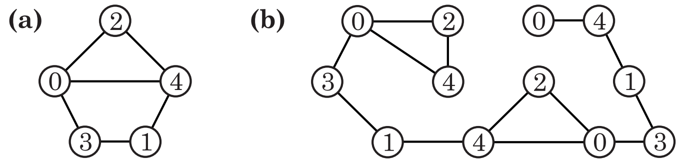

Figure 1: (a) The graph $H = \overline{P_5}$. (b) A graph G with an $L(2,1)$-labeling of span 4 as a locally injective homomorphism into H.

2 Exact algorithm for locally injective homomorphisms

A *graph homomorphism* is an edge preserving vertex mapping between two graphs. More formally, a mapping $f : V(G) \to V(H)$ is a homomorphism from G to H if $f(u)f(v) \in E(H)$ whenever $uv \in V(G)$. Such a mapping is sometimes referred to as an *H-coloring* of G since homomorphisms provide a generalization of the concept of graph colorings – k-colorings of G are exactly homomorphisms from G to the complete graph K_k. Hell and Nešetřil [13] proved that from the computational complexity point of view, homomorphisms admit a complete dichotomy – deciding existence of a homomorphism into a fixed graph H is polynomial when H is bipartite and NP-complete otherwise. The study of exact algorithms for graph homomorphisms was initiated in [9].

A homomorphism $f : G \to H$ is called *locally injective* if for every vertex $u \in V(G)$, its neighborhood is mapped injectively into the neighborhood of $f(u)$ in H, i.e., if every two vertices with a common neighbor in G are mapped onto distinct vertices in H. When deciding the existence of a locally injective homomorphism, one might try to utilize the known algorithms that list all possible homomorphisms and then check if any of them is locally injective. It is not surprising that using the local injectivity, one can often do much better. The trivial brute-force algorithm for H-homomorphism relates the base of the exponential function that expresses the running time to the number of vertices of H, while we show in Theorem 1 that H-locally-injective-homomorphism can be solved in time $O^*((\Delta(H) - 1)^n)$, where $\Delta(H)$ is the maximum degree of a vertex of H. This, in most cases considerable, speed-up is achieved when we label the vertices consecutively using the fact that a neighbor of an already labeled vertex has only a limited number of candidate labels.

Without loss of generality we may assume that G is a connected graph, since otherwise we solve the problem on each connected component of G separately. In the algorithm, f denotes a partial labeling of the vertices of G by vertices of H which is a candidate for a locally injective homomorphism from G to H.

Algorithm-H-LIH(G)

if $\exists v \in V(G)$ *s.t.* v *is unlabeled and* v *has at least one neighbor* u *which was already labeled* **then**

  foreach $c \in N_H(f(u)) \setminus f(N_G(u))$ **do**

    set $f(v) = c$

    Algorithm-H-LIH(G)

else

  if $\exists u \in V(G)$ *s.t.* u *is unlabeled* **then**

    foreach $c \in V(H)$ **do**

      set $f(u) = c$

      Algorithm-H-LIH(G)

  else

    if *the labeling* f *is a locally injective homomorphism from* G *to* H **then**

      return the labeling

Theorem 1. *The H-Locally-Injective-Homomorphism problem is solved in time $O^*((\Delta(H) - 1)^n)$ by Algorithm-H-LIH.*

Proof. In the first step the algorithm picks an unlabeled vertex, say u, and labels it in $|V(H)|$ ways. In the second step, the first rule is used and a neighbor v of u is labeled in $\deg_H(f(u)) \leq \Delta(H)$ ways. From this time on, the algorithm branches each time into at most $\Delta(H)-1$ ways. To see this, let $T = (V(G), E(T))$ be an auxiliary graph which contains the edges uv from the application of the first rule. The loop invariant of the algorithm is that T is an acyclic graph consisting of one connected component – containing the so far labeled vertices – and remaining isolated (and unlabeled) vertices. Also, $f(u)f(v) \in E(H)$ for every edge $uv \in E(T)$. From the third round on, the first rule is always applied, and one edge uv is added to T. And since u had another neighbor w in T, $f(w) \in N_H(f(u))$ and so $N_H(f(u)) \setminus f(N_G(u))$ has at most $\Delta(H) - 1$ available labels for v. □

Corollary 2. *The $L(2,1)$-labeling problem of span k can be decided in time $O^*((k-2)^n)$. In particular, $L(2,1)$-labeling of span 4 can be solved in time $O^*(2^n)$.*

Proof. The maximum degree of a vertex in the complement of the path of length k is $\Delta(\overline{P_{k+1}}) = k - 1$. □

On the other hand, the exact algorithm for channel assignment of Kral' [14] when applied to the special case $L(2,1)$-labeling has running time $O^*(4^n)$. Thus only for small span can we hope to improve on both, the running time $O^*((k-2)^n)$ of Corollary 2 and the running time $O^*(4^n)$ of the dynamic programming algorithm of Kral'.

3 A branching algorithm for computing an $L(2,1)$-labeling of span 4

In this section we present a significantly faster algorithm for the case of $L(2,1)$-labeling of span 4. The main idea is the same as for Algorithm-H-LIH – in the first two steps we label two adjacent vertices in all possible (i.e., at most 12) ways. Then we keep labeling the vertices one by one (and branching into several possibilities when necessary) so that the so far labeled part of the input graph G remains connected. It follows that every newly labeled vertex has (at least) one labeled neighbor, and this labeled neighbor has another (at least one) labeled neighbor. The key idea of the speed-up is two-fold. First, we list several rules and apply them in order of their preferences, thus aiming at reducing the number of branching steps. Secondly, we often label several vertices at a time which leads to a more convenient recursion for the upper bound of the running time.

Throughout this section, we assume that we are in the middle of a run of our algorithm and that $f : X \to \{0,1,2,3,4\}$ is a partial $L(2,1)$-labeling of G such that the labeled vertices $X \subseteq V(G)$ induce a connected subgraph. Note that in order to have a chance to admit a valid labeling, G must have maximum degree at most 3. It is also clear that every vertex of degree 3 must be labeled by 0 or 4, and we will keep checking that this condition is satisfied by each candidate labeling f. To avoid trivial cases we assume that G has at least one vertex of degree 3.

Now we describe the rules and discuss their effect on the running time. When a rule is applied to a partially labeled graph, there are at least two labeled vertices and its labeled vertices induce a connected subgraph. In addition, it may be impossible to extend the partial $L(2,1)$-labeling in which case the algorithm stops. For the sake of clarity and brevity, this is not written but implicitly assumed. Note that only Rules 4 and 5 use branchings.

Rule 1 - Forced extensions

(a) If u is an unlabeled vertex whose labeled neighbor v has two labeled neighbors, then the possible label of u is uniquely determined by the labels of v and its neighbors;

(b) if u is an unlabeled vertex with a neighbor v labeled by 1, 2 or 3, then, since v has another labeled neighbor, the label of u is uniquely determined by the labels of v and this neighbor;

(c) if u is an unlabeled vertex of degree 3 with a labeled neighbor v, then the label of u is either 0 or 4 and is uniquely determined by the label of v and its other labeled neighbor;

(d) if u is an unlabeled vertex of degree 2 such that one of its neighbors is labeled and the other one is a (possibly unlabeled) degree 3 vertex, then the label of u is uniquely determined by the labels of its neighbor(s).

The configurations corresponding to these forced extensions are depicted in Figure 2. In each case the label of vertex u is uniquely determined by the ones of the already labeled vertices (in black).

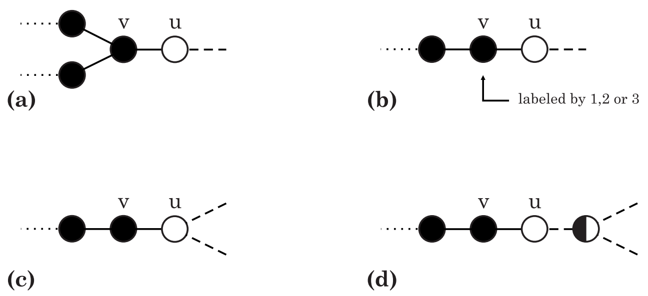

Figure 2: Forced extensions.

Now we show the correctness of the four forced extensions. Let $N_i = \{j : 0 \leq j \leq 4 \text{ and } |j - i| \geq 2\}$, $0 \leq i \leq 4$, represents the set of possible labels of neighbors of a vertex labeled by i.

Rule 1 (a): Suppose that there exists a labeled vertex v having two labeled neighbors, say v_1 and v_2, and one unlabeled neighbor u. Thus, the label of v is either 0 or 4, and the label of u must be the unique element of $N_{f(v)} \setminus \{f(v_1), f(v_2)\}$.

Rule 1 (b): If there is a vertex v with $f(v) \in \{1, 2, 3\}$ having a labeled neighbor v', then its remaining unlabeled neighbor must be labeled by the unique element of $N_{f(v)} \setminus \{f(v')\}$.

Rule 1 (c): Suppose that v is a vertex labeled by 2. The label of its neighbors should be in $N_2 = \{0, 4\}$. Thus, knowing the label of one of its neighbors implies the label of the other. Otherwise, if v has a label different from 2, then $N_{f(v)}$ contains either 0 or 4, forcing the label of a degree-three neighbor.

Rule 1 (d): Suppose that an unlabeled vertex u is adjacent to a labeled vertex v and to u', a vertex of degree 3. If $f(v) = 0$ then $f(u)$ must be 2, otherwise, there is no way to label u'. Symmetrically, if $f(v) = 4$ then $f(u)$ is set to 2. If $f(v)$ is in $\{1, 2, 3\}$ then, as in the second forced extension, the label of u is the unique element of $N_{f(v)} \setminus \{f(v')\}$.

Note that if Rule 1 cannot be applied, every unlabeled vertex that is adjacent to a labeled one has degree at most 2 and each of its adjacent labeled vertices is labeled by 0 or 4.

Definition 3. A path in G is called an *extension path* if all inner vertices are unlabeled and of degree 2, at least one endpoint is labeled and the unlabeled endpoint (if there is one) has either degree 1 or 3. (With a slight abuse of notation we allow that the endpoints are the same vertex, so such an extension path is in fact a cycle and the endpoint is labeled.) The *length* of an extension path is its number of edges.

Lemma 4. *Let $P = v_0 v_1 \dots v_k$ be an extension path such that v_0 is labeled and v_k has degree 1 (and is unlabeled). Let $G' = G[V(G) \setminus \{v_1, \dots, v_k\}]$ be the subgraph obtained by deleting the path P and let $f' : V(G') \to \{0, 1, 2, 3, 4\}$ be a valid extension of f to an $L(2,1)$-labeling of G'. Then f' can be extended to an $L(2,1)$-labeling of the entire graph G.*

Proof. Since $\deg(v_0) \leq 3$ and $f'(v_0) \in \{0, 4\}$ if $deg(v_0) = 3$, there is always a label, say ℓ, available for v_1. The edge $f'(v_0)\ell$ belongs to a cycle in $\overline{P_5}$, and we label the path P wrapping around this cycle. □

Rule 2 - Easy extension

- If P is an extension path with one endpoint of degree 1, Lemma 4 says that the unlabeled vertices of P are irrelevant – we delete them from

G and continue with the reduced graph.

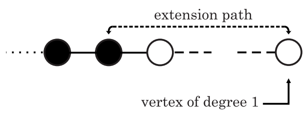

Figure 3: Easy extension. An extension path with one unlabeled endpoint of degree 1.

If neither Rule 1 nor Rule 2 can be applied, every unlabeled vertex that is adjacent to a labeled one has degree 2.

Rule 3 - Cheap extensions

(a) If P is an extension path with both endpoints labeled and of degree 2, we can decide by dynamic programming whether P has an $L(2,1)$-labeling compatible with the labeling of the labeled neighbors of its endpoints. In the affirmative case we just delete the unlabeled vertices and continue with the reduced graph, otherwise we reject the current f as allowing no extension.

(b) If P is an extension path with identical endpoints, we again decide by dynamic programming if the path has an $L(2,1)$-labeling compatible with the label of the endpoint and its labeled neighbor. And we either reduce G or reject f, depending on the outcome.

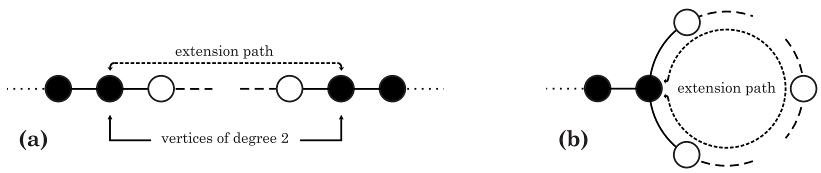

Figure 4: Cheap extensions. (a) An extension path with both endpoints labeled and of degree 2. (b) An extension path with identical endpoints.

The dynamic programming consumes only time polynomial in the length of the path, and so does not affect the base of the exponential function

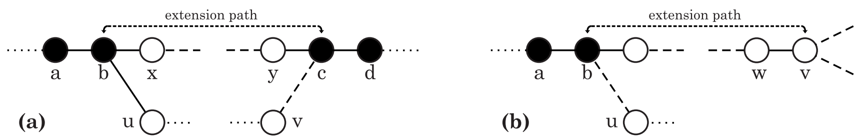

Figure 5: Extensions with strong (a) and weak (b) constraints.

bounding the running time. (To be honest, it only consumes constant time – it can be shown by case analysis that if the path is long enough, then any combination of labelings of its terminal edges is feasible, and so the dynamic programming is only applied to short paths of constant length.)

Rule 4 - Extensions with strong constraints

- Let P be an extension path with both endpoints labeled, each with 0 or 4, such that each endpoint has only one labeled neighbor and at least one of them has another unlabeled neighbor that does not belong to P. In this case we branch along possible labelings of the (at most 4) unlabeled neighbors of the endpoints of P, while extending each of these labelings to entire P (by dynamic programming approach).

Now we discuss the details of this branching rule and the consequences for the running time. The illustrative Figure 5 (a) will be helpful. Let b and c be the labeled endpoints of P, b of degree 3 and c of degree 2 or 3, and let a and d, respectively, be their labeled neighbors. Let further x and y be the unlabeled neighbors of b and c, respectively, on the path P, and let $u \neq x$ be the other unlabeled neighbor of b; and let $v \neq y$ be the other unlabeled neighbor of c, if it exists.

Length 2. If the length of P is 2, then the label of $x = y$ is uniquely determined (in fact, it has to be 2), and we do not really branch.

Length 3. If the length of P is 3, we have the following possible labelings of $ab..cd$ and their extensions to $abxycd$ (up to the symmetric labeling $f' = 4 - f$):

$$\begin{array}{lll} 40xy40 \to 403140 & 40xy42 \to 403142 & 40xy02 \to 402402 \\ 40xy03 \to 402403 & 20xy03 \to 204203 & 20xy04 \to 204204 \\ 20xy40 \to 203140 & 20xy42 \to 203142 & 30xy02 \to 302402 \\ 30xy04 \to 304204 & \multicolumn{2}{c}{30xy03 \to 302403, 304203.} \end{array}$$

We see that most cases allow only one extension of the labeling to P, except for the last case, where branching into two cases occurs. If this happens, we gain at least three newly labeled vertices (x, y, u and possibly also v).

To analyze the running time of our algorithm we determine an upper bound on the maximum number $T(n)$ of leaves in the search tree corresponding to an execution of the algorithm on an input with n unlabeled vertices. The overall running time will then be $O^*(T(n))$ since the application of every rule takes only polynomial time and reduces the number of unlabeled vertices by at least one.

From our above analysis for Rule 4, we obtain two recurrences: $T(n) = 2T(n-3)$ (if c does not have another unlabeled neighbor v or if $v = u$) and $T(n) = 2T(n-4)$ (if the neighbor v exists and is distinct from u). The solution of a recurrence $T(n) = \alpha T(n-\beta)$ is $T(n) = \Theta(c^n)$ for $c = \sqrt[\beta]{\alpha}$. Here we obtain $c = \sqrt[3]{2}$ for the first recurrence and $c = \sqrt[4]{2}$ for the second one.

Length 4. The maximum number of possible extensions of the labelings of an extension path P of length 4 can be established from the exhaustive search trees in Figure 6 depicting all $L(2,1)$-labelings.

Table 1 summarizes the numbers of extensions and corresponding upper bounds for $T(n)$ for extension paths of lengths 2 to 4.

length l of the path P	maximum number of branchings t_1 if $\deg(c) = 2$	solution of the recurrence $T(n) = t_1 T(n-l)$	maximum number of branchings t_2 if $\deg(c) = 3$	solution of the recurrence $T(n) = t_2 T(n-l-1)$
2	1	no branching	1	no branching
3	2	$O(2^{\frac{n}{3}}) = O(1.2600^n)$	2	$O(2^{\frac{n}{4}}) = O(1.1893^n)$
4	2	$O(2^{\frac{n}{4}}) = O(1.1893^n)$	2	$O(2^{\frac{n}{5}}) = O(1.1487^n)$

Table 1: Branching on extension paths of length ≤ 4 with strong constraints.

Length at least 5. If the path is longer, we have two possible extensions of the labeling to the vertices x and u, and two extensions to y and v. For each of these 4 cases we check (in polynomial time, by dynamic programming as in Rule 3) if it extends to a labeling of P. If v exists and $v \neq u$, we gain $\text{length}(P) + 1$ newly labeled vertices (the unlabeled vertices of P plus u and v), which leads to the recurrences $T(n) = 4T(n - \text{length}(P) - 1)$ and $T(n) = O(4^{\frac{n}{6}}) = O(1.2600^n)$.

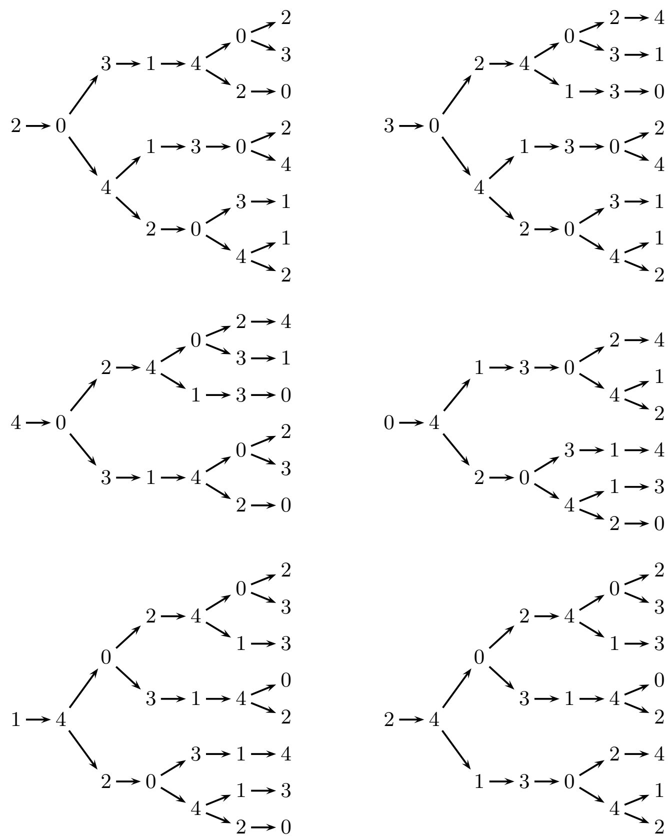

Figure 6: The figures depict all possible $L(2,1)$-labelings of a path starting with labeled vertices a and b and ending with labeled vertices c and d. Recall that the only possible labels of b and c are 0 or 4 when Rule 4 is applied. For example, if $f(a) = 2$, $f(b) = 0$, $f(c) = 0$ and $f(d) = 2$, the first tree shows that there are two possible labelings of a path $abxyzcd:$ 2031402 and 2041302.

If v does not exist, it may seem to mind that we only gain length(P) newly labeled vertices at the same cost of branching. However, in this case we only consider two possible labelings of the pair x and u, and for each of them we only check if it extends to a labeling of P or not. The actual label of y is irrelevant since c has degree 2 in this case. This leads to the recurrences $T(n) = 2T(n - \text{length}(P))$ and $T(n) = O(2^{\frac{n}{5}}) = O(1.1487^n)$.

If $v = u$, then u would be treated by Rule 1 since u is adjacent to c and in that case c has degree 3. Thus $v = u$ is not possible when applying Rule 4.

Comparing all cases we see that the worst case is achieved when $\deg(c) = 2$ and $\text{length}(P) = 3$. Thus using any branching of Rule 4 leads to $T(n) = O(1.2600^n)$.

If none of Rules 1-4 can be applied, then every unlabeled vertex that is adjacent to a labeled one belongs to an extension path with one unlabeled endpoint of degree 3. This is treated by the last branching rule.

Rule 5 - Extensions with weak constraints

- Let P be an extension path with one unlabeled endpoint v of degree 3. Let w be the neighbor of v in P, let the labeled endpoint of P be b, let its labeled neighbor be a and let u be (if it exists) the unlabeled neighbor of b not belonging to P (see Figure 5 (b)). In this case we branch along possible labelings of v, w and (possibly) u, while extending each of these labelings to entire P (by dynamic programming).

Table 2 summarizes the numbers of branchings for paths of length at most 8 (these numbers can be established from the exhaustive search trees of Figure 7 giving all such possible $L(2,1)$-labelings). Again, when $\deg(b) = 2$, we only count the number of labelings of v and w that extend to a labeling of P compatible with the labeling of a and b, since the actual label used on the neighbor of b in P is irrelevant. On the other hand, when $\deg(b) = 3$, we count the number of labelings of v, w and u that allow an extension to a labeling of P compatible with the labels of a and b. Note that in either case both v and b may only receive labels 0 or 4.

In the case of a longer path, we have at most 6 possible labelings of v and w, yielding the recurrence $T(n) = 6T(n - \text{length}(P))$ if $\deg(b) = 2$. If $\deg(b) = 3$, we have at most 12 possible labelings of v, w and u, yielding the recurrence $T(n) = 12T(n - \text{length}(P) - 1)$.

Since $\sqrt[9]{6} < \sqrt[4]{3}$ and $\sqrt[10]{12} < \sqrt[4]{3}$, the overall worst case for Rule 5 is achieved when b is a degree 2 vertex and the extension path has length 5. Hence $T(n) = O(3^{\frac{n}{4}}) = O(1.3161^n)$.

Summarizing the analysis of the algorithm, we obtain the following.

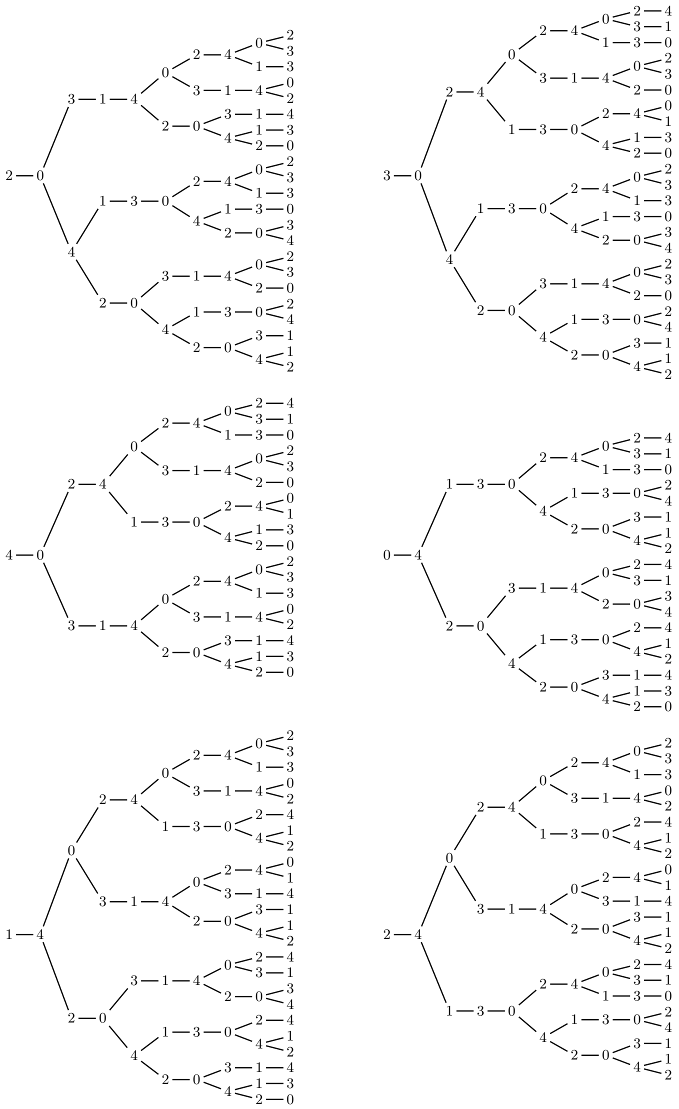

Figure 7: The trees depict all possible $L(2,1)$-labelings of extension paths of length 8 (the root edge corresponds to first two labeled vertices).

length l of the path P	maximum number of branchings t_1 if $\deg(b) = 2$	solution of the recurrence $T(n) = t_1 T(n-l+1)$	maximum number of branchings t_2 if $\deg(b) = 3$	solution of the recurrence $T(n) = t_2 T(n-l)$
1	1	no branching	1	no branching
2	1	no branching	1	no branching
3	2	$O(2^{\frac{n}{3}}) = O(1.2600^n)$	2	$O(2^{\frac{n}{4}}) = O(1.1893^n)$
4	3	$O(3^{\frac{n}{4}}) = O(1.3161^n)$	3	$O(3^{\frac{n}{5}}) = O(1.2458^n)$
5	3	$O(3^{\frac{n}{5}}) = O(1.2458^n)$	3	$O(3^{\frac{n}{6}}) = O(1.2010^n)$
6	5	$O(5^{\frac{n}{6}}) = O(1.3077^n)$	6	$O(6^{\frac{n}{7}}) = O(1.2918^n)$
7	5	$O(5^{\frac{n}{7}}) = O(1.2585^n)$	6	$O(6^{\frac{n}{8}}) = O(1.2511^n)$
8	5	$O(5^{\frac{n}{8}}) = O(1.2229^n)$	7	$O(7^{\frac{n}{9}}) = O(1.2414^n)$

Table 2: Branching on extension paths of length ≤ 8 with weak constraints.

Theorem 5. *The existence of an $L(2,1)$-labeling of span 4 can be decided in time $O^*(1.3161^n)$. If such a labeling exists it can be computed within the same time.*

4 A refined time analysis

In this section, we report on an attempt to improve upon the upper bound of $O^*(1.3161^n)$ for the running time of our algorithm. To do this we use a Measure & Conquer approach (see e.g. [8]). To each graph G with a partial labeling f we assign the following measure

$$\mu = \mu(G, f) = \tilde{n} + \epsilon\hat{n}$$

where $\tilde{n}$ is the number of unlabeled vertices with no labeled neighbor and $\hat{n}$ is the number of unlabeled vertices having a labeled neighbor. Furthermore, ϵ is a constant to be chosen later such that $0 \leq \epsilon \leq 1$.

This means that the weight of a vertex is 0 if it is already labeled, it is ϵ if it is unlabeled with a labeled neighbor, and it is 1 otherwise. Note that $\mu(G, f) \leq n$, where n is the number of vertices of G.

Theorem 6. *The running time of the algorithm of Section 3 is $O^*(1.3006^n)$.*

Proof. Let $G = (V, E)$ be a graph with a partial labeling f. We consider the measure $\mu = \mu(G, f)$ for analyzing the running time of the algorithm

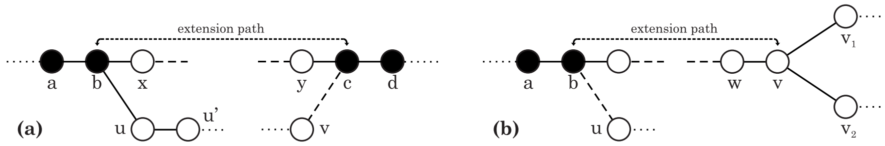

Figure 8: Extensions with strong (a) and weak (b) constraints.

presented in Section 3. The analysis of the running time is quite similar to the one provided in Section 3, but since the measure involves the use of different weights (i.e., ϵ or 1) depending on the status – labeled or unlabeled – of the vertices in their neighborhoods, we obtain new recurrences. To simplify the notation, given a vertex v we denote by $w(v)$ the weight of v. Namely $w(v) = 1$ for each unlabeled vertex v with no labeled neighbor, $w(v) = \epsilon$ for each unlabeled vertex v with a labeled neighbor and $w(v) = 0$ for each labeled vertex v. Thus summing over all vertices of G the equality $\mu(G, f) = \sum_{v \in V} w(v)$ holds.

Rules 1, 2 and 3. The application of Rules 1, 2 and 3 needs only a polynomial time and cannot increase the measure μ.

Rule 4 - Extensions with strong constraints. We consider an extension path P with both endpoints labeled and we branch on the possible labelings of the unlabeled neighbors of the endpoints of P (see Section 3 and Figure 8 (a)).

Recall that by application of Rule 1 and Rule 2, the degrees of u and v (if it exists) are precisely 2. Let u' be the unlabeled neighbor of u. The weight $w(u')$ can be either equal to 1 or equal to ϵ. We distinguish two cases:

- If $w(u') = 1$, labeling u would decrease the weight of u' to ϵ.
- If $w(u') = \epsilon$ then denote by u'' a labeled neighbor of u'. Due to Rule 1, it follows that u' has degree 2 and thus labeling u would create an extension path $P' = uu'u''$ of length two (u' is the unlabeled vertex of P') that can be labeled without any branching by Rule 4 (see analysis of Rule 4 in Section 3). Thus, labeling u would decrease the weight of u' to 0.

If v exists, we can assume that u and v are different since otherwise if $u = v$, Rule 1 (d) would label u. However, in the case that v exists it is possible that $u' = v$. Consequently, labeling the path P and the vertices u and v would decrease the measure by at least $(2\epsilon + (\text{length}(P) - 3) + 2\epsilon)$ if v exists, and by at least $(2\epsilon + (\text{length}(P) - 3) + \epsilon + \min(1 - \epsilon, \epsilon))$ otherwise.

Rule 5 - Extensions with weak constraints. We consider an extension path P with one unlabeled endpoint of degree 3 and we branch on the possible labelings of v, w and (possibly) u (see Section 3 and Figure 8 (b)).

Let v_1 and v_2 be the two neighbors of v which do not belong to P. Note that due to Rule 1 (d), neither v_1 nor v_2 are labeled or adjacent to a labeled vertex.

If u exists, we can assume that u and v_i, $i \in \{1, 2\}$, are different since otherwise if $u = v_i$, Rule 1 would label u. Thus, labeling the path P and the vertex u would decrease the measure by at least $(\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon + \epsilon)$ if u exists, and by at least $(\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon)$ otherwise.

Setting $\epsilon = 0.8190$ and solving the corresponding recurrences we establish a running time bounded by $O(1.3006^n)$. □

It is an interesting question whether a more clever choice of the measure can lead to a more significant improvement of the upper bound on the worst-case running time of the algorithm.

5 A lower bound

It is often the case for branching algorithms that the best known running time is overestimated, even when using the Measure & Conquer technique. Thus it is natural to ask for a lower bound on the worst-case running time. Such bounds give an idea about the (yet unknown) worst-case running time of the algorithm. In this section we prove the following:

Theorem 7. *The worst case running time of our branching algorithm to compute an $L(2,1)$-labeling of span 4 is* $\Omega\left(\left(2 + \sqrt{5}\,\right)^{\frac{n}{7}}\right) = \Omega(1.2290^n)$.

Proof. Consider the graph $G_l = (V_l, E_l)$ defined as follows (see also Figure 5).

Let $V_l = \{a, b\} \cup \bigcup_{\substack{1 \le i \le l \\ 1 \le j \le 6}} \{v_j^i\} \cup \bigcup_{1 \le i \le l} \{x^i\}$. For every i, $1 \le i \le l$, let $E^i = \{v_1^i, v_2^i\}, \{v_2^i, v_3^i\}, \{v_3^i, v_4^i\}, \{v_4^i, v_5^i\}, \{v_5^i, v_6^i\}, \{v_6^i, x^i\}$. The set of edges E_l is defined as $\{a, b\} \cup \{b, v_1\} \cup \bigcup_{1 \le i \le l} E^i \cup \bigcup_{1 \le i < l} \{v_6^i, v_1^{i+1}\}$. Given an integer i, the subgraph induced by $\bigcup_{1 \le j \le 6} \{v_j^i\} \cup \bigcup_{1 \le i \le l} \{x^i\}$ is called the i-th component of G_l. For $i > 1$, we denote by P^i the path induced by

values of $(f(a), f(b))$	$(2,0)$	$(3,0)$	$(4,0)$	$(0,4)$	$(1,4)$	$(2,4)$
possible values for $(f(v_5^1), f(v_6^1))$	$(2,4)$ $(0,4)$ $(1,4)$ $(2,0)$	$(4,0)$ $(1,4)$ $(0,4)$ $(2,4)$ $(2,0)$	$(4,0)$ $(1,4)$ $(0,4)$ $(2,4)$	$(4,0)$ $(2,0)$ $(3,0)$ $(0,4)$	$(2,4)$ $(3,0)$ $(2,0)$ $(4,0)$ $(0,4)$	$(2,4)$ $(3,0)$ $(2,0)$ $(4,0)$
number of branchings	4	5	4	4	5	4

Table 3: Number of branchings for an extension path of length 6 using Rule 5, depending on the labels of a and b. These numbers are established using the search trees of Figure 7.

the vertices $\{v_6^{i-1}\} \cup \bigcup_{1 \leq j \leq 6} \{v_j^i\}$. Note that the graph G_l contains $7l + 2$ vertices.

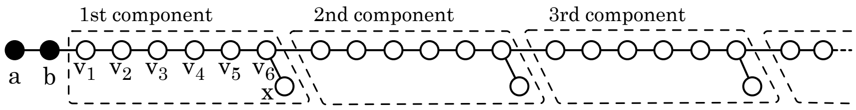

Figure 9: The graph G_l used to prove the lower bound.

Consider an execution of the algorithm on the graph G_l. Recall that, as a preliminary step, the algorithm has to label two adjacent vertices in all possible ways. Suppose that a and b have been labeled such that b receives label 0 or 4. It is not hard to check that none of the rules 1 to 3 can be applied on the resulting (partially labeled) graph. Moreover, there is no extension path with both endpoints labeled. Thus Rule 4 cannot be applied. Hence the algorithm applies Rule 5 to the extension path $P^1 = \{b, v_1^1, v_2^1, \ldots, v_6^1\}$.

The idea is to try out all possible labelings on P^1. More precisely, the algorithm branches along all possible labelings of v_5^1 and v_6^1 and extends these labelings to the other unlabeled vertices of P^1. Then, by application of Rule 2, the vertex x is removed from the graph (according to Lemma 4, its label is easily obtained from a labeling of the remaining graph). The maximum number of branchings along the extension path P^1 having length 6 is given by Table 2, i.e. 5. However, depending on the label of a and b, this number can be even smaller. Table 3 gives the exact number of branchings, depending on the label of a and b.

Note that once P^1 is labeled, then x is removed, v_5^1 and v_6^1 are labeled, and v_6^1 has label 0 or 4. Thus, we can reuse the same arguments as before by renaming v_5^1 by a and v_6^1 by b, and considering the extension path P^2 from the second component. Finally, the whole graph G_l is labeled by labeling component by component using Rule 5.

According to Table 3, the number of branchings is either 4 or 5, depending on the labels of a and b. To analyze the running time of the algorithm on G_l, we denote by $T_4(n)$ the maximum number of leaves in the search tree obtained from an execution of the algorithm on a graph G_l with n unlabeled vertices, providing that the first two vertices (e.g. a and b during the first step, v_5^i and v_6^i, $1 \leq i < l$, at the $(i+1)$-th step) are labeled by $(2,0)$, $(4,0)$, $(0,4)$ or $(2,4)$. Similarly, we define $T_5(n)$ as the maximum number of leaves in the search tree obtained by applying the algorithm on a graph G_l with n unlabeled vertices, providing that the first two vertices are labeled by $(3,0)$ or $(1,4)$. Consequently, for all $n \geq 1$, $T_5(n) \geq T_4(n)$.

Recall that each time an extension path P^i of G_l is labeled, the whole path becomes labeled and the corresponding vertex x^i is removed. Thus, the seven vertices are either labeled or removed from G_l. Moreover, note that among the 4 or 5 branchings done for finding a labeling of the extension path P^i, exactly one ($(3,0)$ or $(1,4)$) produces a branching in 5 labelings in the next step aiming to find a labeling of P^{i+1} (see Table 3). Thus, we obtain the following two recurrences:

$$T_4(n) = 3T_4(n-7) + T_5(n-7) \tag{1}$$

$$T_5(n) = 4T_4(n-7) + T_5(n-7) \tag{2}$$

Subtracting (1) from (2) yields $T_5(n) = T_4(n) + T_4(n-7)$. Together with (1) one obtains $T_4(n) = 4T_4(n-7) + T_4(n-14)$. Solving this latest recurrence, by substituting $T_4(n) = \alpha_4^n$, we obtain $T_4(n) = \left(2+\sqrt{5}\,\right)^{\frac{n}{7}}$ (and thus $T_5(n) = \alpha_4^n(\alpha_4^{-7}+1) \geq T_4(n)$). $\square$

6 Larger span

In this section we show that even in the case of larger span k we can beat $k-2$ as the base of the exponential function bounding the running time of our branching algorithm. Unlike the case of $k=4$, we do not aim at the very best running time achieved by complicated branching rules and a fine tuned analysis. Here we will be satisfied with an improvement of $k-2$ by an additive constant, and we rather aim at simple rules and running time analysis. We are aware that a more careful analysis would lead to a slightly better constant.

Theorem 8. *Deciding if an input graph allows an $L(2,1)$-labeling of span k, for a fixed $k \geq 5$, can be achieved in time $O^*((k-2.5)^n)$ and polynomial space.*

Proof. Note first that in the case of span k, the label space $[0..k]$ contains two labels, namely 0 and k, that allow $k-1$ labels on adjacent vertices, while the other $k-1$ labels from $[1..k-1]$ allow only $k-2$ labels on adjacent vertices.

Our algorithm is similar to the algorithm from Section 3. Again, we start by labeling a chosen vertex in all possible ways, then label a chosen neighbor in all possible ways, and since then on, we keep the labeled part of the input graph connected. In particular, every labeled vertex has at least one labeled neighbor. We use the following rules, and assume they are applied in the preference ordering as they are listed. (See also Figures 10, 11 and 12.)

Rule 1 - Simple branchings

(a) If a vertex v is labeled $f(v) \in [1..k-1]$ and has an unlabeled neighbor u, branch along all possible labelings of u.

(b) If a labeled vertex v has at least two labeled neighbors and an unlabeled neighbor u, branch along all possible labelings of u.

(c) If a labeled vertex v has two unlabeled neighbors u and w, branch along all possible labelings of u and w.

(d) If an unlabeled vertex u has at least two labeled neighbors, branch along all possible labelings of u.

(e) If an unlabeled vertex u has a labeled neighbor and two unlabeled neighbors v, w, branch along all possible labelings of u, v, w.

The configurations corresponding to these forced extensions are depicted in Figure 10.

If Rule 1 cannot be applied, every labeled vertex that has an unlabeled neighbor has degree two, is labeled 0 or k, and has exactly one unlabeled

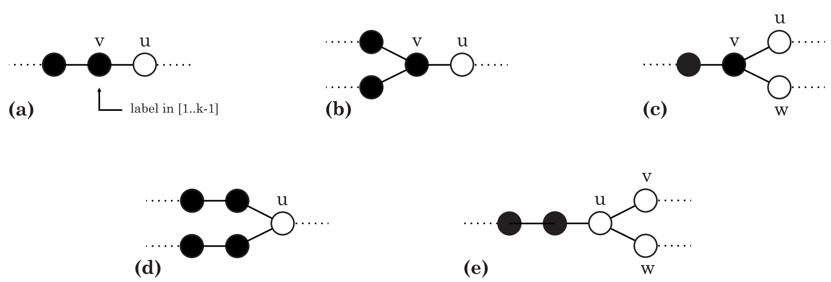

Figure 10: Simple branchings.

neighbor. This unlabeled neighbor has only one labeled neighbor and has degree at most two, and so it belongs to an extension path consisting of unlabeled vertices of degree 2, ending either in an unlabeled vertex of degree 1 or at least 3, or in a labeled vertex of degree 2. These extension paths are treated similarly as in the algorithm of Section 3.

Rule 2 - Cheap extensions

(a) If an extension path ends with an unlabeled vertex of degree 1, disregard the path and continue with the reduced graph.

(b) If an extension path ends with a labeled vertex, check by dynamic programming if there exists a labeling of the path compatible with the labeling of its initial and ending segments.

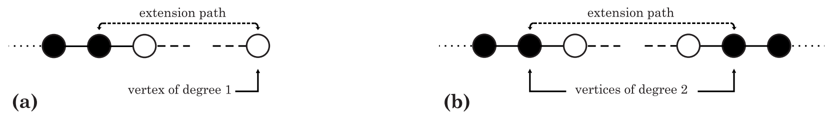

Figure 11: Cheap extensions. (a) An extension path with one unlabeled endpoint of degree 1. (b) An extension path with both endpoints labeled and of degree 2.

Rule 3 - extension path branching

- If an extension path ends with an unlabeled vertex u of degree at least 3, then all neighbors of u are unlabeled (otherwise Rule 1.5 would be applied). Let x be the degree 2 neighbor of u on the extension path and let y, z be two of the other neighbors. Branch along all possible labelings of u, x, y, z and for each such labeling check by dynamic programming if it is compatible with the labeling of the initial segment of the extension path.

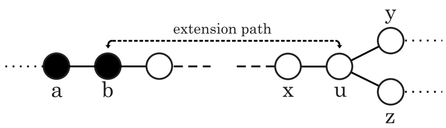

Figure 12: Extension path branching. An extension path with one unlabeled endpoint u of degree at least 3 and no labeled neighbor.

We use the simple weight function - labeled vertices get weight zero, unlabeled vertices get label 1. The total weight in the beginning is n. Application of Rule 2 does not involve branching, does not increase the total weight, and consumes only polynomial amount of time.

Rule 1 (a). As v is labeled in $[1..k-1]$ it forbids 3 labels for u and each of it already labeled neighbours (there is at least one) forbids another label. Hence there are at most $k-3$ possible labels for vertex u. This leads to recurrence $T(n) \leq (k-3)T(n-1)$.

Rule 1 (b). Now v forbids at least 2 labels (with equlity if it is labeled 0 or k) and each of its already labeled neighbours (there are at least two) forbids another label. Hence again there are at most $k-3$ possible labels for u.

Rule 1 (c). There are at most $k-2$ possible labels for u and after u is labeled, at most $k-3$ possible labels are left for w. Hence we gain 2 labeled vertices and have $(k-2)(k-3) < (k-2.5)^2$ possibilities. This leads to $T(n) \leq (k-2.5)^2 T(n-2)$.

Rule 1 (d). As the two labeled neighbors of u have different labels, together they forbid at least 4 labels for u. Hence $T(n) \leq (k-3)T(n-1)$.

Rule 1 (e). This case needs a slightly more careful analysis. Suppose the labeled neighbor of u is labeled by 0. Then u cannot be labeled 0 nor 1. If it is labeled by any $i \in [2..k-1]$, we have $k-3$ possible labels for v (label 0 is blocked) and consequently $k-4$ possible labels for w. This gives $(k-2)(k-3)(k-4)$ possible labelings. If u is labeled by k, we get $k-2$ possible labels for v and $k-3$ for w. So the overall number of possibilities is $(k-2)(k-3)(k-4)+(k-2)(k-3)$ which is smaller than $(k-2.5)^3$ for $k \geq 5$, while we are gaining 3 newly labeled vertices.

Rule 3. Suppose the extension path is $abu_1u_2 \ldots u_{t-1}u$, where a, b are labeled (b with label 0), $u_1, \ldots, u_{t-1}$ are unlabeled vertices of degree 2, u is an unlabeled vertex of degree at least 3, and y, z are other two unlabeled neighbors of u. We may assume $t \geq 2$, since the case $t = 1$ is actually Rule 1 (e).
If $t = 2$, we lead the case analysis according to the label which is assigned to u. If u is labeled in $[2..k-1]$, we have at most $k-3$ possible labels for $x = u_1$, then $k-3$ labels for y and $k-4$ labels for z. If u is labeled by 1, we have $k-2$ possible labels for $x = u_1$, then $k-3$ labels for y and $k-4$ labels for z. If u is labeled by k, we get $k-3$ possible labels for $x = u_1$, $k-2$ labels for y and $k-3$ labels for z. Altogether there are at most $(k-2)(k-3)(k-3)(k-4)+(k-2)(k-3)(k-4)+(k-3)(k-2)(k-3)$ possibilities for labeling the four vertices u_1, u_2, y, z. This number is less than $(k-2.5)^4$ for $k \geq 5$.
If $t > 2$, we either label u with 0 or k, and get $(k-1)(k-2)(k-3)$ possibilities for labeling x, y, z for each of the two. Or u get label $i \in [1..k-1]$, and for each of these $k-1$ cases, we have $(k-2)(k-3)(k-4)$ possibilities for x, y, z. Altogether we have $2(k-1)(k-2)(k-3) + (k-1)(k-2)(k-3)(k-4)$ possible labelings for u, x, y, z, and this number is less than $(k-2.5)^5$ for $k \geq 5$. For each of the labelings of u, x, y, z, we check by dynamic programming along the extension path if the labeling is compatible with the labeling of a and b. So we are gaining at least 5 newly labeled vertices.

We have seen that the application of every rule leads to recurrence $T(n) \leq (k-2.5)^{\ell} T(n-\ell)$ for some ℓ. Hence the number of leaves of the recursion tree is at most $O((k-2.5)^n)$. □

7 An exact dynamic programming algorithm for $L(2,1)$-labeling

Král [14] shows that the channel assignment problem can be solved in time $O^*(4^n)$ if the maximum edge weight is 2. Hence the minimum $L_{2,1}$-span of a graph can be computed in this time. The purpose of this section is to show that in the case of $L(2,1)$-labelings, we can beat the constant 4 in the base of the exponential function expressing the running time.

Theorem 9. *The $L(2,1)$-labeling problem can be solved in time $O^*(15^{\frac{n}{2}}) = O(3.8730^n)$.*

Proof. Let $G = (V, E)$ be a graph. For every integer $i \in \{0, 1, \dots, 2n\}$ and for all subsets $X, Y \subseteq V$ such that $X \cap Y = \emptyset$, we introduce a Boolean variable $\texttt{Lab}[X, Y, i]$. By Dynamic Programming we determine the values of these variables so that

> $\texttt{Lab}[X, Y, i]$ is true if and only if there exists a partial $L(2,1)$-labeling L of span i for X, $L : X \to \{0, 1, \dots, i\}$, such that each vertex of $N(Y) \cap X$ has label at most $i - 1$.

Clearly, $L_{2,1}(G) = \min\{i \mid \texttt{Lab}[V(G), \emptyset, i] \text{ is true}\}$. (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.)

Algorithm Find-L(2,1)-Span
- **forall** $X, Y \subseteq V(G)$, $X \cap Y = \emptyset$, $i = 0, \dots, 2n$ **do**
  - $\texttt{Lab}[X, Y, i] \leftarrow$ false
- **forall** $X, Y \subseteq V(G)$, $X \cap Y = \emptyset$ **do**
  - **if** *X is a 2-packing in G and* $X \cap N(Y) = \emptyset$ **then**
    - $\texttt{Lab}[X, Y, 0] \leftarrow$ true
- **for** $i = 1, \dots, 2n$ **do**
  - **forall** $U, A, Y \subseteq V(G)$, $U \cap A = U \cap Y = A \cap Y = \emptyset$ **do**
    - **if** *U is a 2-packing in G,* $U \cap N(Y) = \emptyset$ *and* $\texttt{Lab}[A, U, i-1]$
    - **then** $\texttt{Lab}[U \cup A, Y, i] \leftarrow$ true
- **for** $i = 0, \dots, 2n$ **do**
  - **if** $\texttt{Lab}[V(G), \emptyset, i]$ **then return** "$L_{2,1}(G) = i$" and Halt

The correctness of the algorithm **Find-L(2,1)-Span** follows from the following observations:

- By the definition of $L(2,1)$-labelling, each set of vertices having the same label induces a 2-packing in G (a set of vertices pairwise at distance greater than 2).
- For every pair of disjoint sets X and Y, X allows a partial labeling of span 0 such that all vertices of X in $N(Y)$ have labels at most -1 if and only if $N(Y) \cap X = \emptyset$ and X induces a 2-packing in G. Hence the initialization of $\mathtt{Lab}[X, Y, 0]$.
- Let X and Y be disjoint sets of vertices and let $i > 0$. Suppose f is an $L(2,1)$-labeling on X of span i such that $f(u) \leq i-1$ for every $u \in N(Y) \cap X$. Let $U \subseteq X$ be the set of vertices of label exactly i, and $A = X \setminus U$. Then U must be a 2-packing in G and $N(Y) \cap U = \emptyset$. Moreover, every labeled neighbor of a vertex from U must have a label at most $i-2$. Thus the labeling f restricted to A satisfies the requirements for A, U, and $i-1$ and $\mathtt{Lab}[A, U, i-1]$ should be true. On the other hand, extending a partial labeling of A of span $i-1$ to a labeling of X by setting $f(u) = i$ for all $u \in U$ gives a labeling satisfying our requirements, provided $N(Y) \cap U = \emptyset$. This justifies the computation of $\mathtt{Lab}[X, Y, i]$ by dynamic programming.

To analyze the running time of our algorithm, the crucial point is the estimation of the number of 2-packings in a connected graph. Let u_k be the number of 2-packings of size k in our graph G. For every 2-packing U of size k, we process all pairs of disjoint subsets A, Y of $V(G) \setminus U$. Since every vertex of $V(G) \setminus U$ has 3 possibilities where to end up (either in A, or in Y, or in neither of them), there are 3^{n-k} such pairs. Thus the running time of our algorithm is

$$O^*\left(\sum_{k=0}^{n} u_k 3^{n-k}\right).$$

We emphasize that the proof of our estimate is constructive and itself leads to an algorithm enumerating all 2-packings of size k of a connected graph with running time $O^*\left(\binom{n/2}{k}2^k\right)$.

Lemma 10. *If n is even, we have $u_k \leq \binom{n/2}{k}2^k$ (and, in particular, $u_k = 0$ for $k > \frac{n}{2}$).*

Proof. We partition the vertex set of G into r stars with $a_1, a_2, \dots, a_r$ vertices, so that $a_1 + a_2 + \cdots + a_r = n$ and $a_i \geq 2$ for every i. Hence $r \leq \frac{n}{2}$. To obtain such partition, we consider a spanning tree T of G. Let u and v be, respectively, the endvertex and its neighbor on a longest path in T. All neighbors of v, with possibly one exception, are leaves in T. These leaves include u and with v they form a star with at least 2 vertices. Deleting it, we get a smaller tree T' which we treat in the same way. This procedure terminates with an empty tree and produces the required partition of vertices into stars of size at least 2 each.

Each 2-packing can have at most one vertex in each star, and so

$$u_k \leq \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq r} a_{i_1} a_{i_2} \dots a_{i_k}.$$

We obtain an upper bound on u_k by maximizing this sum under the condition that the numbers $a_1, a_2, \dots, a_r$ are integers summing up to n and each of them is at least 2. I.e., for the purpose of the estimate, we forget about the underlying graph and regard a_i as free integer variables; their number r varies as well.

It is easy to show that for $a_i \geq 4$, the above sum does not decrease when a_i is replaced with two new numbers $a_i' = 2$ and $a_i'' = a_i - 2$. To see this, let us denote $g_k(A) = \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq r} a_{i_1} a_{i_2} \dots a_{i_k}$ for $A = \{a_1, a_2, \dots, a_r\}$. Let $\widetilde{A} = A \setminus \{a_i\}$ and $A' = \{2, a_i - 2\} \cup \widetilde{A}$. Then

$$\begin{aligned} g_k(A') &= 2(a_i - 2) g_{k-2}(\widetilde{A}) + 2 g_{k-1}(\widetilde{A}) + (a_i - 2) g_{k-1}(\widetilde{A}) + g_k(\widetilde{A}) \\ &\geq a_i g_{k-1}(\widetilde{A}) + g_k(\widetilde{A}) = g_k(A). \end{aligned}$$

Similarly, it does not decrease when two numbers a_i, both equal to 3, are replaced with three new numbers, all equal to 2. As n is even, it follows that the maximum of the above sum under the stated condition is attained for $a_1 = a_2 = \cdots = a_r = 2$, $r = n/2$. (If $k \geq 2$, the sum strictly increases after the replacements and therefore this is the only choice with maximum sum.) The claim of the lemma follows. $\square$

A simple calculation now concludes the proof of Theorem 9 for even n:

$$\sum_{k=0}^{n} u_k 3^{n-k} \leq \sum_{k=0}^{n/2} \binom{n/2}{k} 2^k 3^{n-k}$$

$$= 3^{n/2} \sum_{k=0}^{n/2} \binom{n/2}{k} 2^k 3^{n/2-k} = 3^{n/2}(2+3)^{n/2} = 15^{n/2} = (\sqrt{15})^n.$$

For odd n we note that the number of 2-packings in G is obviously smaller than the number of 2-packings in a graph G' obtained by adding a vertex to G. Then the running time is bounded by $O^*((\sqrt{15})^{n+1}) = O^*((\sqrt{15})^n) = O(3.8730^n)$. □

References

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

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

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

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

[5] Fiala, J., and Kratochvíl, J., Complexity of partial covers of graphs. *Proceedings of ISAAC 2001*, LNCS 2223, 2001, pp. 537–549.

[6] Fiala, J., and Kratochvíl, J., Partial covers of graphs. *Discussiones Mathematicae Graph Theory 22* (2002), pp. 89–99.

[7] Fiala, J., Kratochvíl, J., and Pór, A., On the computational complexity of partial covers of theta graphs. *Electronic Notes in Discrete Mathematics 19* (2005), pp. 79–85.

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

[9] Fomin, F., Heggernes, P., and Kratsch, D., Exact algorithms for graph homomorphisms. *Theory of Computing Systems 41* (2007), pp. 381–393.

[10] Gonçalves, D., On the $L(p, 1)$-labelling of graphs, *DMTCS* Proceedings Volume AE, pp. 81–86.

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

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

[13] Hell, P., and Nešetřil, J., On the complexity of H-colouring. *Journal of Combinatorial Theory Series B 48* (1990), pp. 92–110.

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

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

[16] Leese, R. A., and Noble, S. D., Cyclic labellings with constraints at two distances. *Electronic Journal of Combinatorics 11* (2004), Research paper 16.

[17] Liu, D., and Zhu, X., Multilevel distance labelings for paths and cycles. *SIAM Journal on Discrete Mathematics 19* (2005), pp. 610–621.

[18] Liu, D., and Zhu, X., Circular Distance Two Labelings and Circular Chromatic Numbers. *Ars Combinatoria 69* (2003), pp. 177–183.

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