List Colouring Trees in Logspace

Hans Bodlaender, Carla Groenland and Hugo Jacob
Utrecht University, Utrecht University and ENS Paris-Saclay

Dutch Optimization Seminar

Complexity of LIST COLOURING

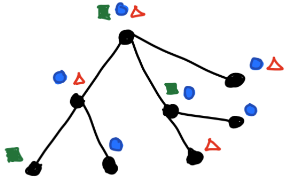
Given. Graph $G = (V, E)$ on n vertices and
a list $L(v) \subseteq \{1, \ldots, n\}$ of colours for each $v \in V$.

Output. Is there a proper vertex colouring c
with $c(v) \in L(v)$ for all $v \in V$?

Complexity of List Colouring

Given. Graph $G = (V, E)$ on n vertices and
a list $L(v) \subseteq \{1, \ldots, n\}$ of colours for each $v \in V$.

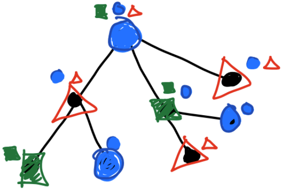
Output. Is there a proper vertex colouring c
with $c(v) \in L(v)$ for all $v \in V$?



Complexity of List Colouring

Given. Graph $G = (V, E)$ on n vertices and
a list $L(v) \subseteq \{1, \ldots, n\}$ of colours for each $v \in V$.

Output. Is there a proper vertex colouring c
with $c(v) \in L(v)$ for all $v \in V$?



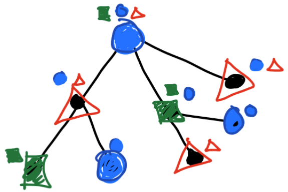
Complexity of LIST COLOURING

Given. Graph $G = (V, E)$ on n vertices and
a list $L(v) \subseteq \{1, \ldots, n\}$ of colours for each $v \in V$.

Output. Is there a proper vertex colouring c
with $c(v) \in L(v)$ for all $v \in V$?

Solvable in linear time on trees.

NP-c for planar bipartite graphs or cographs.



Logspace model

Main result. List Colouring on trees is in L.

- Deterministic Turing machine.
- Input tape (read-only): contains lists and n-vertex tree.
- Space usage: $O(\log n)$ bits on the work tape.

Logspace model

Main result. List Colouring on trees is in L.

- Deterministic Turing machine.
- Input tape (read-only): contains lists and n-vertex tree.
- Space usage: $O(\log n)$ bits on the work tape.

Reingold (2008): undirected vertex connectivity is in L.

Elberfeld, Jakoby and Tantau (2010): Logspace version of Bodlaender's and Courcelle's theorem.

$\implies$ Done if bounded list size.

Talk overview

Main result. List Colouring on trees is in L.

The remainder of the talk:

- Ideas for $O(\log^2 n)$ algorithm.
- Required improvements for $O(\log n)$.
- Relation to larger project in parameterized complexity.

Notation and first ideas

T = input tree.

L = list of colours.

n = number of vertices.

$d(v)$ = degree of v.

- We may set $C \log n$ bits apart.

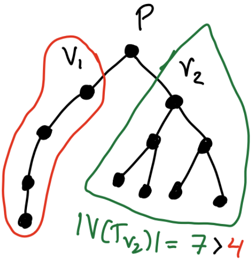
  $\implies$ Can **recompute** relevant logspace computable quantities when needed.

- Only need to try the first $d(v) + 1$ colours from $L(v)$

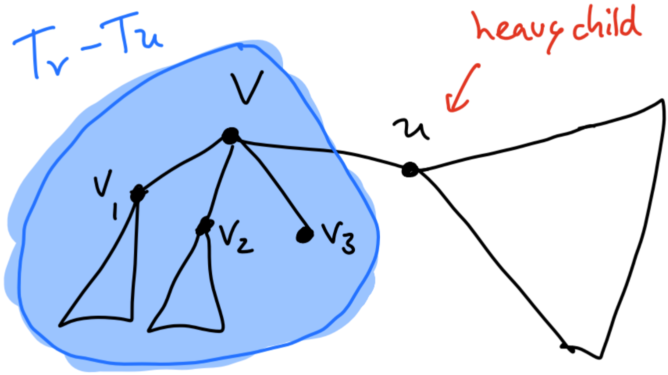
  $\implies$ $O(\log d(v))$ bits for storing **position in list** of colour.

Heavy/light decomposition

- Root the tree (arbitrary but deterministic).
- T_v subtree rooted in v.
- Child v of p is **heavy** if child with largest $|V(T_v)|$.
- Otherwise v is **light** and $|V(T_v)| \leqslant \frac{1}{2}|V(T_p)|$.



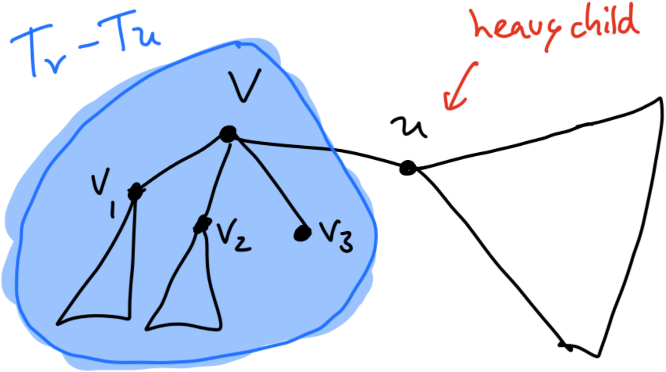
Critical versus non-critical



No way to colour $T_v - T_u$?

$\Longrightarrow$ Return fail.

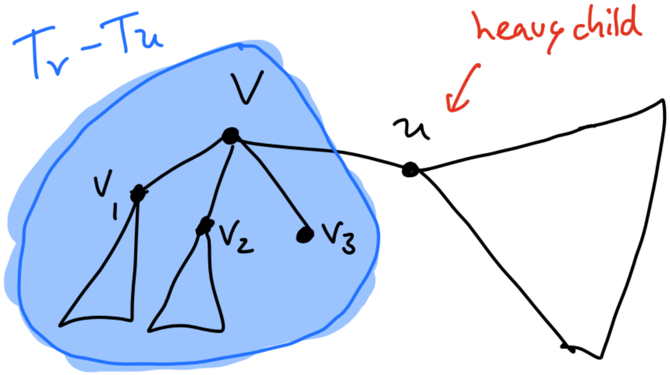
Critical versus non-critical



Non-critical: v can get two colours in $T_v - T_u$.

$\implies$ Continue to u without constraints.

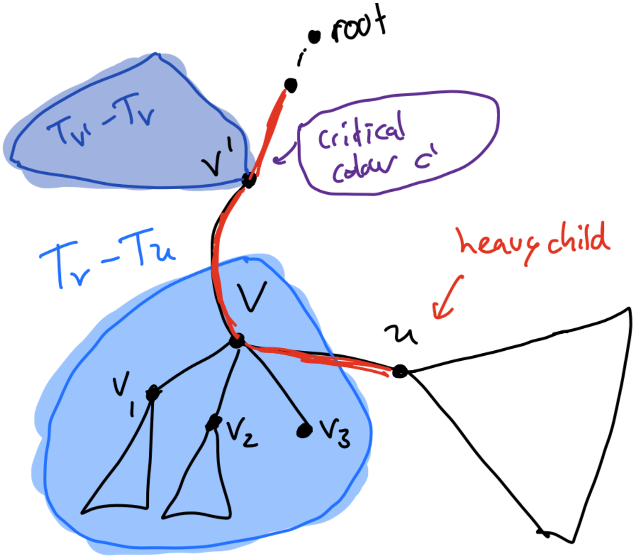
Critical versus non-critical



Critical: v can only get colour c in $T_v - T_u$.

$\implies$ Continue to u while remembering v needs c.

Critical versus non-critical



$O(\log^2 n)$ algorithm time analysis

For vertex v with heavy child u, we check which colours v can get in $T_v - T_u$.

$\implies$ Recursive calls on light children only.

$\implies$ Recursion depth: $O(\log n)$.

$O(\log^2 n)$ algorithm time analysis

For vertex v with heavy child u, we check which colours v can get in $T_v - T_u$.

$\implies$ Recursive calls on light children only.

$\implies$ Recursion depth: $O(\log n)$.

May forget parent v' of v when move to heavy child u of v.

$\implies$ $O(\log n)$ bits per recursion level.

$\implies$ $O(\log^2 n)$ total.

How to reach $O(\log n)$?

Suppose we do a 'recursive call' on light child w of v.

Key idea. *Space allocated for parent v depends on 'size reduction'.*

- $O(1)$ bits if $|V(T_w)| = |V(T_v)|/2$.
- $O(\log n)$ bits if $|V(T_w)| = \sqrt{|V(T_v)|}$.

Algorithm processes small subtrees first.

How to reach $O(\log n)$?

Suppose we do a 'recursive call' on light child w of v.

Key idea. *Space allocated for parent v depends on 'size reduction'.*

- $O(1)$ bits if $|V(T_w)| = |V(T_v)|/2$.
- $O(\log n)$ bits if $|V(T_w)| = \sqrt{|V(T_v)|}$.

Algorithm processes small subtrees first.

Large subtree $\implies$ few children left $\implies$ small 'effective degree'

$\implies$ cheaper description of colour available.

Imaginary lists

G_j induced on v and subtrees of children w with $|V(T_w)| \leqslant n/2^{2^j}$.

Imaginary lists

G_j induced on v and subtrees of children w with $|V(T_w)| \leqslant n/2^{2^j}$.

$$L_j(v) = \{c \in L(v) : G_j \text{ admits list colouring } \alpha \text{ with } \alpha(v) = c\}.$$

Imaginary lists

G_j induced on v and subtrees of children w with $|V(T_w)| \leqslant n/2^{2^j}$.

$$L_j(v) = \{c \in L(v) : G_j \text{ admits list colouring } \alpha \text{ with } \alpha(v) = c\}.$$

Store colour via position in $L_j(v)$: takes $O(\log |L_j(v)|)$ bits.

Imaginary lists

G_j induced on v and subtrees of children w with $|V(T_w)| \leqslant n/2^{2^j}$.

$$L_j(v) = \{c \in L(v) : G_j \text{ admits list colouring } \alpha \text{ with } \alpha(v) = c\}.$$

Store colour via position in $L_j(v)$: takes $O(\log |L_j(v)|)$ bits.

At most 2^{2^j} children w of v are not in G_j (volume argument).

$\implies$ either v non-critical or $|L_j(v)| \leqslant 2^{2^j} + 2$.

$\implies$ use $O(2^j)$ bits for position.

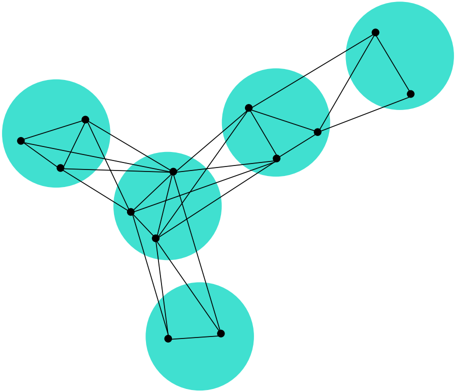
Overview of algorithm

Main result. List Colouring on trees is in L.

- Recurse only on light children, starting with small subtrees.
- Store positions instead of colour; recompute colour only when needed.
- Technical detail: need to group children into brackets based on subtree size, for $M = \Theta(\log \log n)$:

$$[1, n/2^{2^{M-1}}), \ldots, [n/2^{2^{j+1}}, n/2^{2^j}), \ldots, [n/4, n/2).$$

List Colouring on tree-like graphs?



Algorithm gives $O(f(k) \log n)$ space for n-vertex graphs of tree-partition-width k.

Primer on parameterized complexity

INDEPENDENT SET.
Given n-vertex graph, does it have independent set of size k?

Usual complexity: running time in terms of n.

Parameterized complexity: separate out influence of parameter (e.g. k).

Primer on parameterized complexity

Independent Set.
Given n-vertex graph, does it have independent set of size k?

Usual complexity: running time in terms of n.

Parameterized complexity: separate out influence of parameter (e.g. k).

Para NP-hard: NP-hard for constant value of parameter.

Primer on parameterized complexity

INDEPENDENT SET.
Given n-vertex graph, does it have independent set of size k?

Usual complexity: running time in terms of n.

Parameterized complexity: separate out influence of parameter (e.g. k).

Para NP-hard: NP-hard for constant value of parameter.

XP: $n^{f(k)}$ Try all subsets: $\binom{n}{k} = O(n^k)$.

Primer on parameterized complexity

INDEPENDENT SET.
Given n-vertex graph, does it have independent set of size k?

Usual complexity: running time in terms of n.

Parameterized complexity: separate out influence of parameter (e.g. k).

Para NP-hard: NP-hard for constant value of parameter.

XP: $n^{f(k)}$ Try all subsets: $\binom{n}{k} = O(n^k)$.

FPT: $f(k)n^{O(1)}$ Not possible?

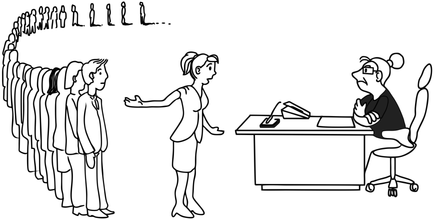
Proving hardness interesting?

"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, but neither can all these famous people."

Proving hardness interesting?

“I can’t find an efficient algorithm, I guess I’m just too dumb.”



“I can’t find an efficient algorithm, but neither can all these famous people.”

C-hard: ‘at least as hard’ as all problems in C.

All C-complete problems are ‘similarly hard’.

Primer on parameterized complexity

Downey & Fellows (1999):

- W[1] $\subseteq$ W[2] $\subseteq$ W[3] $\subseteq \ldots\ldots$
- W[1]: class for INDEPENDENT SET.
- W[2]: class for DOMINATING SET.

Primer on parameterized complexity

Downey & Fellows (1999):

- W[1] $\subseteq$ W[2] $\subseteq$ W[3] $\subseteq \ldots\ldots$
- W[1]: class for INDEPENDENT SET.
- W[2]: class for DOMINATING SET.

Recent joint work in team headed by Hans Bodlaender:

- XNLP: class for LIST COLOURING parameterized by pathwidth.
- XALP: class for LIST COLOURING parameterized by treewidth.

Primer on parameterized complexity

Downey & Fellows (1999):

- W[1] $\subseteq$ W[2] $\subseteq$ W[3] $\subseteq \ldots$.
- W[1]: class for INDEPENDENT SET.
- W[2]: class for DOMINATING SET.

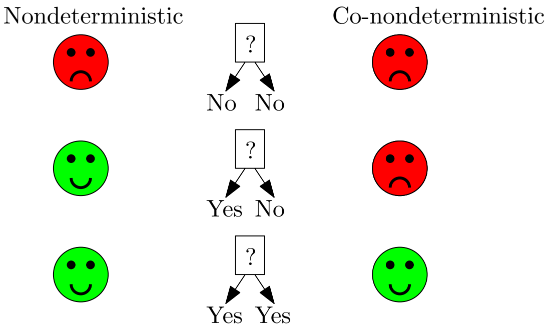
Recent joint work in team headed by Hans Bodlaender:

- XNLP: class for LIST COLOURING parameterized by pathwidth.
- XALP: class for LIST COLOURING parameterized by treewidth.

XALP-hard $\implies$ XNLP-hard $\implies$ W[t]-hard for all t.

Natural 'home' for path/tree-structured problems.

Nondeterminism versus co-nondeterminism



Alternating Turing machine admits both nondeterminism and co-nondeterminism.

Machine models and a conjecture

- X = slice-wise, parameterized problem (n, k)
- N = nondeterministic Turing machine
- A = alternating Turing machine*
- L = logspace $f(k) \log n$
- P = fpt time $f(k)n^{O(1)}$

Machine models and a conjecture

- X = slice-wise, parameterized problem (n, k)
- N = nondeterministic Turing machine
- A = alternating Turing machine*
- L = logspace $f(k) \log n$
- P = fpt time $f(k)n^{O(1)}$

Conjecture (Pilipczuk, Wrochna). *XNLP-hard problems admit no deterministic $n^{f(k)}$ time $f(k)n^{O(1)}$ space algorithm.*

$\implies$ For some constant k_0, there is no $O(\log n)$ space algorithm for list colouring graphs of pathwidth at most k_0.

XNLP/XALP-completeness recipe

Membership:

- Use machine model definition.
- Deterministic dynamic programming $\rightarrow$ nondeterministic ‘guess’ table entries, conondeterminism to handle ‘branching’ in tree.

XNLP/XALP-completeness recipe

Membership:

- Use machine model definition.
- Deterministic dynamic programming $\rightarrow$ nondeterministic 'guess' table entries, conondeterminism to handle 'branching' in tree.

Completeness:

- Reduce from known complete problems.
- pl-reduction: $O(\log n) + f(k)$ space, do not blow up parameter.

First complete problem (Cook-style): Binary CSP (think: List Colouring with arbitrary constraints).

Credits

First XNLP-completeness result:

- M. Elberfeld, C. Stockhusen, and T. Tantau. *On the space and circuit complexity of parameterized problems: Classes and completeness*, 2015.

XALP paper builds on classical analogues (SAC, NAuxPDA, ASPSZ):

- E. Allender, S. Chen, T. Lou, P. A. Papakonstantinou, and B. Tang. *Width-parametrized SAT: time-space tradeoffs*, 2015.
- W. L. Ruzzo. *Tree-size bounded alternation*, 1980.

Parameter	Problem
Pathwidth	List Colouring*, All-or-Nothing Flow*, Capacitated Dominating Set
Linear clique-width	Chromatic Number, Maximum Regular Induced Subgraph*, Max Cut*
Pathwidth / log n	q-Coloring, Dominating Set*, Independent Set*, Odd Cycle Transversal
Linear mim-width	Independent Set, Dominating Set, Feedback Vertex Set
Bandwidth	Bandwidth*
Number of sequences	Longest Common Subsequence

Table: Overview of XNLP-completeness results

*: XALP-complete when replacing

pathwidth → treewidth
linear cliquewidth → cliquewidth
bandwidth → tree-partition width.

Future directions

Space complexity of LIST COLOURING:

$O(\log n)$ deterministic for treewidth 1. *Q: treewidth 2?*

Future directions

Space complexity of List Colouring:

$O(\log n)$ deterministic for treewidth 1. *Q: treewidth 2?*

$O(\log n)$ nondeterministic for constant pathwidth.

$O(\log^2 n)$ nondeterministic for constant treewidth. *Q: $o(\log^2 n)$?*

Future directions

Space complexity of LIST COLOURING:

$O(\log n)$ deterministic for treewidth 1. *Q: treewidth 2?*

$O(\log n)$ nondeterministic for constant pathwidth.

$O(\log^2 n)$ nondeterministic for constant treewidth. *Q:* $o(\log^2 n)$*?*

Q: Space efficiency of other problems?

Future directions

Space complexity of List Colouring:

$O(\log n)$ deterministic for treewidth 1. *Q: treewidth 2?*

$O(\log n)$ nondeterministic for constant pathwidth.

$O(\log^2 n)$ nondeterministic for constant treewidth. *Q:* $o(\log^2 n)$*?*

Q: Space efficiency of other problems?

Q: Problem with tree/path-like structure? $\rightarrow$ *XNLP/XALP-complete?*

Future directions

Space complexity of List Colouring:

$O(\log n)$ deterministic for treewidth 1. *Q: treewidth 2?*

$O(\log n)$ nondeterministic for constant pathwidth.

$O(\log^2 n)$ nondeterministic for constant treewidth. *Q:* $o(\log^2 n)$*?*

Q: Space efficiency of other problems?

Q: Problem with tree/path-like structure? $\rightarrow$ *XNLP/XALP-complete?*

Thank you for your attention!

XNLP/XALP definitions

XNLP = parameterized problems (n, k) solvable in nondeterministic $f(k)n^{O(1)}$ time and $f(k) \log n$ space.

XALP = XNLP with auxiliary stack
= parameterized problems (n, k) solvable by Alternating Turing Machine in $f(k)n^{O(1)}$ time and $f(k) \log n$ space plus one of following:
$O(\log n)$ co-nondeterministic steps per branch.
Computation tree has size $f(k)n^{O(1)}$.)

Conjecture (Pilipczuk, Wrochna). *XNLP-hard problems admit no deterministic $n^{f(k)}$ time $f(k)n^{O(1)}$ space algorithm.*

‘Folklore algorithm’

n = number of vertices.
$d(v)$ = degree of v.
Δ = maximum degree.

- Can ‘construct’ nice path decomposition of width $w = O(\log n)$.
  Recompute relevant parts whenever needed.
- Only need to try the first $d(v) + 1$ colours for v
  $\implies O(\log \Delta)$ bits for storing **position in list** of colour.
- Nondeterministic ‘guess’ colours for vertices in current bag, check compatible with previous bag. At most two bags in memory.
  $\implies O(w \log \Delta) = O(\log^2 n)$ bits.

Recursion

$$C\log(n/2) = C\log n - 2C \implies O(1) \text{ bits if } |V(T_w)| = n/2.$$
$$C\log(\sqrt{n}) = C\log n - C\log n/2 \implies O(\log n) \text{ bits if } |V(T_w)| = \sqrt{n}.$$

Brackets

For $M = \Theta(\log \log n)$, we consider brackets:

$$[1, n/2^{2^{M-1}}), \dots, [n/2^{2^{j+1}}, n/2^{2^j}), \dots, [n/4, n/2).$$

These are used to group together children by the size of their subtrees.

When computing position in $L_j(v)$, may need to store information about positions in $L_i(v)$ for $i \leqslant j$, but this sums nicely:

$$\sum_{i=1}^{j} 2^i \leqslant 2 \cdot 2^j.$$