Algorithmes modérément exponentiels pour l’étiquetage $L(2, 1)$

Mathieu Liedloff

Université d’Orléans - LIFO

Journées nationales du GDR IM

mars 2017
Introduction

Many problems need super-polynomial time to be solved, due to:

- NP-hardness (*the question* $P = NP$ *is still open*)
- nature of the problem (*enumerating a large number of objects*)

*Kurt Gödel to John von Neumann* (1956):

« *It would be interesting to know [...] how strongly in general the number of steps in finite combinatorial problems can be reduced with respect to simple exhaustive search. »

For some problems (e.g. SAT), the best known algorithms are just trivial enumeration, but for many others we can do better.
Our goal:

Focus on NP-hard problems and solve it provably faster than by exhaustive search.

Under the scope of moderately exponential-time algorithms, we deal with the following types of problems:

- decision
- optimization
- counting
- enumeration
In this talk we give moderately exponential-time algorithms for a frequency assignment problem:

computing $L(2, 1)$-labelings in graphs.
In this talk we give moderately exponential-time algorithms for a frequency assignment problem:

computing $L(2, 1)$-labelings in graphs.

- broadcast network
- assign frequencies to transmitters
- avoid undesired interference
In this talk we give *moderately exponential-time algorithms* for a frequency assignment problem: computing $L(2, 1)$-labelings in graphs.

- broadcast network
- assign frequencies to transmitters
- avoid undesired interference
In this talk we give moderately exponential-time algorithms for a frequency assignment problem: computing $L(2, 1)$-labelings in graphs.

- broadcast network
- assign frequencies to transmitters
- avoid undesired interference
Outline of the talk

1. Introduction
2. Definition of $L(2,1)$-labelings and known results
3. Branching algorithm for span 4 labelings
4. A fast algorithm to compute the minimum span
5. Conclusion
An algorithm to compute the minimum span

1. Introduction

2. Definition of $L(2, 1)$-labelings and known results

3. Branching algorithm for span 4 labelings

4. A fast algorithm to compute the minimum span

5. Conclusion
Definition of $L(2, 1)$-labeling

$L(2, 1)$-LABELING

**Input**: A graph $G = (V, E)$.

**Question**: Compute a function $\ell$ of minimum span $k$

\[ \ell : V \to \{0, \ldots, k\} \text{ s.t.} \]

- $u$ and $v$ adjacent $\Rightarrow |\ell(u) - \ell(v)| \geq 2$
- $u$ and $v$ at distance two $\Rightarrow |\ell(u) - \ell(v)| \geq 1$

Model introduced by Roberts, 1988
Complexity results

Many complexity results:

**Theorem** \[\text{Griggs and Yeh, 1992} \] \[\text{Fiala, Kloks, Kratochv\l, 2001}\]

Determining the minimum span $\lambda(G)$ of a graph $G$ is NP-hard.
Deciding whether $\lambda(G) \leq k$ remains NP-c for every fixed $k \geq 4$.

*Separates* treewidth 1 and 2 by P / NP-completeness dichotomy:

**Theorem** \[\text{Chang, Kuo 1996} \] \[\text{Fiala, Golovach, Kratochv\l, 2005}\]

$L(2,1)$-labeling problem is polynomial time solvable on trees, but NP-complete for series-parallel graphs ($k$ is part of the input).

Much more difficult than ordinary coloring:

**Theorem** \[\text{Fiala, Golovach, Kratochv\l, 2005} \] \[\text{Janczewski, Kosowski, Ma\lafiejski, 2009}\]

NP-completeness for series-parallel graphs ($k$ is part of the input).
Deciding whether $\lambda \leq 4$ is NP-complete for planar graphs.
Moderately exponential-time algorithms

- decide span 4: $O(1.3006^n)$ (poly-space) [HKKKL, 2011]
- count span 4: $O(1.1269^n)$ (exp-space) [CGKLP, 2013]
- enumerate span 5 in cubic graphs: $O(1.7990^n)$ [CGKLP, 2013]

Computing the minimum span $k$:

- polynomial space:
  - $O^*((k − 2.5)^n)$ [HKKKL, 2011]
  - $O(7.50^n)$ [JSKL R, 2012]
  - $O(3.4642^n)$ [Kowalik, Socała, 2014]

- exponential space:
  - $O^*(4^n)$ [Král', 2006]
  - $O^*(15^{n/2}) = O(3.88^n)$ [HKKKL, 2011]
  - $O^*(3^n)$ [Cygan, Kowalik, 2011]
  - $O^*(2.6488^n)$ [JSKLRR, 2013]
An algorithm to compute the minimum span

1. Introduction

2. Definition of $L(2,1)$-labelings and known results

3. Branching algorithm for span 4 labelings

4. A fast algorithm to compute the minimum span

5. Conclusion
A convenient way to study $L(2, 1)$-labelings is via locally injective homomorphisms:

**Homomorphism**: A mapping $f : V(G) \rightarrow V(H)$ is a homomorphism from $G$ to $H$ if $f(u)f(v) \in E(H)$ for every edge $uv \in E(G)$.

**Locally injective homomorphism (LIH)**: A homomorphism $f : G \rightarrow H$ is 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., every two vertices having a common neighbor in $G$ are mapped onto distinct vertices in $H$.

Fiala and Kratochvíl, 2002:

**Theorem.** $L(2, 1)$-labelings of span $k$ are locally injective homomorphisms into the complement of the path of length $k$. 
$L(2, 1)$-labelings and LIH
\( L(2, 1) \)-labelings and LIH

\[ \overline{P_5} : \]

\[ G : \]

\[ uv \in E(G) \Rightarrow f(u)f(v) \in E(H) \]
\(\text{Definition and results} \quad \underbrace{L(2, 1)\text{-lab of span 4}}_{\text{minimum } L(2, 1)\text{-lab span}}\)

\(L(2, 1)\)-labelings and LIH

\[\overline{P}_5: \]

\[G: \]

\(uv \in E(G) \Rightarrow f(u)f(v) \in E(H)\)
**Definition and results**

$L(2, 1)$-labeling of span 4 minimum $L(2, 1)$-lab span

$L(2, 1)$-labelings and LIH

$\overline{P}_5$:

$G$:

$u \in V(G) \Rightarrow N(u)$ is mapped injectively on $N(f(u))$ in $H$
Definition and results

L(2, 1)-labelings and LIH

$G$:

$\overline{P}_5$:

$u \in V(G) \Rightarrow N(u)$ is mapped injectively on $N(f(u))$ in $H$
\(L(2, 1)\)-labelings and LIH

\(L(2, 1)\)-labelings of span 4 can trivially be decided in \(O(2^n)\) time.
L(2, 1)-labelings of span 4 can trivially be decided in $O(2^n)$ time.
$L(2, 1)$-labelings and LIH

$L(2, 1)$-labelings of span 4 can trivially be decided in $\mathcal{O}(2^n)$ time.
$L(2, 1)$-labelings and LIH

$L(2, 1)$-labelings of span 4 can trivially be decided in $O(2^n)$ time.
\(L(2, 1)\)-labelings and LIH

\(L(2, 1)\)-labelings of span 4 can trivially be decided in \(O(2^n)\) time.
An algorithm for $L(2, 1)$-labelings of span 4

Description of the rules of the algorithm:

**Rule 1 - Forced Extensions**

- If $u$ is unlabeled and its labeled neighbor $v$ has two labeled neighbors
  $\Rightarrow$ label of $u$ is forced

- If $u$ is unlabeled and its labeled neighbor $v$ has label 1, 2 or 3
  $\Rightarrow$ label of $u$ is forced
An algorithm for $L(2,1)$-labelings of span 4

Description of the rules of the algorithm:

**Rule 1 - Forced Extensions**

- if $u$ is unlabeled and its labeled neighbor $v$ has two labeled neighbors
  $\Rightarrow$ label of $u$ is forced

- if $u$ is unlabeled and its labeled neighbor $v$ has label 1, 2 or 3
  $\Rightarrow$ label of $u$ is forced
An algorithm for \( L(2, 1) \)-labelings of span 4

- if \( u \) is unlabeled, \( d(u) = 3 \) and \( u \) has a labeled neighbor \( v \)
  \( \Rightarrow \) label of \( u \) is forced

\[ \begin{array}{ccc}
0 & 1 & 2 \\
\hline
3 & 4 & v \\
\hline
u & & \\
\end{array} \]

- if \( u \) is unlabeled, \( d(u) = 2 \) and \( u \) has a labeled neighbor \( v \) and a (possibly unlabeled) neighbor of degree 3
  \( \Rightarrow \) label of \( u \) is forced
An algorithm for $L(2, 1)$-labelings of span 4

- if $u$ is unlabeled, $d(u) = 3$ and $u$ has a labeled neighbor $v$ 
  ⇒ label of $u$ is forced

- if $u$ is unlabeled, $d(u) = 2$ and $u$ has a labeled neighbor $v$ and a (possibly unlabeled) neighbor of degree 3 
  ⇒ label of $u$ is forced
An algorithm for $L(2,1)$-labelings of span 4

**Rule 2 - Easy Extension**

- If $P$ is an extension path with one endpoint of degree 1
  - by Lemma 1, $P$ is irrelevant, thus we remove $P$ from $G$

→ If neither Rule 1 nor Rule 2 can be applied, every unlabeled vertex adjacent to the connected labeled component has degree 2.
Rule 2 - Easy Extension

- if \( P \) is an extension path with one endpoint of degree 1

\[ \Rightarrow \] by Lemma 1, \( P \) is irrelevant, thus we remove \( P \) from \( G \)

\[ \rightarrow \] If neither Rule 1 nor Rule 2 can be applied, every unlabeled vertex adjacent to the connected labeled component has degree 2.
Rule 3 - Cheap Extensions

- if $P$ is an extension path with both endpoints labeled and of degree 2

$\Rightarrow$ it is easy to decide whether $P$ has a labeling compatible with its labeled endpoints
An algorithm for $L(2, 1)$-labelings of span 4

- if $P$ is an extension path with identical endpoints
  $\Rightarrow$ it is easy to decide whether $P$ has a labeling compatible with its labeled endpoints

Remark: up to now, no branching was needed
An algorithm for $L(2, 1)$-labelings of span 4

- if $P$ is an extension path with identical endpoints
  \[\Rightarrow\] it is easy to decide whether $P$ has a labeling compatible with its labeled endpoints

Remark: up to now, no branching was needed
Rule 4 - Extensions with Strong Constraints

- if $P$ is an extension path such that
  - both endpoints are labeled by 0 or 4
  - each endpoint has only one labeled neighbor
  - at least one endpoint has degree 3

$\Rightarrow$ Branch along possible labelings of the (at most 4) unlabeled neighbor of the endpoints and extend these labelings to entire path $P$.

By Rule 1-2, degrees of $u$ and $v$ (if it exists) are precisely 2.
An algorithm for $L(2, 1)$-labelings of span 4

Let $T(\mu(G))$ be the maximum number of leaves in a search tree corresponding to an execution on a graph with measure $\mu(G)$.

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

where

- $\hat{n}$ is the number of unlabeled vertices with no labeled neighbor
- $\hat{n}$ is the number of unlabeled vertices having a labeled neighbor
- $\epsilon$ is a constant in $[0, 1]$ \(\Rightarrow\) $\mu(G) \leq n$.

If $\text{length}(P) = 1$. Let $P = b, x, c$.

\(\Rightarrow\) Since the labels of $b$ and $c$ are in $\{0, 4\}$, the label of $x$ is 2 (no branching is needed).
An algorithm for $L(2, 1)$-labelings of span 4

Let $T(\mu(G))$ be the maximum number of leaves in a search tree corresponding to an execution on a graph with measure $\mu(G)$.

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

where

- $\tilde{n}$ is the number of unlabeled vertices with no labeled neighbor
- $\hat{n}$ is the number of unlabeled vertices having a labeled neighbor
- $\epsilon$ is a constant in $[0, 1]$ \Rightarrow $\mu(G) \leq n$.

If $\text{length}(P) = 1$. Let $P = b, x, c$.

\Rightarrow Since the labels of $b$ and $c$ are in $\{0, 4\}$, the label of $x$ is 2 (no branching is needed).
An algorithm for $L(2,1)$-labelings of span 4

If $\text{length}(P) = 2$. Let $P = b, x, y, c$. 

⇒ The possible labelings for $abxycd$ are (up to symmetric labeling $f' = 4 - f$):

- $40xy40 \rightarrow 403140$
- $40xy42 \rightarrow 403142$
- $40xy02 \rightarrow 402402$
- $40xy03 \rightarrow 402403$
- $20xy03 \rightarrow 204203$
- $20xy04 \rightarrow 204204$
- $20xy40 \rightarrow 203140$
- $20xy42 \rightarrow 203142$
- $30xy02 \rightarrow 302402$
- $30xy04 \rightarrow 304204$
- $30xy03 \rightarrow 302403, 304203$.

Only the last case needs to branch into 2 subproblems and for each

- 3 vertices are labeled if $d(c) = 2$; or
- 4 vertices are labeled if $d(c) = 3$. 

$P_5$
An algorithm for $L(2, 1)$-labelings of span 4

If $\text{length}(P) = 3$. Let $P = b, x, z, y, c$.

⇒ By doing the same analysis, we can establish that we have to branch in at most 2 subproblems.

If $\text{length}(P) \geq 4$. Let $P = b, x, \ldots, y, c$.

⇒ There are two possible labelings for $x, u$ and two possible labelings for $y$ and eventually $v$.

For each of these 4 cases we check if it extends to a labeling of $P$. 
An algorithm for $L(2, 1)$-labelings of span 4

If $\text{length}(P) = 3$. Let $P = b, x, z, y, c$.

⇒ By doing the same analysis, we can establish that we have to branch in at most 2 subproblems.

If $\text{length}(P) \geq 4$. Let $P = b, x, \ldots, y, c$.

⇒ There are two possible labelings for $x, u$ and two possible labelings for $y$ and eventually $v$.

For each of these 4 cases we check if it extends to a labeling of $P$. 
Consider the unlabeled neighbor $u'$ of $u$.

**if** $w(u') = 1$. Labeling $u$ would decrease $w(u')$ to $\epsilon$.

**if** $w(u') = \epsilon$. Then $u'$ has a labeled neighbor $u''$.

Due to Rule 1, $u'$ has degree 2.

Labeling $u$ would create an extension path $P' = uu'u''$ that can be labeled without branching by Rule 4.

Thus $w(u')$ would decrease to 0.

If $v$ exists then $u \neq v$, otherwise Rule 1 would label $u$.

However, it is possible that $u' = v$. 
An algorithm for $L(2, 1)$-labelings of span 4

Consider the unlabeled neighbor $u'$ of $u$.

**if** $w(u') = 1$. Labeling $u$ would decrease $w(u')$ to $\epsilon$.

**if** $w(u') = \epsilon$. Then $u'$ has a labeled neighbor $u''$.

Due to Rule 1, $u'$ has degree 2.

Labeling $u$ would create an extension path $P' = uu'u''$ that can be labeled without branching by Rule 4.

Thus $w(u')$ would decrease to 0.

If $v$ exists then $u \neq v$, otherwise Rule 1 would label $u$.

However, it is possible that $u' = v$. 
An algorithm for $L(2, 1)$-labelings of span 4

Consider the unlabeled neighbor $u'$ of $u$.

**if** $w(u') = 1$. Labeling $u$ would decrease $w(u')$ to $\epsilon$.

**if** $w(u') = \epsilon$. Then $u'$ has a labeled neighbor $u''$.

Due to Rule 1, $u'$ has degree 2.

Labeling $u$ would create an extension path $P' = uu'u''$ that can be labeled without branching by Rule 4.

Thus $w(u')$ would decrease to 0.

If $v$ exists then $u \neq v$, otherwise Rule 1 would label $u$.

However, it is possible that $u' = v$. 
An algorithm for $L(2, 1)$-labelings of span 4

Consider the unlabeled neighbor $u'$ of $u$.

**if** $w(u') = 1$. Labeling $u$ would decrease $w(u')$ to $\epsilon$.

**if** $w(u') = \epsilon$. Then $u'$ has a labeled neighbor $u''$.

Due to Rule 1, $u'$ has degree 2.

Labeling $u$ would create an extension path $P' = uu'u''$ that can be labeled without branching by Rule 4.

Thus $w(u')$ would decrease to 0.

If $v$ exists then $u \neq v$, otherwise Rule 1 would label $u$.

However, it is possible that $u' = v$. 
An algorithm for $L(2, 1)$-labelings of span 4

Putting it all together, labeling $P$, $u$ and $v$ would decrease the measure $\mu(G)$ by:

**if $v$ exists.**  
$2\epsilon + (\text{length}(P) - 2) + 2\epsilon$
- $2\epsilon$ for vertices $x$ and $y$
- $\text{length}(P) - 2$ for the other vertices of $P$
- $2\epsilon$ for vertices $u$ and $v$

**if $v$ do not exists.**  
$2\epsilon + (\text{length}(P) - 2) + \epsilon + \min(1 - \epsilon, \epsilon)$
- $2\epsilon$ for vertices $x$ and $y$
- $\text{length}(P) - 2$ for the other vertices of $P$
- $\epsilon$ for vertex $u$
- $\epsilon$ for vertex $u'$
- $\min(1 - \epsilon, \epsilon)$ for vertex $u'$

(depend on the existence of an already labeled neighbor).
Putting it all together, labeling $P$, $u$ and $v$ would decrease the measure $\mu(G)$ by:

**if** $v$ **exists.** $2\epsilon + (\text{length}(P) - 2) + 2\epsilon$

- $2\epsilon$ for vertices $x$ and $y$
- $\text{length}(P) - 2$ for the other vertices of $P$
- $2\epsilon$ for vertices $u$ and $v$

**if** $v$ **do not exists.** $2\epsilon + (\text{length}(P) - 2) + \epsilon + \min(1 - \epsilon, \epsilon)$

- $2\epsilon$ for vertices $x$ and $y$
- $\text{length}(P) - 2$ for the other vertices of $P$
- $\epsilon$ for vertex $u$
- $\epsilon$ for vertex $u'$
- $\min(1 - \epsilon, \epsilon)$ for vertex $u'$

(depends on the existence of an already labeled neighbor).
An algorithm for $L(2,1)$-labelings of span 4

Putting it all together, labeling $P$, $u$ and $v$ would decrease the measure $\mu(G)$ by:

**if $v$ exists.** \[2\varepsilon + (\text{length}(P) - 2) + 2\varepsilon\]
- $2\varepsilon$ for vertices $x$ and $y$
- $\text{length}(P) - 2$ for the other vertices of $P$
- $2\varepsilon$ for vertices $u$ and $v$

**if $v$ do not exists.** \[2\varepsilon + (\text{length}(P) - 2) + \varepsilon + \min(1 - \varepsilon, \varepsilon)\]
- $2\varepsilon$ for vertices $x$ and $y$
- $\text{length}(P) - 2$ for the other vertices of $P$
- $\varepsilon$ for vertex $u$
- $\varepsilon$ for vertex $u'$
- $\min(1 - \varepsilon, \varepsilon)$ for vertex $u'$

(depends on the existence of an already labeled neighbor).
An algorithm for $L(2, 1)$-labelings of span 4

If none of Rules 1-4 can be applied, every unlabeled vertex adjacent to a labeled vertex belongs to an extension path with one unlabeled endpoint of degree 3.

**Rule 5 - Extensions with Weak Constraints**

- if $P$ is an extension path such that the unlabeled endpoint has degree 3
  - Branch along possible labelings of $v$, $w$ and eventually $u$ extend these labelings to entire $P$.

By Rule 1, neither $v_1$ nor $v_2$ are labeled or adjacent to a labeled vertex $\Rightarrow w(v_1) = w(v_2) = 1$. And thus $u \neq v_1$ and $u \neq v_2$. 
An algorithm for \( L(2, 1) \)-labelings of span 4

→ If none of Rules 1-4 can be applied, every unlabeled vertex adjacent to a labeled vertex belongs to an extension path with one unlabeled endpoint of degree 3.

**Rule 5 - Extensions with Weak Constraints**

- if \( P \) is an extension path such that the unlabeled endpoint has degree 3

⇒ Branch along possible labelings of \( v, w \) and eventually extend these labelings to entire \( P \).

By Rule 1, neither \( v_1 \) nor \( v_2 \) are labeled or adjacent to a labeled vertex ⇒ \( w(v_1) = w(v_2) = 1 \). And thus \( u \neq v_1 \) and \( u \neq v_2 \).
An algorithm for $L(2, 1)$-labelings of span 4

Labeling $P$ and the vertex $u$ would decrease the measure $\mu(G)$ by:

**if $u$ exists.** $\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon + \epsilon$

$\Downarrow$ $\epsilon$ for the first vertex of $P$

$\Downarrow$ length($P$) $- 1$ for the other vertices of $P$

$\Downarrow$ $2 - 2\epsilon$ for vertices $v_1$ and $v_2$

$\Downarrow$ $\epsilon$ for vertex $u$

**if $u$ do not exists.** $\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon$

$\Downarrow$ $\epsilon$ for the first vertex of $P$

$\Downarrow$ length($P$) $- 1$ for the other vertices of $P$

$\Downarrow$ $2 - 2\epsilon$ for vertices $v_1$ and $v_2$
An algorithm for $L(2, 1)$-labelings of span 4

If $\text{length}(P) \leq 8$. Number of branchings:

<table>
<thead>
<tr>
<th>length($P$)</th>
<th>number of branchings if $\deg(b) = 2$</th>
<th>number of branchings if $\deg(b) = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

If $\text{length}(P) \geq 9$. If $\deg(b) = 2$, there are 6 possible labelings of $v$ and $w$; otherwise, there are 12 possible labelings of $v$, $w$ and $u$. 
An algorithm for $L(2, 1)$-labelings of span 4

Setting $\epsilon = 0.819$ in the measure

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

and solving the corresponding recurrences establishes:

**Theorem.** The computation of an $L(2, 1)$-labeling of span 4, if one exists, can be done in time $O(1.3006^n)$. 
An algorithm to compute the minimum span

1. Introduction

2. Definition of $L(2, 1)$-labelings and known results

3. Branching algorithm for span 4 labelings

4. A fast algorithm to compute the minimum span

5. Conclusion
Dynamic programming for $L(2,1)$-labeling

Simple idea: fill-in table $T_\ell$ corresponding to partial labelings using up to $\ell$ labels.

span 3

Table $T_3$
Dynamic programming for $L(2,1)$-labeling

Simple idea: fill-in table $T_\ell$ corresponding to partial labelings using up to $\ell$ labels.

Span 3 table $T_3$
Dynamic programming for $L(2,1)$-labeling

Simple idea: fill-in table $T_\ell$ corresponding to partial labelings using up to $\ell$ labels.
Dynamic programming for $L(2,1)$-labeling

Simple idea: fill-in table $T_\ell$ corresponding to partial labelings using up to $\ell$ labels.

Use a compact representation for partial labelings

+ reduce the number of algebraic operations to compute next tables
Representation of partial $L(2, 1)$-labelings

Table $T_{\ell}$ contains a vector $\vec{a} \in \{0, \overline{0}, 1, \overline{1}\}^n$ if and only if there is a partial labeling $\varphi : V \rightarrow \{0, \ldots, \ell\}$ such that:

- $a_i = 0$ iff $v_i$ is not labeled by $\varphi$ and there is no neighbor $u$ of $v_i$ with $\varphi(u) = \ell$
- $a_i = \overline{0}$ iff $v_i$ is not labeled by $\varphi$ and there is a neighbor $u$ of $v_i$ with $\varphi(u) = \ell$
- $a_i = 1$ iff $\varphi(v_i) < \ell$
- $a_i = \overline{1}$ iff $\varphi(v_i) = \ell$
Representation of partial $L(2, 1)$-labelings

Table $T_3$
Computing the tables

How to compute table $T_{\ell+1}$ from table $T_{\ell}$?

Let $P \subseteq \{0, 1\}^n$ be the encodings of all 2-packings of $G$. Formally, $\bar{p} \in P \iff \exists$ a 2-packing $S \subseteq V$ such that $\forall i, p_i = 1 \text{ iff } v_i \in S$.

Compute $T_{\ell+1}$ from “$T_{\ell} \oplus P$”.

Define the partial function $\oplus : \{0, \bar{0}, 1, \bar{1}\} \times \{0, 1\} \rightarrow \{0, 1, \bar{1}\}$ as:

<table>
<thead>
<tr>
<th></th>
<th>$0$</th>
<th>$\bar{0}$</th>
<th>$1$</th>
<th>$\bar{1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>$1$</td>
<td>$\bar{1}$</td>
<td>$\sim$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

Entry “$-$” signifies that $\oplus$ is not defined.

Generalization of $\oplus$ to vectors:

$a_1 a_2 \ldots a_n \oplus b_1 b_2 \ldots b_n = \begin{cases} (a_1 \oplus b_1) \ldots (a_n \oplus b_n) & \text{if } \oplus \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$
Computing the tables

Then $T_{\ell} \oplus P$ is already almost the same as $T_{\ell+1}$:

$$\vec{a} \in T_{\ell+1} \text{ iff there is an } \vec{a}' \in T_{\ell} \oplus P \text{ such that}$$

- $a_i = 0$ iff $a'_i = 0$ and there is no $v_j \in N(v_i)$ with $a'_j = 1$
- $a_i = \overline{0}$ iff $a'_i = 0$ and there is a $v_j \in N(v_i)$ with $a'_j = 1$
- $a_i = 1$ iff $a'_i = 1$
- $a_i = \overline{1}$ iff $a'_i = \overline{1}$
Computing the tables

How to compute $T_{\ell} \oplus P$ rapidly?

**Definition.** $A_w = \{ \vec{v} \mid w \cdot v \in A \}$

<table>
<thead>
<tr>
<th>$\oplus$</th>
<th>0</th>
<th>0̄</th>
<th>1</th>
<th>1̄</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1̄</td>
<td>$\sim$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

example: $\oplus \begin{pmatrix} 0, 0, 0̄, 1, 1̄ \end{pmatrix}$

$A \oplus B = 0((A_0 \cup A_0) \oplus B_0) \cup 1((A_1 \cup A_1) \oplus B_0) \cup 1(A_0 \oplus B_1)$

where $A := T_{\ell}$ (partial labelings) and $B := P$ (encodings of the 2-packings)
Computing the tables

\[
\begin{array}{c|cccc}
\oplus & 0 & \overline{0} & 1 & \overline{1} \\
0 & 0 & 0 & 1 & 1 \\
1 & \overline{1} & \sim & \_ & \_ \\
\end{array}
\]

two adjacent vertices

\[
\begin{array}{cccccccccccccccc}
\oplus & 00 & 0\overline{0} & 01 & 0\overline{1} & \overline{00} & \overline{0}1 & \overline{0}\overline{1} & 10 & 1\overline{0} & 11 & 1\overline{1} & \overline{10} & \overline{1}0 & \overline{1}1 & \overline{1}\overline{1} \\
00 & & & & & & & & & & & & & & \\
01 & & & & & & & & & & & & & & \\
10 & & & & & & & & & & & & & & \\
11 & & & & & & & & & & & & & & \\
\end{array}
\]
Computing the tables

<table>
<thead>
<tr>
<th>⊕</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>~</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**two adjacent vertices**
Computing the tables

<table>
<thead>
<tr>
<th>⊕</th>
<th>0</th>
<th>(\bar{0})</th>
<th>1</th>
<th>(\bar{1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(\bar{1})</td>
<td>(\sim)</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

two adjacent vertices

| ⊕ | 00 | 0\(\bar{0}\) | 01 | 0\(\bar{1}\) | 00 | 0\(\bar{0}\) | 01 | 0\(\bar{1}\) | 10 | 1\(\bar{0}\) | 11 | 1\(\bar{1}\) | 10 | 1\(\bar{1}\) | 11 | 1\(\bar{1}\) |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 00 | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 01 | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 10 | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| 11 | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
Computing the tables

\[ \begin{array}{c|cccc}
\oplus & 0 & \bar{0} & 1 & \bar{1} \\
\hline
0 & 0 & 0 & 1 & 1 \\
1 & \bar{1} & \sim & - & - \\
\end{array} \]

two adjacent vertices

\[ \begin{array}{cccccccccccccccc}
\oplus & 00 & 0\bar{0} & 01 & 0\bar{1} & \bar{0}0 & \bar{0}\bar{1} & 01 & \bar{0}1 & 10 & 1\bar{0} & 11 & 1\bar{1} & \bar{1}0 & \bar{1}1 & \bar{1}\bar{1} \\
\hline
00 & \sim & - & - & - & \sim & - & - & - & \sim & - & - & - & \sim & - & - & - \\
\end{array} \]
Computing the tables

<table>
<thead>
<tr>
<th>⊕</th>
<th>0</th>
<th>0̅</th>
<th>1</th>
<th>1̅</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1̅</td>
<td>~</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

two adjacent vertices

<table>
<thead>
<tr>
<th>⊕</th>
<th>00</th>
<th>00̅</th>
<th>01</th>
<th>01̅</th>
<th>0̅0</th>
<th>0̅0̅</th>
<th>0̅1</th>
<th>0̅1̅</th>
<th>10</th>
<th>1̅0</th>
<th>11</th>
<th>1̅1</th>
<th>1̅0</th>
<th>1̅0</th>
<th>1̅1</th>
<th>1̅1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>0̅0</td>
<td>0̅0</td>
<td>0̅1</td>
<td>0̅1</td>
<td>10</td>
<td>1̅0</td>
<td>11</td>
<td>1̅1</td>
<td>1̅0</td>
<td>1̅0</td>
<td>1̅1</td>
<td>1̅1</td>
</tr>
<tr>
<td>01</td>
<td>0̅1</td>
<td>~</td>
<td>−</td>
<td>−</td>
<td>0̅1</td>
<td>0̅1</td>
<td>~</td>
<td>~</td>
<td>1̅1</td>
<td>~</td>
<td>−</td>
<td>−</td>
<td>1̅1</td>
<td>~</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>10</td>
<td>1̅0</td>
<td>1̅0</td>
<td>1̅1</td>
<td>1̅1</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>11</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

36/44
Computing the tables

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>~</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

two adjacent vertices

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>-</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>~</td>
<td>-</td>
<td>-</td>
<td>01</td>
<td>~</td>
<td>-</td>
<td>-</td>
<td>11</td>
<td>~</td>
<td>-</td>
<td>-</td>
<td>11</td>
<td>~</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

→ Prefix 11 cannot appear.
## Computing the tables

<table>
<thead>
<tr>
<th>☀</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>01</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

\[ A \oplus B = \]
Computing the tables

<table>
<thead>
<tr>
<th>⊕</th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>01</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

\[ A \oplus B = 00((A_{00} \cup A_{00} \cup A_{00} \cup A_{00}) \oplus B_{00}) \]
## Computing the tables

<table>
<thead>
<tr>
<th>⊕</th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>01</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

\[
A \oplus B = 00((A_{00} \cup A_{00} \cup A_{00} \cup A_{00}) \oplus B_{00}) \\
\cup 01((A_{01} \cup A_{01} \cup A_{01} \cup A_{01}) \oplus B_{00})
\]
Computing the tables

\[
\begin{array}{cccccccccccccc}
\oplus & 00 & 0\bar{0} & 01 & 0\bar{1} & \bar{0}0 & \bar{0}\bar{0} & \bar{0}1 & \bar{0}\bar{1} & 10 & 1\bar{0} & 11 & 1\bar{1} & 10 & 1\bar{1} & 11 & 1\bar{1} \\
00 & 00 & 00 & 01 & 01 & 00 & 00 & 01 & 01 & 10 & 10 & 11 & 11 & 10 & 10 & 11 & - \\
01 & 0\bar{1} & \sim & \sim & \sim & 0\bar{1} & \sim & \sim & \sim & 1\bar{1} & \sim & \sim & \sim & 1\bar{1} & \sim & \sim & - \\
10 & 10 & 10 & 1\bar{1} & 1\bar{1} & \sim & \sim & \sim & \sim & \sim & \sim & \sim & \sim & \sim & \sim & \sim & - \\
\end{array}
\]

\[
A \oplus B = 00((A_{00} \cup A_{\bar{0}\bar{0}} \cup A_{0\bar{0}} \cup A_{\bar{0}0}) \oplus B_{00})
\]

\[
\cup 01((A_{01} \cup A_{\bar{0}\bar{1}} \cup A_{0\bar{1}} \cup A_{\bar{0}1}) \oplus B_{00})
\]

\[
\cup 10((A_{10} \cup A_{\bar{1}\bar{0}} \cup A_{1\bar{0}} \cup A_{\bar{1}0}) \oplus B_{00})
\]

\[
\cup 11((A_{11} \cup A_{\bar{1}\bar{1}} \cup A_{1\bar{1}} \cup A_{\bar{1}1}) \oplus B_{00})
\]

\[
\cup 0\bar{1}((A_{00} \cup A_{\bar{0}\bar{0}}) \oplus B_{01})
\]

\[
\cup 1\bar{1}((A_{10} \cup A_{\bar{1}\bar{0}}) \oplus B_{01})
\]

\[
\cup \bar{1}0((A_{00} \cup A_{\bar{0}\bar{0}}) \oplus B_{10})
\]

\[
\cup \bar{1}1((A_{01} \cup A_{\bar{0}\bar{1}}) \oplus B_{10})
\]
### Computing the tables

<table>
<thead>
<tr>
<th>⊕</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>01</th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>01</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>11</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>11</td>
<td>∼</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
</tr>
<tr>
<td>11</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
<td>∼</td>
</tr>
</tbody>
</table>

\[
A \oplus B = \begin{align*}
00 & (\bigcup (A_{00} \cup A_{01} \cup A_{00} \cup A_{00}) \oplus B_{00}) \\
\cup & 01((A_{01} \cup A_{01} \cup A_{01} \cup A_{01}) \oplus B_{00}) \\
\cup & 10((A_{10} \cup A_{10} \cup A_{10} \cup A_{10}) \oplus B_{00}) \\
\cup & 11((A_{11} \cup A_{11} \cup A_{11}) \oplus B_{00}) \\
\cup & 01((A_{00} \cup A_{00}) \oplus B_{01}) \\
\cup & 11((A_{10} \cup A_{10}) \oplus B_{01}) \\
\cup & 00((A_{00} \cup A_{00}) \oplus B_{10}) \\
\cup & 10((A_{01} \cup A_{01}) \oplus B_{10})
\end{align*}
\]

**Running-time:** \( T(n) = 8 \cdot T(n - 2) = 8^{n/2} < 2.8285^n \)
Extension to larger prefixes

**Theorem.** The minimum span of an L(2,1)-labeling can be computed in time $O(2.6488^n)$.

We need further results:

- instead of considering 2 adjacent vertices, consider $k' = O(1)$ vertices;
- consider prefix of larger length, when it makes sense for $\oplus$ operation;
- show that any connected graph can be “partitioned” into sufficiently large connected subgraphs of size about $k'$;
- establish a combinatorial upper-bound on the number of proper pairs.
Decomposing the graph into connected subgraphs

**Theorem.** Let $G$ be a connected graph and let $k < n$. Then there exist connected subgraphs $G_1, G_2, \ldots, G_q$ of $G$ s.t.

(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 $|V(G_q)| \leq 2k$)

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

**Additional Result 1**
2-Packings and Proper Pairs

*Independent sets* are related to colorings, but *2-packings* to \( L(2, 1) \)-labelings.

**Definition.** 2-packings = Independent Sets in \( G^2 \).

A subset \( S \subseteq V \) s.t. \( \forall u, v \in S, N[u] \cap N[v] = \emptyset \) is a 2-packing.

**Definition.** A pair \((S, X)\) of subsets of \( V \) is a **proper pair** if \( S \cap X = \emptyset \) and \( S \) is a 2-packing.

**Definition.** The number of proper pairs in a graph \( G \) is given by

\[
pp(G) = \sum_{2\text{-packings } S} 2^{n-|S|}
\]

Let \( pp(n) = \max pp(G) \) be the maximum number of proper pairs in a connected graph with \( n \) vertices.

**Theorem.** \( 2.6117^n \leq pp(n) \leq 2.6488^n \)
An exact algorithm – Running-time analysis

Let \( A \subseteq \{0, \overline{0}, 1, \overline{1}\}^n \) and \( B \subseteq \{0, 1\}^n \) and \( k' < n' \).

Compute \( A \oplus B \) in the following way:

\[
A \oplus B = \bigcup_{\vec{u} \in \{0,0,1,\overline{1}\}^{k'}} \bigcup_{\vec{v} \in \{0,1\}^{k'}} \left( (\vec{u} \oplus \vec{v})(A_{\vec{u}} \oplus B_{\vec{v}}) \right)
\]

\[
= \bigcup_{\vec{v} \in \{0,1\}^{k'}} \bigcup_{\vec{w} \in \{0,1,\overline{1}\}^{k'}} \left( \bigcup_{\vec{u} \in \{0,0,1,\overline{1}\}^{k'}} A_{\vec{u}} \right) \oplus B_{\vec{v}}
\]

Remark:
\( \oplus \) computation can be omitted whenever \( \left( \bigcup_{\vec{u} \in \{0,0,1,\overline{1}\}^{k'}} A_{\vec{u}} \right) \) is empty.
An exact algorithm – Running-time analysis

How many pairs \( \vec{v}, \vec{w} \) s.t. there is at least one \( \vec{u} \) with \( \vec{u} \oplus \vec{v} = \vec{w} \)?

If \( \vec{v} \) is fixed, then \( v_i = 1 \Rightarrow w_i = 1 \).

Thus, for a fixed \( \vec{v} \) there are at most \( 2^{k'} - ||\vec{v}|| \) many \( \vec{w} \)'s, where \( ||\vec{v}|| \) denotes the number of positions \( i \) such that \( v_i = 1 \).

The total number of pairs \( \vec{v}, \vec{w} \) such that \( \vec{w} = \vec{u} \oplus \vec{v} \) for some \( \vec{u} \) is therefore at most

\[
\sum_{\vec{v} \in \{0,1\}^{k'}} 2^{k'} - ||\vec{v}|| \leq pp(k')
\]

\( \Rightarrow \) We need to make \( pp(k') \) recursive computations of \( \oplus \) on sets of vectors of length \( n - k' \).

**Theorem.** The minimum span of an \( L(2,1) \)-labeling can be computed in time \( \mathcal{O}(2.6488^n) \).
An exact algorithm – Running-time analysis

How many pairs $\vec{v}, \vec{w}$ s.t. there is at least one $\vec{u}$ with $\vec{u} \oplus \vec{v} = \vec{w}$?

If $\vec{v}$ is fixed, then $v_i = 1 \Rightarrow w_i = 1$.
Thus, for a fixed $\vec{v}$ there are at most $2^{k' - ||\vec{v}||}$ many $\vec{w}$'s, where $||\vec{v}||$ denotes the number of positions $i$ such that $v_i = 1$.

The total number of pairs $\vec{v}, \vec{w}$ such that $\vec{w} = \vec{u} \oplus \vec{v}$ for some $\vec{u}$ is therefore at most

$$\sum_{\vec{v} \in \{0,1\}^{k'}} 2^{k' - ||\vec{v}||} \leq pp(k')$$

$\Rightarrow$ We need to make $pp(k')$ recursive computations of $\oplus$ on sets of vectors of length $n - k'$.

**Theorem.** The minimum span of an L(2,1)-labeling can be computed in time $O(2.6488^n)$. 
An exact algorithm – Running-time analysis

How many pairs \( \vec{v}, \vec{w} \) s.t. there is at least one \( \vec{u} \) with \( \vec{u} \oplus \vec{v} = \vec{w} \)?

If \( \vec{v} \) is fixed, then \( v_i = 1 \Rightarrow w_i = 1 \).

Thus, for a fixed \( \vec{v} \) there are at most \( 2^{k' - ||\vec{v}||} \) many \( \vec{w} \)'s, where \( ||\vec{v}|| \) denotes the number of positions \( i \) such that \( v_i = 1 \).

The total number of pairs \( \vec{v}, \vec{w} \) such that \( \vec{w} = \vec{u} \oplus \vec{v} \) for some \( \vec{u} \) is therefore at most

\[
\sum_{\vec{v} \in \{0,1\}^{k'}} 2^{k' - ||\vec{v}||} \leq pp(k')
\]

\( \Rightarrow \) We need to make \( pp(k') \) recursive computations of \( \oplus \) on sets of vectors of length \( n - k' \).

**Theorem.** The minimum span of an L(2,1)-labeling can be computed in time \( O(2.6488^n) \).
An exact algorithm – Running-time analysis

How many pairs $\vec{v}, \vec{w}$ s.t. there is at least one $\vec{u}$ with $\vec{u} \oplus \vec{v} = \vec{w}$?

If $\vec{v}$ is fixed, then $v_i = 1 \Rightarrow w_i = 1$.

Thus, for a fixed $\vec{v}$ there are at most $2^{k'} - ||\vec{v}||$ many $\vec{w}$'s, where $||\vec{v}||$ denotes the number of positions $i$ such that $v_i = 1$.

The total number of pairs $\vec{v}, \vec{w}$ such that $\vec{w} = \vec{u} \oplus \vec{v}$ for some $\vec{u}$ is therefore at most

$$\sum_{\vec{v} \in \{0,1\}^{k'}} 2^{k'} - ||\vec{v}|| \leq pp(k')$$

⇒ We need to make $pp(k')$ recursive computations of $\oplus$ on sets of vectors of length $n - k'$.

**Theorem.** The minimum span of an L(2,1)-labeling can be computed in time $O(2.6488^n)$. 
An exact algorithm – Running-time analysis

How many pairs \( \vec{v}, \vec{w} \) s.t. there is at least one \( \vec{u} \) with \( \vec{u} \oplus \vec{v} = \vec{w} \)?

If \( \vec{v} \) is fixed, then \( v_i = 1 \) \( \Rightarrow \) \( w_i = \overline{1} \).

Thus, for a fixed \( \vec{v} \) there are at most \( 2^{k' - ||\vec{v}||} \) many \( \vec{w} \)'s, where \( ||\vec{v}|| \) denotes the number of positions \( i \) such that \( v_i = 1 \).

The total number of pairs \( \vec{v}, \vec{w} \) such that \( \vec{w} = \vec{u} \oplus \vec{v} \) for some \( \vec{u} \) is therefore at most

\[
\sum_{\vec{v} \in \{0,1\}^{k'}} 2^{k' - ||\vec{v}||} \leq pp(k')
\]

\( \Rightarrow \) We need to make \( pp(k') \) recursive computations of \( \oplus \) on sets of vectors of length \( n - k' \).

**Theorem.** The minimum span of an L(2,1)-labeling can be computed in time \( \mathcal{O}(2.6488^n) \).
Conclusion

1. Introduction

2. Definition of $L(2,1)$-labelings and known results

3. Branching algorithm for span 4 labelings

4. A fast algorithm to compute the minimum span

5. Conclusion
Conclusion

**Short summary.**

- decide span 4 : $O(1.3006^n)$
- solving $L(2,1)$ in time $O(2.6488^n)$ (best known algo)

It is also possible to consider **counting** and **enumeration** versions of the problem:

- count span 4 : $O(1.1269^n)$ (exp-space)  
  [CGKLP,2013]
- enumerate span 5 in cubic graphs : $O(1.7990^n)$  
  [CGKLP,2013]
Conclusion

**Short summary.**
- decide span 4: $O(1.3006^n)$
- solving $L(2,1)$ in time $O(2.6488^n)$ (best known algo)

It is also possible to consider **counting** and **enumeration** versions of the problem:
- count span 4: $O(1.1269^n)$ (exp-space) \[\text{[CGKLP, 2013]}\]
- enumerate span 5 in cubic graphs: $O(1.7990^n)$ \[\text{[CGKLP, 2013]}\]

**Interesting questions.**
- Does a clever choice of the measure $\mu(G)$ can help to improve significantly the running time analysis?
- Is it possible to solve $L(2,1)$-labeling faster? E.g. in $O^*(2^n)$-time? To establish lower-bound, via ETH?
“For every polynomial-time algorithm you have, there is an exponential algorithm that I would rather run.”

[Alan Perlis\textsuperscript{1}]

co-authors of the presented works:

Frédéric Havet, Konstanty Junosza-Szaniawski, Martin Klazar, Jan Kratochvíl, Dieter Kratsch, Peter Rossmanith, Pawel Rzązewski

\textsuperscript{1} first recipient of the Turing price, 1966
“For every polynomial-time algorithm you have, there is an exponential algorithm that I would rather run.”

[Alan Perlis\textsuperscript{1}]

Merci!

co-authors of the presented works:
Frédéric Havet  Konstanty Junosza-Szaniawski  Martin Klazar
Jan Kratochvíl  Dieter Kratsch  Peter Rossmanith  Pawel Rzazewski

\textsuperscript{1}. first recipient of the Turing price, 1966
Definition and results

$L(2, 1)$-lab of span 4

minimum $L(2, 1)$-lab span
Decomposing the graph into connected subgraphs (proof)

Proof

1. Consider a DFS-tree $T$ of $G$ rooted at $r$.
2. For every $v$ let $T(v)$ be the subtree rooted in $v$.
3. If $|T(r)| \leq 2k$ then add $G$ to the set of desired subgraphs and stop.
4. 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 proceed recursively with $G \setminus V(T(v))$. 
Decomposing the graph into connected subgraphs (proof)

**Proof**

- 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 $\{u_1, \ldots, u_i\}$ of children of $v$ such that $k - 1 \leq |T(u_1)| + \cdots + |T(u_i)| \leq 2k - 1$.

Add $G[\{v\} \cup V(T(u_1)) \cup \cdots \cup V(T(u_i))]$ to the set of desired subgraphs and proceed recursively with $G \setminus (V(T(u_1)) \cup \cdots \cup V(T(u_i)))$.

- 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.
Definition and results

$L(2, 1)$-lab of span 4

minimum $L(2, 1)$-lab span
Definition and results

L(2, 1)-lab of span 4 minimum L(2, 1)-lab span

2-Packings and Proper Pairs (proof)

Proof.
Let \( G = (V, E) \) be a connected graph.

Fact 1. If \( S \) is a 2-packing, then \( S \) is also a 2-packing of \( G = (V, E \setminus e) \), for any edge \( e \).
\[ \Rightarrow \text{we can assume } G \text{ to be a tree.} \]

Fact 2. Suppose that there are two leaves which have a common neighbor. Every 2-packing in \( G \) is also a 2-packing in \( H \).
\[ \Rightarrow \text{we can assume that there are no two or more leaves with a common neighbor} \]
Proof. 

(A) If \( \deg(c) \leq 2 \) then

\[
\begin{array}{c}
\quad v \\
\quad \quad u \\
\quad \quad \quad c \\
\quad \quad \quad \quad \ldots
\end{array}
\]

\[
pp(n) \leq 2 \cdot pp(n - 1) + 4 \cdot pp(n - 3)
\]

(B) If \( \deg(c) > 2 \) and

\[
\begin{array}{c}
w_1 \\
w_2 \\
w_3 \\
w_q
\end{array}
\]

\[
\begin{array}{c}
u_1 \\
u_2 \\
u_3 \\
u_q
\end{array}
\]


\[
\begin{array}{c}
x
\end{array}
\]

\[
\begin{array}{c}
w_1 \\
w_2 \\
w_q
\end{array}
\]

\[
\begin{array}{c}
u_1 \\
u_2 \\
u_q
\end{array}
\]

for \( q \geq 2 \)

(B0) no neighbor of \( c \) is a leaf ...

\[
pp(n) \leq 2^{2q} \cdot pp(n - 2q) + (3^{q-1} \cdot 2^{q+1} \cdot (3 + q) - 2^{2q+1}) \cdot pp(n - 2q - 1)
\]

(B1) one neighbor of \( c \) is a leaf ...

\[
pp(n) \leq 2^{2q+1} \cdot pp(n - 2q - 1) + (3^{q-1} \cdot 2^{q+1} \cdot (9 + 2q) - 2^{2q+2}) \cdot pp(n - 2q - 2)
\]
2-Packings and Proper Pairs (proof)

To show the lower bound, we consider the following graphs:

\[ a_k = 2b_{k-1} + 4a_{k-1} \]
\[ b_k = 2c_k + 2d_k \]
\[ c_k = 2a_k + 12d_{k-1} \]
\[ d_k = 4d_{k-1} + 12a_{k-1} \]

Theorem. \[ 2.6117^n \leq pp(n) \leq 2.6488^n \]
Definition and results

$L(2, 1)$-lab of span $4$

Minimum $L(2, 1)$-lab span
An exact algorithm – Running-time analysis

By Theorem (⋆), the total length of the vectors is $n' \leq n(1 + 1/k)$.

In each recursive computation:

- Prepare up to $pp(k')$ many pairs of sets of vectors of length $n' - k'$
- Recursively compute $\oplus$ on these pairs
- From the result, compute $T_{\ell+1}$ in linear time
- The size of $B$ is at most $O(n2^{n'})$ bits
- The size of $A$ is at most $O(npp(n'))$ bits:
  the $\overline{1}$'s form a 2-packing and there are only two possibilities (1 or 0/0) for the other nodes.

Thus the running-time is given by

$$T(n) \leq O(n \cdot pp(n') + pp(k') \cdot T(n' - k'))$$

where $k \leq k' \leq 2k$. 
The solution of

\[ T(n) \leq O(n \cdot pp(n') + pp(k') \cdot T(n' - k')) \]

is

\[ T(n) = O^*(pp(n')) = O^*(pp(n(1 + 1/k))) \]

Choosing constant \( k \) big enough:

\[ T(n) = O(2.6488^n) \]