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)*

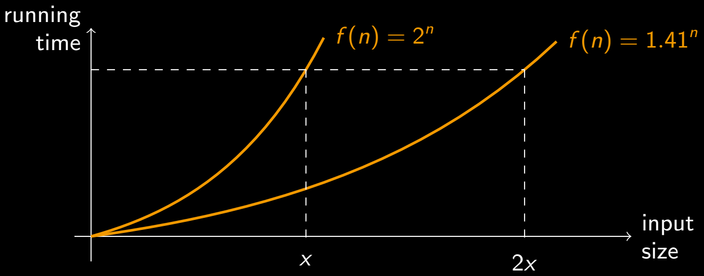
Kurt Gödel to **John von Neumann** (1956) :

≪ *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 :

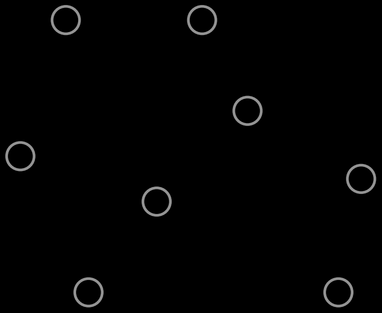
computing $L(2,1)$-**labelings** in graphs.

- 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.

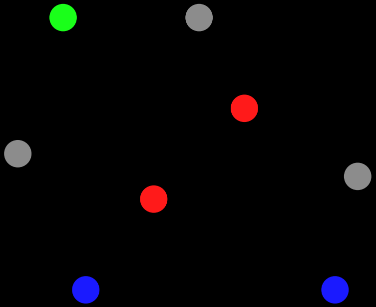
- broadcast network
- assign frequencies to transmitters
- avoid undesired interference



In this talk we give **moderately exponential-time algorithms** for a frequency assignment problem :

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

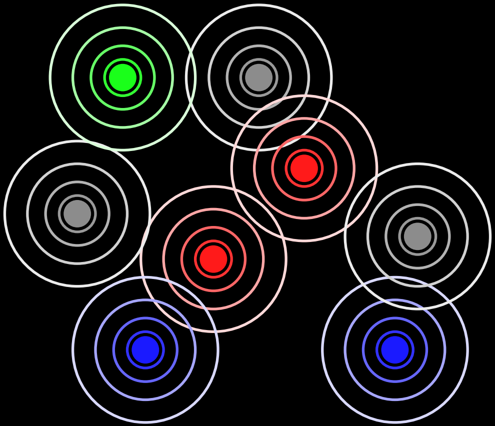
- broadcast network
- assign frequencies to transmitters
- avoid undesired interference



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

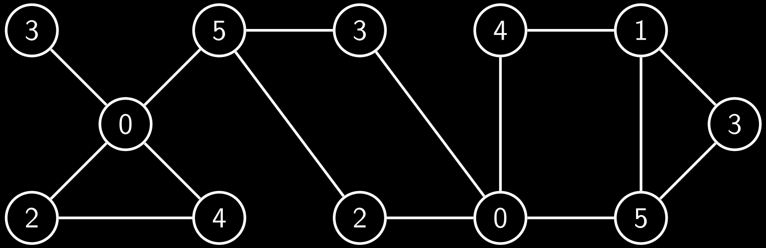
An algorithm to compute the minimim span

Definition of $L(2,1)$-labeling

$L(2,1)$-LABELING

Input : A graph $G = (V, E)$.
Question : Compute a function ℓ of minimum span k
$\ell : V \to \{0, \dots, k\}$ 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 [Griggs and Yeh, 1992] [Fiala, Kloks, Kratochvíl, 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 [Chang, Kuo 1996] [Fiala, Golovach, Kratochvíl, 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 [Fiala, Golovach, Kratochvíl, 2005] [Janczewski, Kosowski, Małafiejski, 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 : $\mathcal{O}(1.3006^n)$ (poly-space) [HKKKL,2011]
- count span 4 : $\mathcal{O}(1.1269^n)$ (exp-space) [CGKLP,2013]
- enumerate span 5 in cubic graphs : $\mathcal{O}(1.7990^n)$ [CGKLP,2013]

Computing the minimum span k :

- polynomial space :
  - $\mathcal{O}^*((k-2.5)^n)$ [HKKKL,2011]
  - $\mathcal{O}(7.50^n)$ [JSKLR,2012]
  - $\mathcal{O}(3.4642^n)$ [Kowalik, Socala,2014]
- exponential space :
  - $\mathcal{O}^*(4^n)$ [Král',2006]
  - $\mathcal{O}^*(15^{n/2}) = \mathcal{O}(3.88^n)$ [HKKKL,2011]
  - $\mathcal{O}^*(3^n)$ [Cygan, Kowalik,2011]
  - $\mathcal{O}^*(2.6488^n)$ [JSKLRR,2013]

An algorithm to compute the minimim span

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

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

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

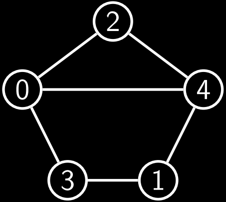
locally injective homomorphism (LIH) : A homomorphism $f : G \to H$ is locally injective if for every vertex $u \in V(G)$ its neighborhood is mapped injectively into the neighborhood of $f(u)$ in H, i.e., every two vertices having a common neighbor in G are mapped onto disctinct vertices in H.

Fiala and Kratochvíl, 2002 :

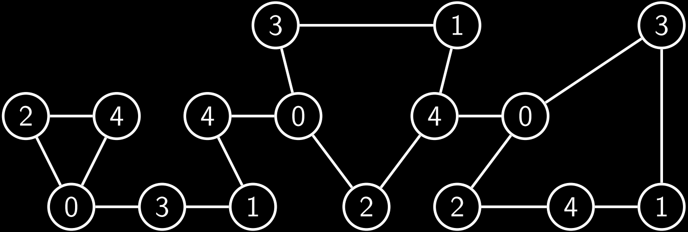
Theorem. $L(2,1)$-labelings of span k are locally injective homomorphisms into the complement of the path of length k.

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

$\overline{P_5}$:

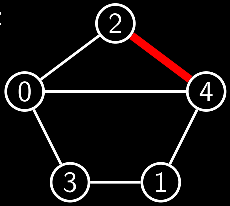


G :

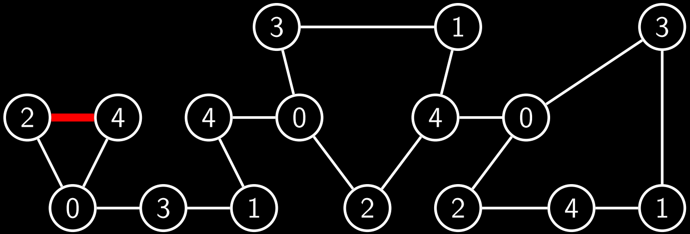


$L(2,1)$-labelings and LIH

$\overline{P_5}$:



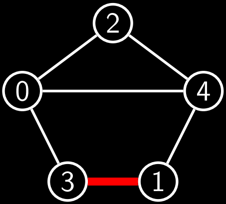
G :



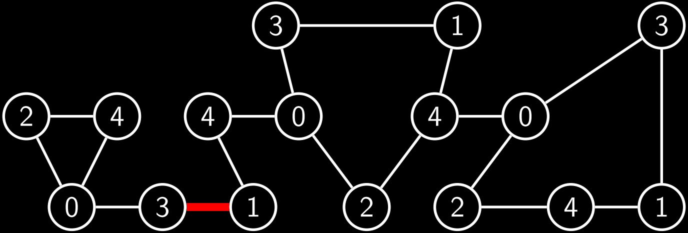
$uv \in E(G) \Rightarrow f(u)f(v) \in E(H)$

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

$\overline{P_5}$:



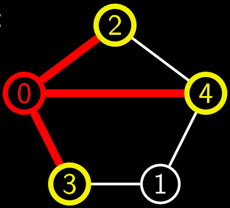
G :



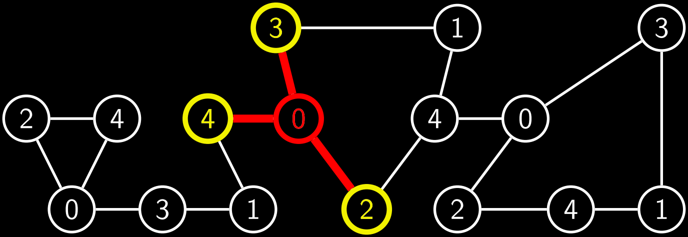
$uv \in E(G) \Rightarrow f(u)f(v) \in E(H)$

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

$\overline{P_5}$:



G :

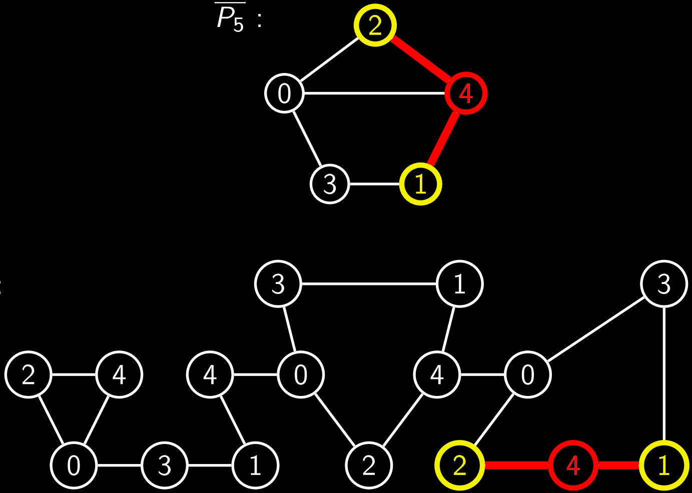


$u \in V(G) \Rightarrow N(u)$ is mapped injectively on $N(f(u))$ in H

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

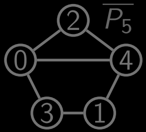
$\overline{P_5}$:

G :

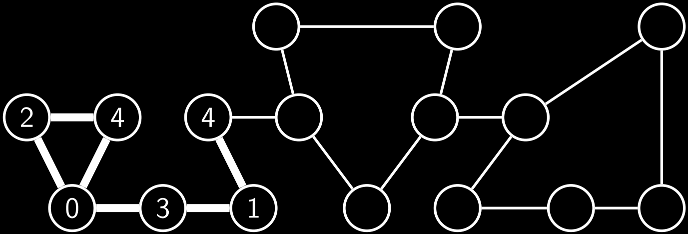


$u \in V(G) \Rightarrow N(u)$ is mapped injectively on $N(f(u))$ in H

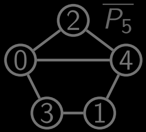
$L(2,1)$-labelings and LIH



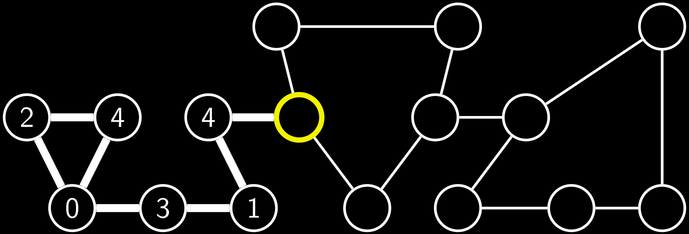
$L(2,1)$-labelings of span 4 can trivially be decided in $\mathcal{O}(2^n)$ time.



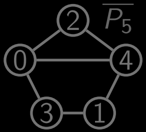
$L(2,1)$-labelings and LIH



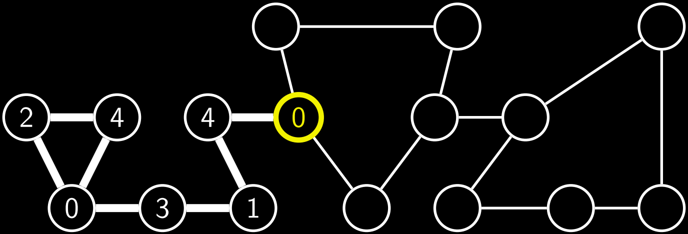
$L(2,1)$-labelings of span 4 can trivially be decided in $\mathcal{O}(2^n)$ time.



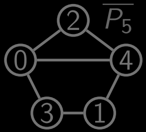
$L(2,1)$-labelings and LIH



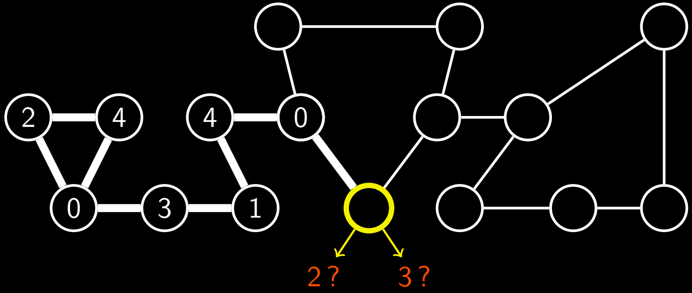
$L(2,1)$-labelings of span 4 can trivially be decided in $\mathcal{O}(2^n)$ time.



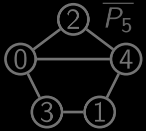
$L(2,1)$-labelings and LIH



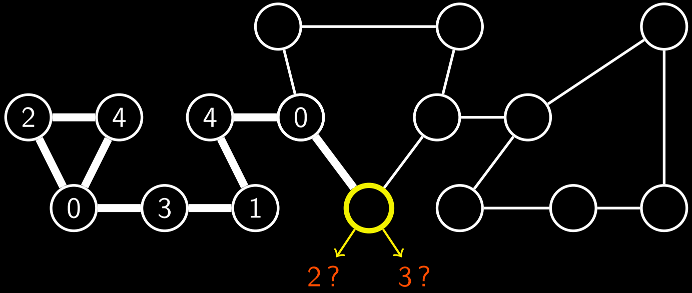
$L(2,1)$-labelings of span 4 can trivially be decided in $\mathcal{O}(2^n)$ time.



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

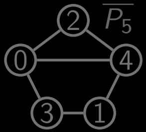


$L(2,1)$-labelings of span 4 can trivially be decided in $\mathcal{O}(2^n)$ time.



Something faster ?

An algorithm for $L(2,1)$-labelings of span 4 (1/5)

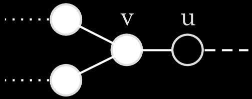


Description of the rules of the algorithm :

Rule 1 - Forced Extensions

- if u is unlabeled and its labeled neighbor v has two labeled neighbors

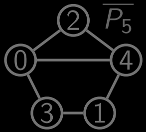
$\Rightarrow$ label of u is forced



- 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 (1/5)

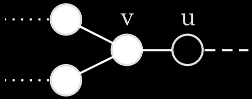


Description of the rules of the algorithm :

Rule 1 - Forced Extensions

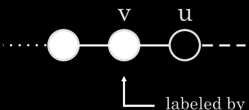
- if u is unlabeled and its labeled neighbor v has two labeled neighbors

$\Rightarrow$ label of u is forced

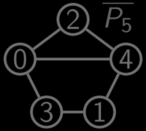


- if u is unlabeled and its labeled neighbor v has label 1, 2 or 3

$\Rightarrow$ label of u is forced

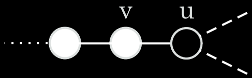


An algorithm for $L(2,1)$-labelings of span 4 (1/5)



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

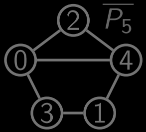
$\Rightarrow$ label of u is forced



- if u is unlabeled, $d(u) = 2$ and u has a labeled neighbor v and a (possibly unlabeled) neighbor of degree 3

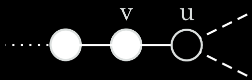
$\Rightarrow$ label of u is forced

An algorithm for $L(2,1)$-labelings of span 4 (1/5)



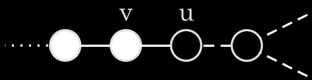
- if u is unlabeled, $d(u) = 3$ and u has a labeled neighbor v

$\Rightarrow$ label of u is forced

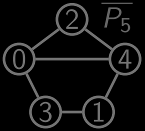


- if u is unlabeled, $d(u) = 2$ and u has a labeled neighbor v and a (possibly unlabeled) neighbor of degree 3

$\Rightarrow$ label of u is forced



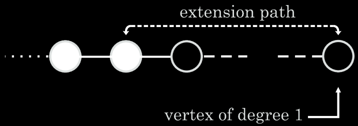
An algorithm for $L(2,1)$-labelings of span 4 (2/5)



Rule 2 - Easy Extension

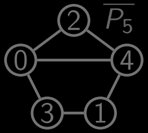
- if P is an extension path with one endpoint of degree 1

$\Rightarrow$ 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.

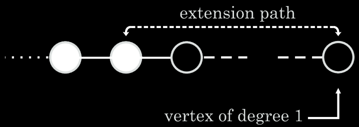
An algorithm for $L(2,1)$-labelings of span 4 (2/5)



Rule 2 - Easy Extension

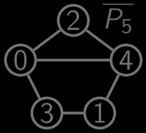
- if P is an extension path with one endpoint of degree 1

$\Rightarrow$ 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.

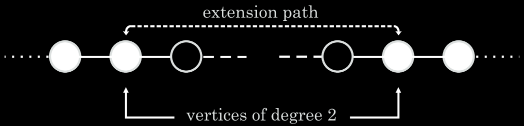
An algorithm for $L(2,1)$-labelings of span 4 (3/5)



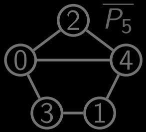
Rule 3 - Cheap Extensions

- 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

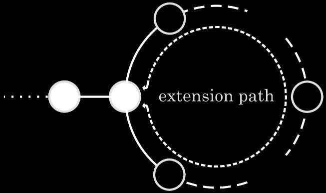


An algorithm for $L(2,1)$-labelings of span 4 (3/5)



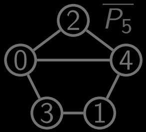
- if P is an extension path with identical endpoints

$\Rightarrow$ it is easy to decide whether P has a labeling compatible with its labeled endpoints



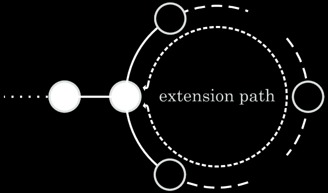
Remark : up to now, no branching was needed

An algorithm for $L(2,1)$-labelings of span 4 (3/5)



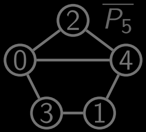
- if P is an extension path with identical endpoints

$\Rightarrow$ it is easy to decide whether P has a labeling compatible with its labeled endpoints



Remark : up to now, no branching was needed

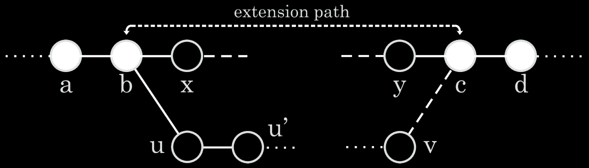
An algorithm for $L(2,1)$-labelings of span 4 (4/5)



Rule 4 - Extensions with Strong Constraints

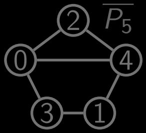
- if P is an extension path such that
  - both endpoints are labeled by 0 or 4
  - each endpoint has only one labeled neighbor
  - at least one endpoint has degree 3

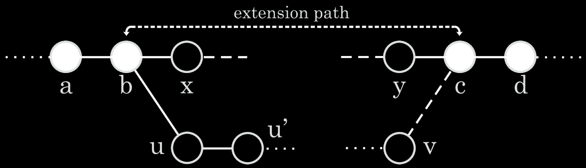
$\Rightarrow$ Branch along possible labelings of the (at most 4) unlabeled neighb of the endpoints + extend these labelings to entire path P.



By Rule 1-2, degrees of u and v (it it exists) are precisely 2.

An algorithm for $L(2,1)$-labelings of span 4 (4/5)





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

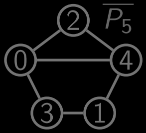
$$\mu(G) = \tilde{n} + \epsilon\hat{n}$$

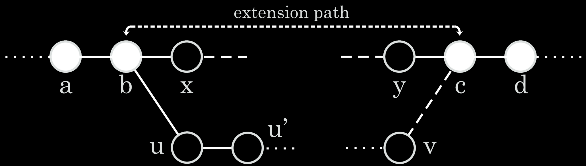
where

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

If 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 (4/5)





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

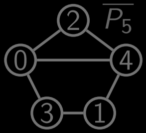
$$\mu(G) = \tilde{n} + \epsilon\hat{n}$$

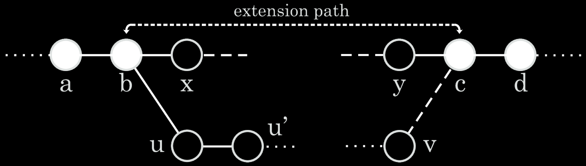
where

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

If 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 (4/5)





If length$(P) = 2$. Let $P = b, x, y, c$.

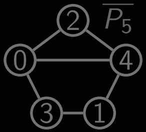
$\Rightarrow$ The possible labelings for *abxycd* are (up to symmetric labeling $f' = 4 - f$) :

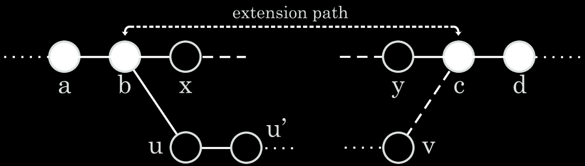
$40xy40 \to 403140$ $\quad$ $40xy42 \to 403142$ $\quad$ $40xy02 \to 402402$
$40xy03 \to 402403$ $\quad$ $20xy03 \to 204203$ $\quad$ $20xy04 \to 204204$
$20xy40 \to 203140$ $\quad$ $20xy42 \to 203142$ $\quad$ $30xy02 \to 302402$
$30xy04 \to 304204$ $\quad$ $30xy03 \to 302403, 304203.$

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 $d(c) = 3$.

An algorithm for $L(2,1)$-labelings of span 4 (4/5)





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

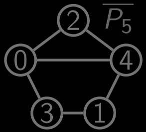
$\Rightarrow$ By doing the same analysis, we can establish that we have to branch in at most 2 subproblems.

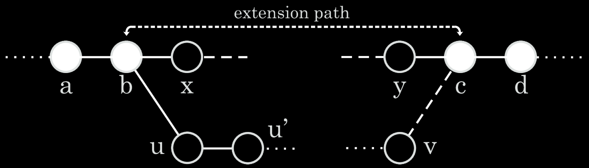
If length$(P) \geq 4$. Let $P = b, x, \ldots, y, c$.

$\Rightarrow$ 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 (4/5)





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

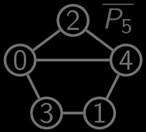
$\Rightarrow$ By doing the same analysis, we can establish that we have to branch in at most 2 subproblems.

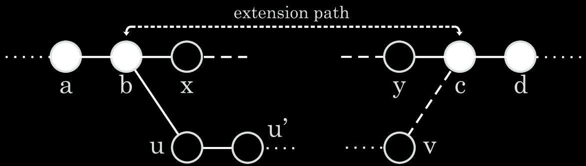
If length$(P) \geq 4$. Let $P = b, x, \ldots, y, c$.

$\Rightarrow$ 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 (4/5)





Consider the unlabeled neighbor u' of u.

if $w(u') = 1$. Labeling u would decrease $w(u')$ to ϵ.

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

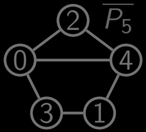
Due to Rule 1, u' has degree 2.

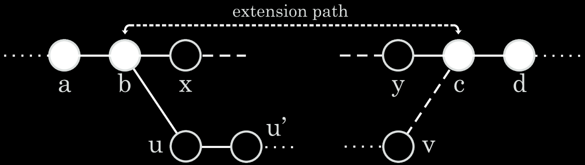
Labeling u would create an extension path $P' = uu'u''$ that can be labeled without branching by Rule 4.

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 (4/5)





Consider the unlabeled neighbor u' of u.

if $w(u') = 1$. Labeling u would decrease $w(u')$ to ϵ.

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

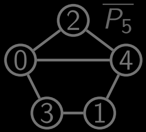
Due to Rule 1, u' has degree 2.

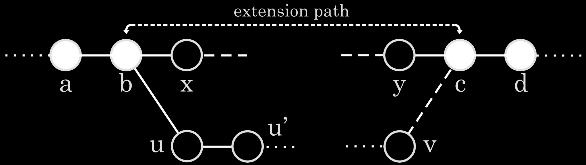
Labeling u would create an extension path $P' = uu'u''$ that can be labeled without branching by Rule 4.

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 (4/5)





Consider the unlabeled neighbor u' of u.

if $w(u') = 1$. Labeling u would decrease $w(u')$ to ϵ.

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

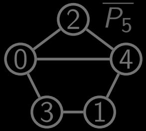
Due to Rule 1, u' has degree 2.

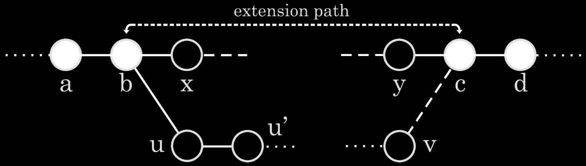
Labeling u would create an extension path $P' = uu'u''$ that can be labeled without branching by Rule 4.

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 (4/5)





Consider the unlabeled neighbor u' of u.

if $w(u') = 1$. Labeling u would decrease $w(u')$ to ϵ.

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

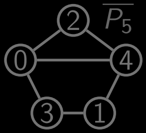
Due to Rule 1, u' has degree 2.

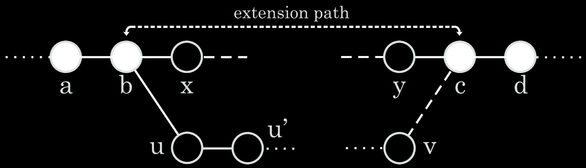
Labeling u would create an extension path $P' = uu'u''$ that can be labeled without branching by Rule 4.

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 (4/5)





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

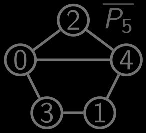
if v exists. $2\epsilon + (\text{length}(P) - 2) + 2\epsilon$

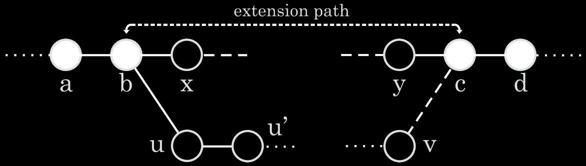
- 2ϵ for vertices x and y
- $\text{length}(P) - 2$ for the other vertices of P
- 2ϵ for vertices u and v

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

- 2ϵ for vertices x and y
- $\text{length}(P) - 2$ for the other vertices of P
- ϵ 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 (4/5)





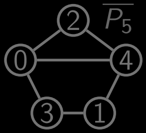
Putting it all together, labeling P, u and v would decrease the measure $\mu(G)$ by :

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

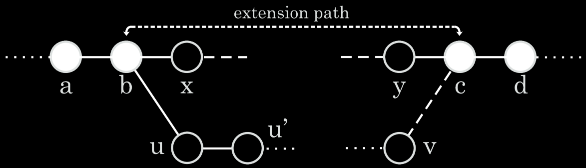
- 2ϵ for vertices x and y
- $\text{length}(P) - 2$ for the other vertices of P
- 2ϵ for vertices u and v

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

- 2ϵ for vertices x and y
- $\text{length}(P) - 2$ for the other vertices of P
- ϵ 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 (4/5)



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

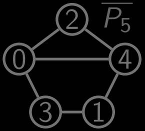
if v exists. $2\epsilon + (\text{length}(P) - 2) + 2\epsilon$

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

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

- 2ϵ for vertices x and y
- $\text{length}(P) - 2$ for the other vertices of P
- ϵ 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 (5/5)

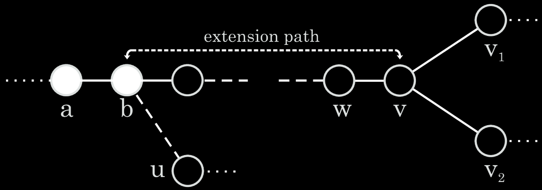


→ If none of Rules 1-4 can be applied, every unlabeled vertex adjacent to a labeled vertex belongs to an extension path with one unlabeled endpoint of degree 3.

Rule 5 - Extensions with Weak Constraints

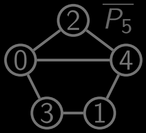
- if P is an extension path such that the unlabeled endpoint has degree 3

⇒ 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 ⇒ $w(v_1) = w(v_2) = 1$. And thus $u \neq v_1$ and $u \neq v_2$.

An algorithm for $L(2,1)$-labelings of span 4 (5/5)

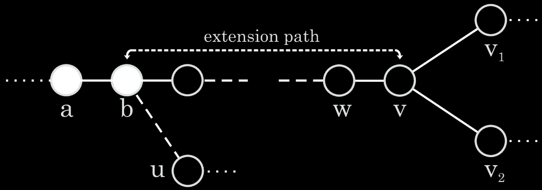


→ If none of Rules 1-4 can be applied, every unlabeled vertex adjacent to a labeled vertex belongs to an extension path with one unlabeled endpoint of degree 3.

Rule 5 - Extensions with Weak Constraints

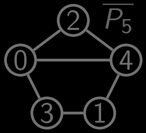
- if P is an extension path such that the unlabeled endpoint has degree 3

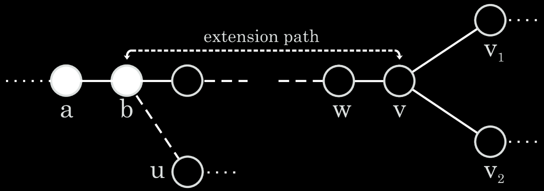
⇒ Branch along possible labelings of v , w and eventually u ✛ extend these labelings to entire P.



By Rule 1, neither v_1 nor v_2 are labeled or adjacent to a labeled vertex ⇒ $w(v_1) = w(v_2) = 1$. And thus $u \neq v_1$ and $u \neq v_2$.

An algorithm for $L(2,1)$-labelings of span 4 (5/5)





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

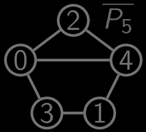
if u exists. $\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon + \epsilon$

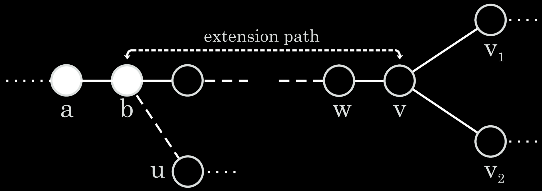
- ϵ for the first vertex of P
- $\text{length}(P) - 1$ for the other vertices of P
- $2 - 2\epsilon$ for vertices v_1 and v_2
- ϵ for vertex u

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

- ϵ for the first vertex of P
- $\text{length}(P) - 1$ for the other vertices of P
- $2 - 2\epsilon$ for vertices v_1 and v_2

An algorithm for $L(2,1)$-labelings of span 4 (5/5)



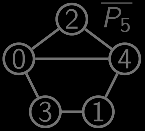


If length$(P) \leq 8$. Number of branchings :

length(P)	number of branchings if $\deg(b) = 2$	number of branchings if $\deg(b) = 3$
1	1	1
2	1	1
3	2	2
4	3	3
5	3	3
6	5	6
7	5	6
8	5	7

If 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 minimim span

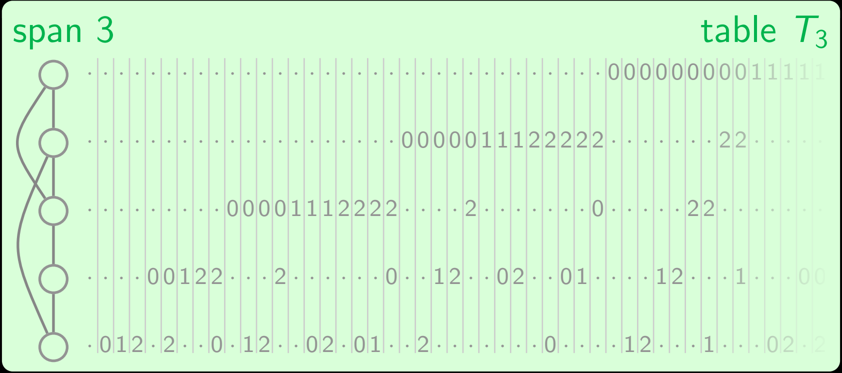
Dynamic programming for L(2,1)-labeling

Simple idea : fill-in table T_ℓ corresponding to partial labelings using up to ℓ labels.

span 3 table T_3

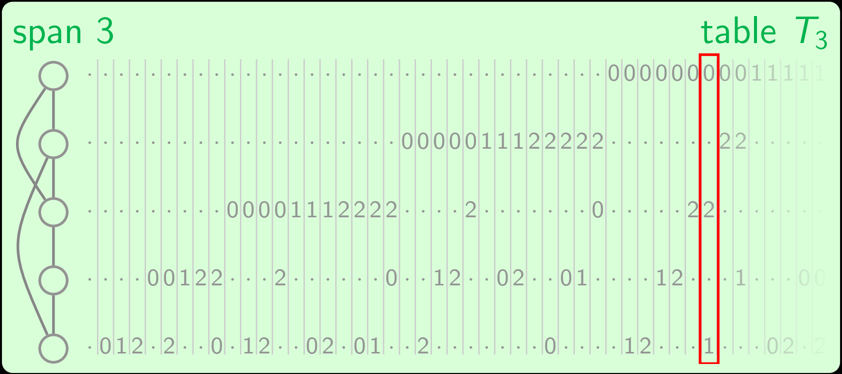
Dynamic programming for L(2,1)-labeling

Simple idea : fill-in table T_ℓ corresponding to partial labelings using up to ℓ labels.



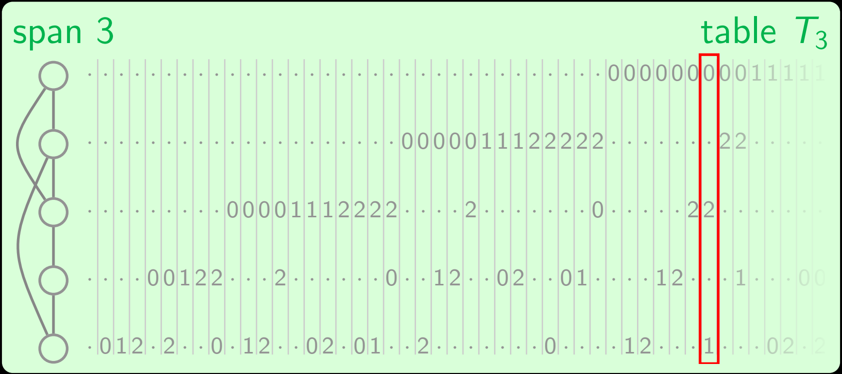
Dynamic programming for L(2,1)-labeling

Simple idea : fill-in table T_ℓ corresponding to partial labelings using up to ℓ labels.



Dynamic programming for L(2,1)-labeling

Simple idea : fill-in table T_ℓ corresponding to partial labelings using up to ℓ 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

Jump to a compact representation

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

- $a_i = 0$ iff v_i is not labeled by φ and there is no neighbor u of v_i with $\varphi(u) = \ell$
- $a_i = \overline{0}$ iff v_i is not labeled by φ 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

span 3 | table T_3

$0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,\bar{1}\,\bar{1}\,\bar{1}\,\bar{1}\,\bar{1}\,\bar{1}\,\bar{1}$

$0\,0\,0\,0\,0\,0\,0\,0\,\bar{0}\,\bar{0}\,\bar{0}\,1\,1\,1\,1\,\bar{0}\,\bar{0}\,\bar{0}\,1\,\bar{1}\,\bar{1}\,\bar{1}\,\bar{1}\,0\,0\,0\,0\,0\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{1}\,\bar{1}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,1\,1$

$0\,0\,0\,\bar{0}\,\bar{0}\,1\,1\,1\,0\,0\,1\,0\,0\,0\,\bar{0}\,\bar{1}\,\bar{1}\,\bar{1}\,\bar{1}\,\bar{0}\,\bar{0}\,\bar{0}\,1\,0\,0\,0\,\bar{0}\,\bar{0}\,0\,0\,\bar{1}\,\bar{1}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,\bar{0}\,1\,1\,\bar{0}\,\bar{0}$

$0\,0\,1\,\bar{1}\,\bar{1}\,0\,0\,\bar{1}\,\bar{0}\,1\,\bar{0}\,0\,\bar{0}\,1\,\bar{1}\,\bar{0}\,\bar{0}\,1\,\bar{0}\,0\,0\,1\,0\,0\,0\,1\,\bar{1}\,\bar{1}\,\bar{0}\,1\,\bar{0}\,\bar{0}\,0\,1\,0\,0\,1\,0\,0\,0\,1$

$0\,1\,0\,\bar{0}\,1\,0\,1\,\bar{0}\,\bar{1}\,\bar{1}\,\bar{1}\,0\,\bar{1}\,0\,\bar{0}\,0\,1\,0\,0\,\bar{0}\,1\,\bar{0}\,\bar{0}\,0\,1\,0\,\bar{0}\,1\,\bar{1}\,\bar{1}\,0\,1\,\bar{0}\,\bar{0}\,0\,1\,0\,0\,1\,0\,0$

Computing the tables

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

Let $P \subseteq \{0,1\}^n$ be the encodings of all 2-packings of G.

Formally, $\vec{p} \in P \Leftrightarrow \exists$ a 2-packing $S \subseteq V$ such that $\forall i$, $p_i = 1$ iff $v_i \in S$.

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

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

$\oplus$	0	$\bar{0}$	1	$\bar{1}$
0	0	0	1	1
1	$\bar{1}$	$\sim$	–	–

Entry "–" signifies that $\oplus$ is not defined.

Generalization of $\oplus$ to vectors :

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

Computing the tables

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

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

- $a_i = 0$ iff $a'_i = 0$ and there is no $v_j \in N(v_i)$ with $a'_j = \overline{1}$
- $a_i = \overline{0}$ iff $a'_i = 0$ and there is a $v_j \in N(v_i)$ with $a'_j = \overline{1}$
- $a_i = 1$ 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. $A_w = \{\vec{v} \mid w \cdot v \in A\}$

$\oplus$	0	$\bar{0}$	1	$\bar{1}$
0	0	0	1	1
1	$\bar{1}$	$\sim$	$-$	$-$

example :

$$\begin{array}{rl} & (0,\ 0,\ \bar{0},\ 1,\ \bar{1}) \\ \oplus & (0,\ 1,\ 0,\ 0,\ 0) \\ \hline & (0,\ \bar{1},\ 0,\ 1,\ 1) \end{array}$$

$$\begin{aligned} A \oplus B = \quad & 0((A_0 \cup A_{\bar{0}}) \oplus B_0) \\ \cup \quad & 1((A_1 \cup A_{\bar{1}}) \oplus B_0) \\ \cup \quad & \bar{1}(A_0 \oplus B_1) \end{aligned}$$

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

Computing the tables

$\oplus$	0	$\overline{0}$	1	$\overline{1}$
0	0	0	1	1
1	$\overline{1}$	$\sim$	$-$	$-$

two adjacent vertices

$\oplus$	00	$0\overline{0}$	01	$0\overline{1}$	$\overline{0}0$	$\overline{0}\overline{0}$	$\overline{0}1$	$\overline{0}\overline{1}$	10	$1\overline{0}$	11	$1\overline{1}$	$\overline{1}0$	$\overline{1}\overline{0}$	$\overline{1}1$	$\overline{1}\overline{1}$
00																
01																
10																
11																

Computing the tables

$\oplus$	0	$\bar{0}$	1	$\bar{1}$
0	0	0	1	1
1	$\bar{1}$	$\sim$	–	–

↓ two adjacent vertices ↓

$\oplus$	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	$1\bar{1}$	$\bar{1}0$	$\bar{1}\bar{0}$	$\bar{1}1$	$\bar{1}\bar{1}$
00																
01																
10																
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Computing the tables

$\oplus$	0	$\bar{0}$	1	$\bar{1}$
0	0	0	1	1
1	$\bar{1}$	$\sim$	–	–

↓ two adjacent vertices ↓

$\oplus$	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	$1\bar{1}$	$\bar{1}0$	$\bar{1}\bar{0}$	$\bar{1}1$	$\bar{1}\bar{1}$
00																
01			–	–			–	–			–	–			–	–
10									–	–	–	–	–	–	–	–
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Computing the tables

$\oplus$	0	$\bar{0}$	1	$\bar{1}$
0	0	0	1	1
1	$\bar{1}$	$\sim$	–	–

↓ two adjacent vertices ↓

$\oplus$	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	$1\bar{1}$	$\bar{1}0$	$\bar{1}\bar{0}$	$\bar{1}1$	$\bar{1}\bar{1}$
00																
01		$\sim$	–	–		$\sim$	–	–		$\sim$	–	–		$\sim$	–	–
10					$\sim$	$\sim$	$\sim$	$\sim$	–	–	–	–	–	–	–	–
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Computing the tables

$\oplus$	0	$\bar{0}$	1	$\bar{1}$
0	0	0	1	1
1	$\bar{1}$	$\sim$	$-$	$-$

↓ two adjacent vertices ↓

$\oplus$	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	$1\bar{1}$	$\bar{1}0$	$\bar{1}\bar{0}$	$\bar{1}1$	$\bar{1}\bar{1}$
00	00	00	01	01	00	00	01	01	10	10	11	11	10	10	11	-
01	$0\bar{1}$	$\sim$	$-$	$-$	$0\bar{1}$	$\sim$	$-$	$-$	$1\bar{1}$	$\sim$	$-$	$-$	$1\bar{1}$	$\sim$	$-$	$-$
10	$\bar{1}0$	$\bar{1}0$	$\bar{1}1$	$\bar{1}1$	$\sim$	$\sim$	$\sim$	$\sim$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Computing the tables

$\oplus$	0	$\overline{0}$	1	$\overline{1}$
0	0	0	1	1
1	$\overline{1}$	$\sim$	–	–

two adjacent vertices

$\oplus$	00	$0\overline{0}$	01	$0\overline{1}$	$\overline{0}0$	$\overline{0}\overline{0}$	$\overline{0}1$	$\overline{0}\overline{1}$	10	$1\overline{0}$	11	$1\overline{1}$	$\overline{1}0$	$\overline{1}\overline{0}$	$\overline{1}1$	$\overline{1}\overline{1}$
00	00	00	01	01	00	00	01	01	10	10	11	11	10	10	11	-
01	$0\overline{1}$	$\sim$	–	–	$0\overline{1}$	$\sim$	–	–	$1\overline{1}$	$\sim$	–	–	$1\overline{1}$	$\sim$	–	–
10	$\overline{1}0$	$\overline{1}0$	$\overline{1}1$	$\overline{1}1$	$\sim$	$\sim$	$\sim$	$\sim$	–	–	–	–	–	–	–	–
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

$\rightarrow$ Prefix $\overline{11}$ cannot appear.

Computing the tables

$\oplus$	00	$0\overline{0}$	01	$0\overline{1}$	$\overline{0}0$	$\overline{0}\overline{0}$	$\overline{0}1$	$\overline{0}\overline{1}$	10	$1\overline{0}$	11	$1\overline{1}$	$\overline{1}0$	$\overline{1}\overline{0}$	$\overline{1}1$	$\overline{1}\overline{1}$
00	00	00	01	01	00	00	01	01	10	10	11	11	10	10	11	-
01	$0\overline{1}$	$\sim$	–	–	$0\overline{1}$	$\sim$	–	–	$1\overline{1}$	$\sim$	–	–	$1\overline{1}$	$\sim$	–	–
10	$\overline{1}0$	$\overline{1}0$	$\overline{1}1$	$\overline{1}1$	$\sim$	$\sim$	$\sim$	$\sim$	–	–	–	–	–	–	–	–
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

$$A \oplus B =$$

Computing the tables

$\oplus$	00	$0\bar{0}$	01	$0\bar{1}$	$\bar{0}0$	$\overline{00}$	$\bar{0}1$	$\overline{01}$	10	$1\bar{0}$	11	$1\bar{1}$	$\bar{1}0$	$\overline{10}$	$\bar{1}1$	$\overline{11}$
00	00	00	01	01	00	00	01	01	10	10	11	11	10	10	11	-
01	$0\bar{1}$	$\sim$	–	–	$0\bar{1}$	$\sim$	–	–	$1\bar{1}$	$\sim$	–	–	$1\bar{1}$	$\sim$	–	–
10	$\bar{1}0$	$\bar{1}0$	$\bar{1}1$	$\bar{1}1$	$\sim$	$\sim$	$\sim$	$\sim$	–	–	–	–	–	–	–	–
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

$$A \oplus B = \quad 00((A_{00} \cup A_{0\bar{0}} \cup A_{\bar{0}0} \cup A_{\overline{00}}) \oplus B_{00})$$

Computing the tables

$\oplus$	00	$0\overline{0}$	01	$0\overline{1}$	$\overline{0}0$	$\overline{0}\overline{0}$	$\overline{0}1$	$\overline{0}\overline{1}$	10	$1\overline{0}$	11	$1\overline{1}$	$\overline{1}0$	$\overline{1}\overline{0}$	$\overline{1}1$	$\overline{1}\overline{1}$
00	00	00	01	01	00	00	01	01	10	10	11	11	10	10	11	-
01	$0\overline{1}$	$\sim$	–	–	$0\overline{1}$	$\sim$	–	–	$1\overline{1}$	$\sim$	–	–	$1\overline{1}$	$\sim$	–	–
10	$\overline{1}0$	$\overline{1}0$	$\overline{1}1$	$\overline{1}1$	$\sim$	$\sim$	$\sim$	$\sim$	–	–	–	–	–	–	–	–
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

$$
\begin{aligned}
A \oplus B = \quad & 00((A_{00} \cup A_{0\overline{0}} \cup A_{\overline{0}0} \cup A_{\overline{0}\overline{0}}) \oplus B_{00}) \\
\cup \quad & 01((A_{01} \cup A_{0\overline{1}} \cup A_{\overline{0}1} \cup A_{\overline{0}\overline{1}}) \oplus B_{00})
\end{aligned}
$$

Computing the tables

$\oplus$	00	$0\overline{0}$	01	$0\overline{1}$	$\overline{0}0$	$\overline{0}\overline{0}$	$\overline{0}1$	$\overline{0}\overline{1}$	10	$1\overline{0}$	11	$1\overline{1}$	$\overline{1}0$	$\overline{1}\overline{0}$	$\overline{1}1$	$\overline{1}\overline{1}$
00	00	00	01	01	00	00	01	01	10	10	11	11	10	10	11	-
01	$0\overline{1}$	$\sim$	–	–	$0\overline{1}$	$\sim$	–	–	$1\overline{1}$	$\sim$	–	–	$1\overline{1}$	$\sim$	–	–
10	$\overline{1}0$	$\overline{1}0$	$\overline{1}1$	$\overline{1}1$	$\sim$	$\sim$	$\sim$	$\sim$	–	–	–	–	–	–	–	–
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

$$
\begin{aligned}
A \oplus B = \quad & 00((A_{00} \cup A_{0\overline{0}} \cup A_{\overline{0}0} \cup A_{\overline{0}\overline{0}}) \oplus B_{00}) \\
\cup \quad & 01((A_{01} \cup A_{0\overline{1}} \cup A_{\overline{0}1} \cup A_{\overline{0}\overline{1}}) \oplus B_{00}) \\
\cup \quad & 10((A_{10} \cup A_{1\overline{0}} \cup A_{\overline{1}0} \cup A_{\overline{1}\overline{0}}) \oplus B_{00}) \\
\cup \quad & 11((A_{11} \cup A_{1\overline{1}} \cup A_{\overline{1}1}) \oplus B_{00}) \\
\cup \quad & 0\overline{1}((A_{00} \cup A_{\overline{0}0}) \oplus B_{01}) \\
\cup \quad & 1\overline{1}((A_{10} \cup A_{\overline{1}0}) \oplus B_{01}) \\
\cup \quad & \overline{1}0((A_{00} \cup A_{0\overline{0}}) \oplus B_{10}) \\
\cup \quad & \overline{1}1((A_{01} \cup A_{0\overline{1}}) \oplus B_{10})
\end{aligned}
$$

Computing the tables

$\oplus$	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	$1\bar{1}$	$\bar{1}0$	$\bar{1}\bar{0}$	$\bar{1}1$	$\bar{1}\bar{1}$
00	00	00	01	01	00	00	01	01	10	10	11	11	10	10	11	-
01	$0\bar{1}$	$\sim$	–	–	$0\bar{1}$	$\sim$	–	–	$1\bar{1}$	$\sim$	–	–	$1\bar{1}$	$\sim$	–	–
10	$\bar{1}0$	$\bar{1}0$	$\bar{1}1$	$\bar{1}1$	$\sim$	$\sim$	$\sim$	$\sim$	–	–	–	–	–	–	–	–
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

$$
\begin{aligned}
A \oplus B = \quad & 00((A_{00} \cup A_{0\bar{0}} \cup A_{\bar{0}0} \cup A_{\bar{0}\bar{0}}) \oplus B_{00}) \\
\cup \quad & 01((A_{01} \cup A_{0\bar{1}} \cup A_{\bar{0}1} \cup A_{\bar{0}\bar{1}}) \oplus B_{00}) \\
\cup \quad & 10((A_{10} \cup A_{1\bar{0}} \cup A_{\bar{1}0} \cup A_{\bar{1}\bar{0}}) \oplus B_{00}) \\
\cup \quad & 11((A_{11} \cup A_{1\bar{1}} \cup A_{\bar{1}1}) \oplus B_{00}) \\
\cup \quad & 0\bar{1}((A_{00} \cup A_{\bar{0}0}) \oplus B_{01}) \\
\cup \quad & 1\bar{1}((A_{10} \cup A_{\bar{1}0}) \oplus B_{01}) \\
\cup \quad & \bar{1}0((A_{00} \cup A_{0\bar{0}}) \oplus B_{10}) \\
\cup \quad & \bar{1}1((A_{01} \cup A_{0\bar{1}}) \oplus B_{10})
\end{aligned}
$$

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 $\mathcal{O}(2.6488^n)$.

We need further results :

- instead of considering 2 adjacent vertices, consider $k' = \mathcal{O}(1)$ 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

Additional Result 1

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 $n(1 + \frac{1}{k})$

▸ proof

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

Additional Result 2

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

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

Theorem. $2.6117^n \leq pp(n) \leq 2.6488^n$

▸ proof

An exact algorithm – Running-time analysis

Let $A \subseteq \{0, \bar{0}, 1, \bar{1}\}^n$ and $B \subseteq \{0, 1\}^n$ and $k' < n'$.

Compute $A \oplus B$ in the following way :

$$A \oplus B = \bigcup_{\substack{\vec{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \vec{v} \in \{0,1\}^{k'} \\ \text{s.t. } \vec{u} \oplus \vec{v} \text{ is defined}}} (\vec{u} \oplus \vec{v})(A_{\vec{u}} \oplus B_{\vec{v}})$$

$$= \bigcup_{\substack{\vec{v} \in \{0,1\}^{k'} \\ \vec{w} \in \{0,1,\bar{1}\}^{k'}}} \vec{w} \left[\left(\bigcup_{\substack{\vec{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \text{s.t. } \vec{u} \oplus \vec{v} = \vec{w}}} A_{\vec{u}} \right) \oplus B_{\vec{v}} \right]$$

Remark :

$\oplus$ computation can be omitted whenever $\left(\bigcup_{\substack{\vec{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \text{s.t. } \vec{u} \oplus \vec{v} = \vec{w}}} A_{\vec{u}} \right)$ 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 $v_i = 1 \Rightarrow w_i = \overline{1}$.

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

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

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

▸ more details

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 $v_i = 1 \Rightarrow w_i = \overline{1}$.

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 $n - k'$.

▸ more details

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 $v_i = 1 \Rightarrow w_i = \overline{1}$.

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 $n - k'$.

▸ more details

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 $v_i = 1 \Rightarrow w_i = \overline{1}$.

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 $n - k'$.

more details

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 $v_i = 1 \Rightarrow w_i = \overline{1}$.

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 $n - k'$.

▸ more details

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

Conclusion

Conclusion

Short summary.

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

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

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

Conclusion

Short summary.

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

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

- count span 4 : $\mathcal{O}(1.1269^n)$ (exp-space) [CGKLP,2013]
- enumerate span 5 in cubic graphs : $\mathcal{O}(1.7990^n)$ [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 [1]]

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

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 [1]]

Merci !

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

1. first recipient of the Turing price, 1966

Decomposing the graph into connected subgraphs (proof)

◂ back

Proof **1/2**

- Consider a DFS-tree T of G rooted at r.
- For every v let $T(v)$ be the subtree rooted in v.
- If $|T(r)| \leq 2k$ then add G to the set of desired subgraphs and stop.
- 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)

back

Proof **2/2**

- 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 $\{u_1, \dots, u_i\}$ 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 $G \setminus (V(T(u_1)) \cup .. \cup V(T(u_i)))$.

- 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. □

2-Packings and Proper Pairs (proof)

back

Proof. 1/2

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.

$\Rightarrow$ we can assume G to be a tree.

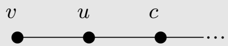
Fact 2. Suppose that there are two leaves which have a common neighbor. Every 2-packing in G is also a 2-packing in H.

$\Rightarrow$ we can assume that there are no two or more leaves with a common neighbor

2-Packings and Proper Pairs (proof)

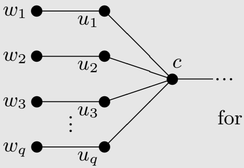
Proof. ◀ back

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

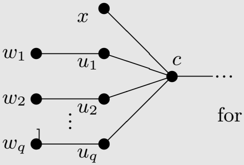


$$pp(n) \leq 2\, pp(n-1) + 4\, pp(n-3)$$

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



for $q \geq 2$



for $q \geq 1$

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

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

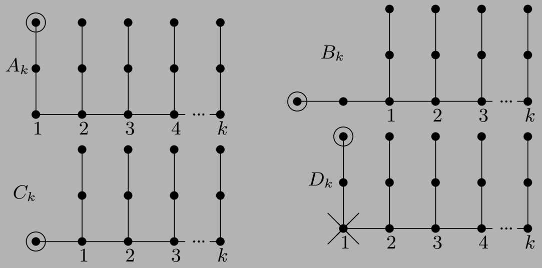
(B1) one neighbor of c is a leaf ...

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

2-Packings and Proper Pairs (proof)

back

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



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

Theorem. $2.6117^n \le pp(n) \le 2.6488^n$

An exact algorithm – Running-time analysis

back

By Theorem ($\star$), 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 $\bar{1}$'s form a 2-packing and there are only two possibilities (1 or $0/\bar{0}$) 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 $k \leq k' \leq 2k$.

An exact algorithm – Running-time analysis

◂ back

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 :

$$T(n) = O(2.6488^n)$$