Developing Heuristic Algorithms for Graph Optimization Problems using Tree Decompositions

Louis Carpentier
Promotor: Jan Goedgebeur
Mentor: Jorik Jooken

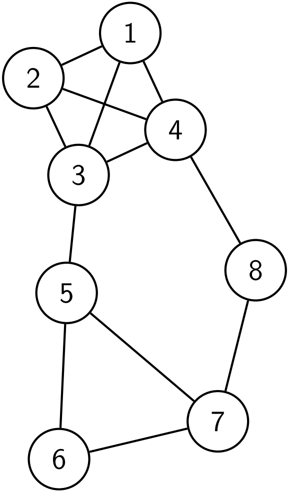
KU Leuven
December, 2021

Overview

Overview

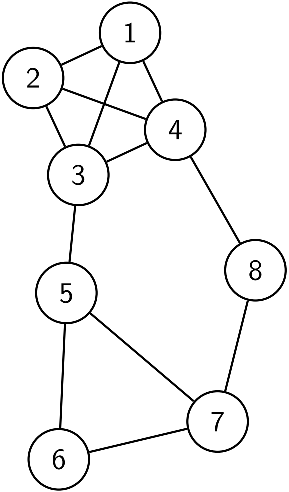
Graph

- A graph $G = (V, E)$
  - vertices $V(G)$: 1, 2, 3, ...
  - edges $E(G)$: $\{1, 2\}$, $\{3, 5\}$, $\{7, 8\}$, ...



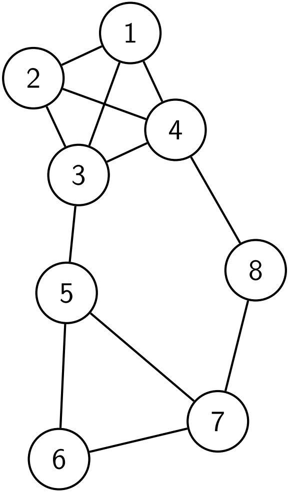
Graph

- A graph $G = (V, E)$
  - vertices $V(G)$: 1, 2, 3, ...
  - edges $E(G)$: $\{1, 2\}$, $\{3, 5\}$, $\{7, 8\}$, ...
- For all $v \in V(G)$, $deg(v)$ is the degree of vertex v
  - $deg(3) = 4$, $deg(8) = 2$



Graph

- A graph $G = (V, E)$
  - vertices $V(G)$: 1, 2, 3, ...
  - edges $E(G)$: $\{1, 2\}$, $\{3, 5\}$, $\{7, 8\}$, ...
- For all $v \in V(G)$, $deg(v)$ is the degree of vertex v
  - $deg(3) = 4$, $deg(8) = 2$
- For all $v \in V(G)$, $N(v)$ is the set of vertices u such that $uv \in E(G)$
  - $N(3) = \{1, 2, 4, 5\}$, $N(8) = \{4, 7\}$



KU LEUVEN

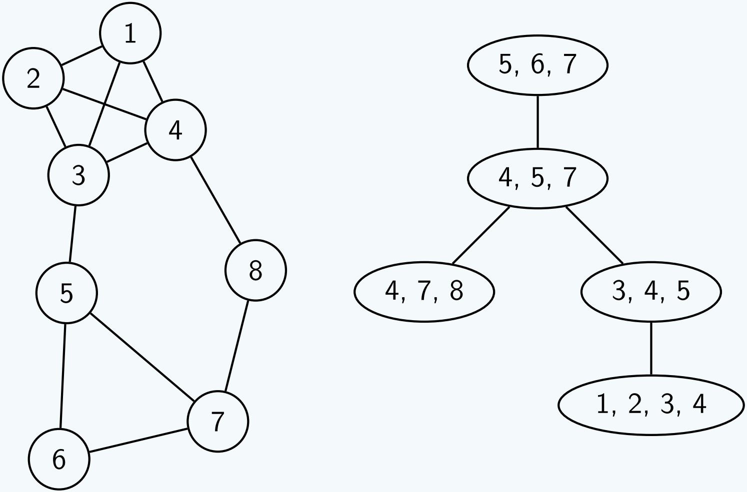
Tree Decomposition

Definition (Tree decomposition Cygan et al. 2015)

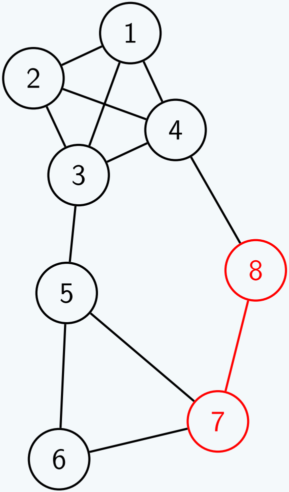
A *tree decomposition* of graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a bag, such that the following three conditions hold:

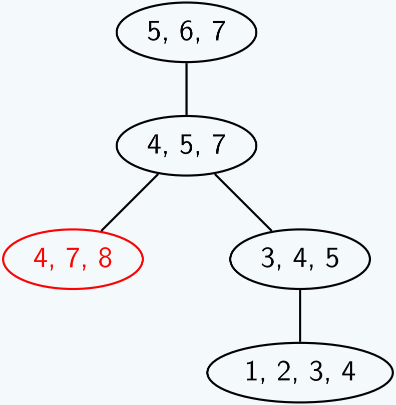
1. $\bigcup_{t\in V(T)} X_t = V(G)$: every vertex $v \in V(G)$ is contained in some bag X_t for $t \in V(T)$.
2. For every $\{u, v\} \in E(G)$, there exists a node t of $V(T)$ such that bag X_t contains both u and v.
3. For every $v \in V(G)$, the set $T_v = \{t \in V(T) : v \in X_t\}$ induces a connected subtree of T.

Example (Property 1: $\bigcup_{t \in V(T)} X_t = V(G)$: every vertex $v \in V(G)$ is contained in some bag X_t for $t \in V(T)$)

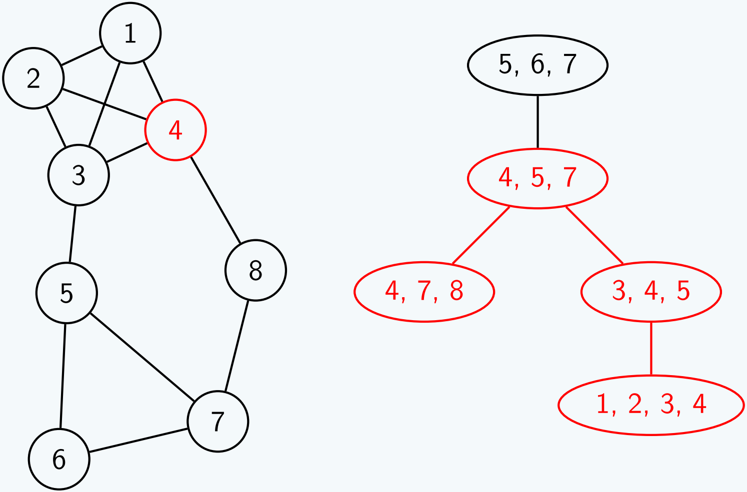


Example (Property 2: For every $\{u, v\} \in E(G)$, there exists a node t of $V(T)$ such that bag X_t contains both u and v)





Example (Property 3: For every $v \in V(G)$, the set $T_v = \{t \in V(T) : v \in X_t\}$ induces a connected subtree of T)



Treewidth

Definition (Width of a tree decomposition)

The *width of a tree decomposition* $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ equals $max_{t \in V(T)} |X_t| - 1$.

Definition (Treewidth of a graph)

The *treewidth* of a graph G – denoted by $t = tw(G)$ – is the minimum possible width of a tree decomposition of G.

The Treewidth Problem

Theorem (Arnborg, Corneil, and Proskurowski 1987)

Given a graph G and integer t, deciding if G has a treewidth of at most t is $\mathcal{NP}$-complete.

The Treewidth Problem

Theorem (Arnborg, Corneil, and Proskurowski 1987)

Given a graph G and integer t, deciding if G has a treewidth of at most t is $\mathcal{NP}$-complete.

- Many algorithms exist for computing a tree decomposition exist
  - Exponential time exact algorithms
  - Polynomial time approximation algorithms
  - Heuristics with local search
  - See Bodlaender 2005 for an extensive overview

The Treewidth Problem

Theorem (Arnborg, Corneil, and Proskurowski 1987)

Given a graph G and integer t, deciding if G has a treewidth of at most t is $\mathcal{NP}$-complete.

- Many algorithms exist for computing a tree decomposition exist
  - Exponential time exact algorithms
  - Polynomial time approximation algorithms
  - Heuristics with local search
  - See Bodlaender 2005 for an extensive overview
- PACE 2017: Dell et al. 2018
  - Construct an algorithm to compute tree decompositions
  - See for example Tamaki 2019; Strasser 2017; Bannach and Berndt 2019

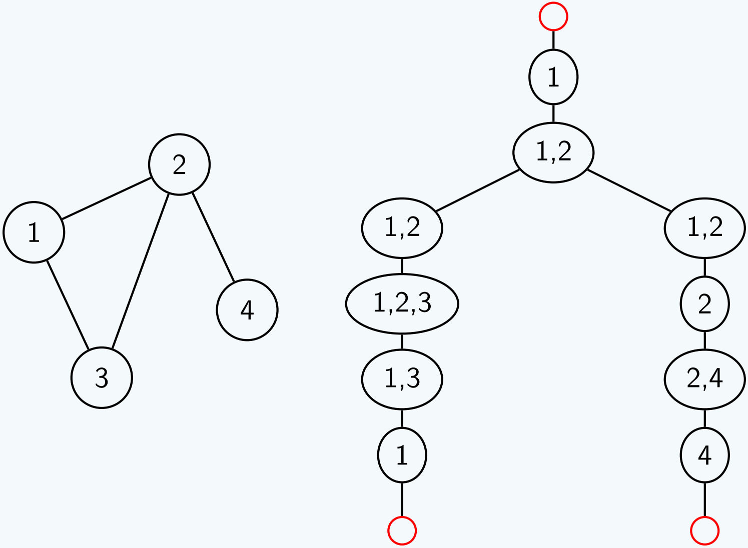
Nice Tree Decomposition

Definition (Nice Tree Decomposition)

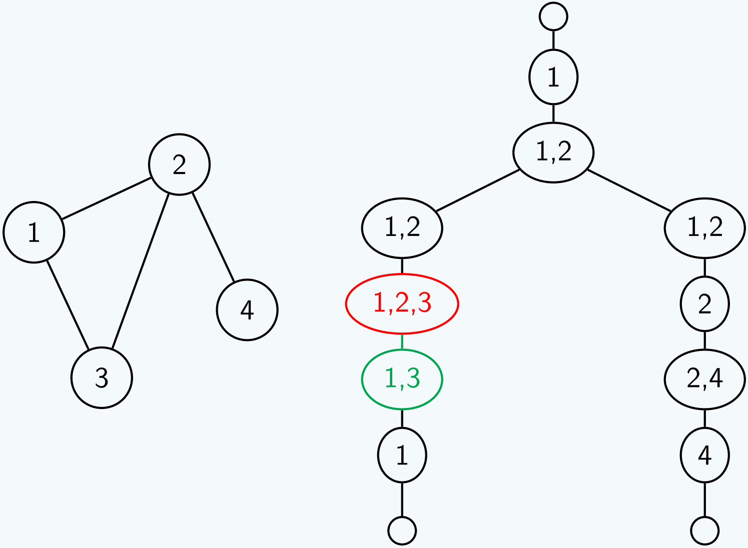
A rooted tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of graph G is *nice* if the following conditions hold:

- $X_r = \emptyset$ and $X_l = \emptyset$ for every leaf l of $V(T)$.
- Every non-leaf node of T is one of the following types:
  1. Introduce node: a node t with exactly 1 child t' such that $X_t = X_{t'} \cap \{v\}$ for some vertex $v \notin X_{t'}$.
  2. Forget node: a node t with exactly 1 child t' such that $X_t = X_{t'} \setminus \{v\}$ for some vertex $v \in X_{t'}$.
  3. Join node: a node t with exactly two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$

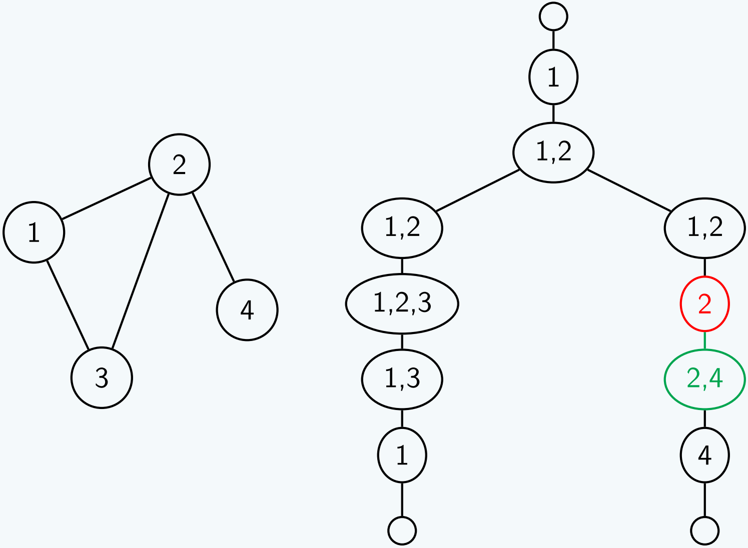
Example (Root and leaves)



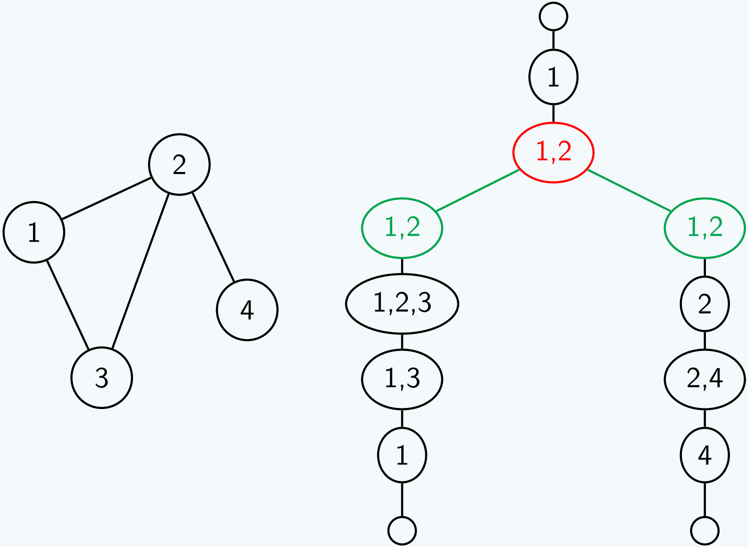
Example (Introduce node)



Example (Forget node)



Example (Join node)



Dynamic Programming using Nice Tree Decompositions

- Define cost $\tau[t, X]$ at every node of $t \in V(T)$
  - X depends on the problem

Dynamic Programming using Nice Tree Decompositions

- Define cost $\tau[t, X]$ at every node of $t \in V(T)$
  - X depends on the problem
- Compute $\tau[t, X]$ in a bottom-up fashion
  - Bags can only relate in a few simple ways to the bags of the children in a nice tree decomposition

Dynamic Programming using Nice Tree Decompositions

- Define cost $\tau[t, X]$ at every node of $t \in V(T)$
  - X depends on the problem
- Compute $\tau[t, X]$ in a bottom-up fashion
  - Bags can only relate in a few simple ways to the bags of the children in a nice tree decomposition
- Final evaluation is $\tau[r, \emptyset]$ for root r of T

Dynamic Programming using Nice Tree Decompositions

- Define cost $\tau[t, X]$ at every node of $t \in V(T)$
  - X depends on the problem
- Compute $\tau[t, X]$ in a bottom-up fashion
  - Bags can only relate in a few simple ways to the bags of the children in a nice tree decomposition
- Final evaluation is $\tau[r, \emptyset]$ for root r of T
- Results in time complexity of $\mathcal{O}(c^{\mathcal{O}(tw(G))} \cdot n^{\mathcal{O}(1)})$

Dynamic Programming using Nice Tree Decompositions

- Define cost $\tau[t, X]$ at every node of $t \in V(T)$
  - X depends on the problem
- Compute $\tau[t, X]$ in a bottom-up fashion
  - Bags can only relate in a few simple ways to the bags of the children in a nice tree decomposition
- Final evaluation is $\tau[r, \emptyset]$ for root r of T

- Results in time complexity of $\mathcal{O}(c^{\mathcal{O}(tw(G))} \cdot n^{\mathcal{O}(1)})$
- What if the constraint of an exact solution is dropped in order to reduce the exponential term in $tw(G)$?

Dynamic Programming using Nice Tree Decompositions

- Define cost $\tau[t, X]$ at every node of $t \in V(T)$
  - X depends on the problem
- Compute $\tau[t, X]$ in a bottom-up fashion
  - Bags can only relate in a few simple ways to the bags of the children in a nice tree decomposition
- Final evaluation is $\tau[r, \emptyset]$ for root r of T

- Results in time complexity of $\mathcal{O}(c^{\mathcal{O}(tw(G))} \cdot n^{\mathcal{O}(1)})$
- What if the constraint of an exact solution is dropped in order to reduce the exponential term in $tw(G)$?

➡ Goal of my thesis

Overview

Maximum Happy Vertices Problem (Zhang and Li 2015)

Definition (Happy and Unhappy Vertices)

Given a graph G and a colouring $c : V(G) \to \{1 \ldots k\}$. A vertex $v \in V(G)$ is said to be *happy* if and only if $c(v) = c(v')$ for all vertices $v' \in N(v)$, otherwise vertex v is said to be *unhappy*.

Maximum Happy Vertices Problem (Zhang and Li 2015)

Definition (Happy and Unhappy Vertices)

Given a graph G and a colouring $c : V(G) \to \{1 \dots k\}$. A vertex $v \in V(G)$ is said to be *happy* if and only if $c(v) = c(v')$ for all vertices $v' \in N(v)$, otherwise vertex v is said to be *unhappy*.

Definition (Maximum Happy Vertices Problem – MHV)

Given a graph G and a partial colouring $c : V(G) \to \{1 \dots k\}$. Extend the partial colouring c to a full colouring c' such that the number of happy vertices is maximized.

Maximum Happy Vertices Problem (Zhang and Li 2015)

Definition (Happy and Unhappy Vertices)

Given a graph G and a colouring $c : V(G) \to \{1 \dots k\}$. A vertex $v \in V(G)$ is said to be *happy* if and only if $c(v) = c(v')$ for all vertices $v' \in N(v)$, otherwise vertex v is said to be *unhappy*.

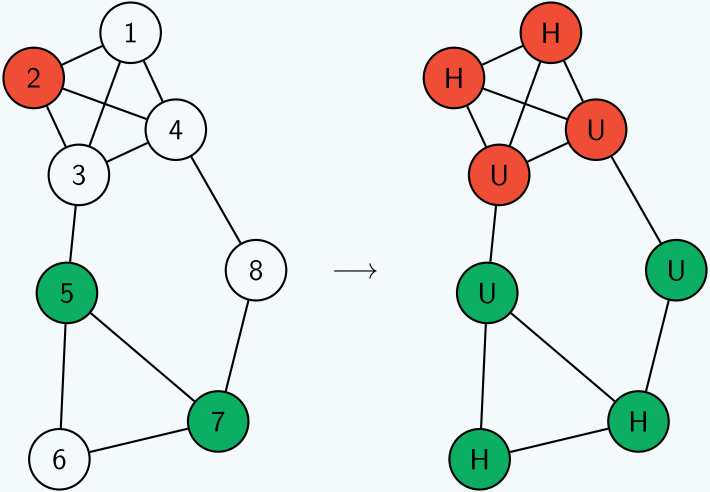
Definition (Maximum Happy Vertices Problem – MHV)

Given a graph G and a partial colouring $c : V(G) \to \{1 \dots k\}$. Extend the partial colouring c to a full colouring c' such that the number of happy vertices is maximized.

Definition (k-MHV)

An instance of the MHV problem, in which the number of colours k is a constant.

Example



Applications of Maximum Happy Vertices

1. Modeling social networks: *homophyly*
   - People tend to be similar to their friends
   - Elements in a small community share remarkable common features
   - The feature values are represented by a colour
   - See Easley, Kleinberg, et al. 2010; Li and Peng 2011; Li and Peng 2012 for more information

Applications of Maximum Happy Vertices

1. Modeling social networks: *homophyly*
   - People tend to be similar to their friends
   - Elements in a small community share remarkable common features
   - The feature values are represented by a colour
   - See Easley, Kleinberg, et al. 2010; Li and Peng 2011; Li and Peng 2012 for more information
2. Clustering
   - The vertices represent data points
   - The edges represent strong connections between the data points
   - The colours represent the different clusters

Complexity of the Maximum Happy Vertices Problem

Theorem (Zhang and Li 2015)

The k-MHV problem is $\mathcal{NP}$-complete for every constant $k \geq 3$.

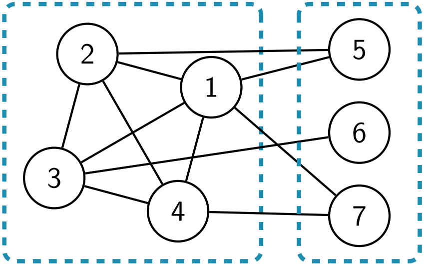
Complexity of the Maximum Happy Vertices Problem

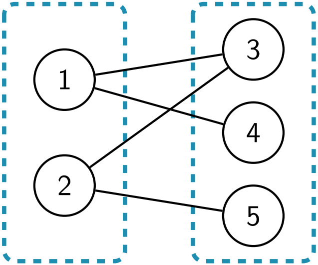
Theorem (Zhang and Li 2015)

The k-MHV problem is $\mathcal{NP}$-complete for every constant $k \geq 3$.

Theorem (Aravind et al. 2017)

The k-MHV problem is $\mathcal{NP}$-complete for split graphs and bipartite graphs for every constant $k \geq 3$.





Complexity of the Maximum Happy Vertices Problem

Theorem (Zhang and Li 2015)

The k-MHV problem is $\mathcal{NP}$-complete for every constant $k \geq 3$.

Theorem (Aravind et al. 2017)

The k-MHV problem is $\mathcal{NP}$-complete for split graphs and bipartite graphs for every constant $k \geq 3$.

Theorem (Agrawal 2017)

The MHV problem when parametrized by the number of happy vertices is $\mathcal{W}[1]$-hard.

Overview

Heuristic Approach

- What causes the exponential term in the running time?

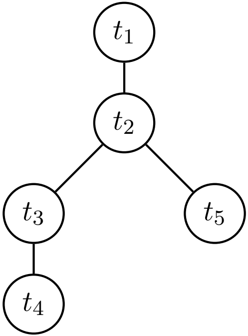
Heuristic Approach

- What causes the exponential term in the running time?
  - Due to the size of the dynamic programming table

Heuristic Approach

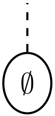
- What causes the exponential term in the running time?
  - Due to the size of the dynamic programming table

- Compute a fixed number of colour functions at each node
- Colour functions at node t contain information about $X_{t'}$ for all descendants t' of t
  - Descendants of t_3: $\{t_4\}$
  - Descendants of t_2: $\{t_3, t_4, t_5\}$



Leaf Nodes

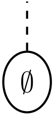
- Leaf nodes have empty bags
  - ⇒ Pass initial colour function to parent



Leaf Nodes

- Leaf nodes have empty bags

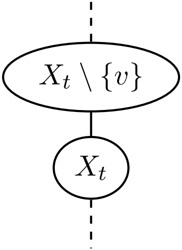
  Pass initial colour function to parent



Forget Nodes

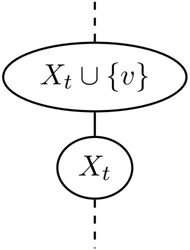
- Vertex v is 'forgotten' from the tree decomposition
- Sub tree contains the same vertices

  Pass colour functions of child to parent



Introduce Nodes

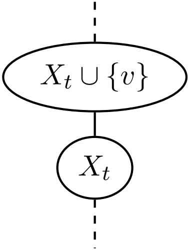
- New information introduced: vertex v
  - v must receive a colour



Introduce Nodes

- New information introduced: vertex v
  - v must receive a colour

Two approaches:

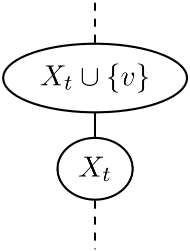


Introduce Nodes

- New information introduced: vertex v
  - v must receive a colour

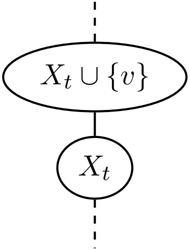
Two approaches:

1. For each colour function of child, give v the colour that results in the most happy vertices



Introduce Nodes

- New information introduced: vertex v
  - v must receive a colour

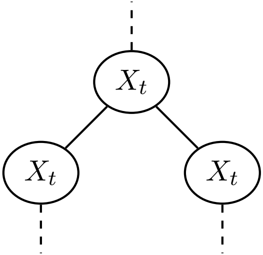


Two approaches:

1. For each colour function of child, give v the colour that results in the most happy vertices
2. For each colour function of child, give v all the possible colours and select the best colour functions

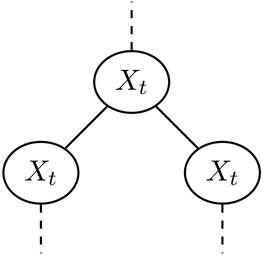
Join Nodes

- The left and right children pass different colour functions to the join node
- Merge all colour functions c_l of the left child with all colour functions c_r of the right child
- Iterate over all vertices $v \in X_t$ and assign it a colour



Join Nodes

- The left and right children pass different colour functions to the join node
- Merge all colour functions c_l of the left child with all colour functions c_r of the right child
- Iterate over all vertices $v \in X_t$ and assign it a colour



Order to iterate over the vertices in X_t?

1. Static: random, greatest degree first, smallest degree first
2. Dynamic: vertex connected to most/fewest already coloured vertices, vertex with fewest differently coloured neighbours

Evaluation method

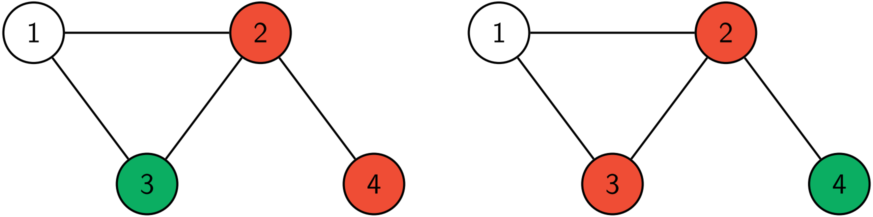
How to decide what the best colour function is?

1. Count the number of happy vertices

Evaluation method

How to decide what the best colour function is?

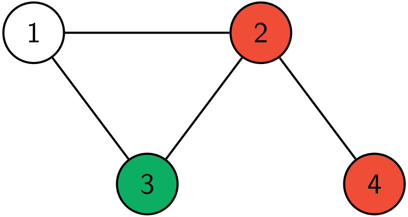
1. Count the number of happy vertices



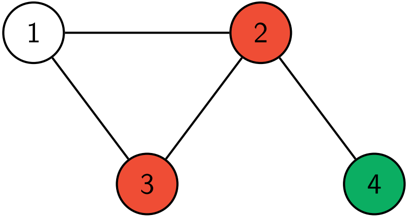
Evaluation method

How to decide what the best colour function is?

1 Count the number of happy vertices



vertex 4 is happy, but the other vertices can never be happy

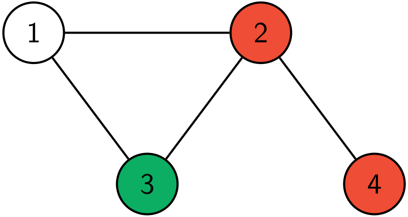


No vertex is happy, but both vertex 1 and 3 are happy if vertex 1 is coloured red

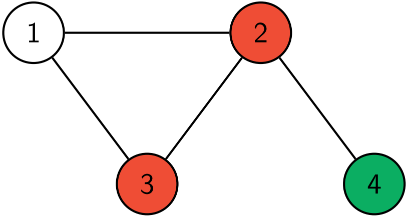
Evaluation method

How to decide what the best colour function is?

1. Count the number of happy vertices
2. Count the number of happy and potentially happy vertices



vertex 4 is happy, but the other vertices can never be happy



No vertex is happy, but both vertex 1 and 3 are happy if vertex 1 is coloured red

Overview

Future Work

Algorithm improvement:

- Implement more techniques to handle the different types of nodes
- Apply kernelization on the input graph H. Gao and W. Gao 2018

Future Work

Algorithm improvement:

- Implement more techniques to handle the different types of nodes
- Apply kernelization on the input graph H. Gao and W. Gao 2018

Experimental analysis:

- Compare algorithm performance against other, existing heuristic algorithms
- Compare the influence on both solution quality and running time for different tree decompositions
- Compare the retrieved colouring functions of the heuristic approach and exact algorithm at each node t of the tree decomposition
- Apply *framework* on other problems

Thank you for your attention!

Bibliography I

Agrawal, Akanksha (2017). "On the parameterized complexity of happy vertex coloring". In: *International Workshop on Combinatorial Algorithms*. Springer, pp. 103–115.

Aravind, NR et al. (2017). "Algorithms and hardness results for happy coloring problems". In: *arXiv preprint arXiv:1705.08282*.

Arnborg, Stefan, Derek G Corneil, and Andrzej Proskurowski (1987). "Complexity of finding embeddings in a k-tree". In: *SIAM Journal on Algebraic Discrete Methods* 8.2, pp. 277–284.

Bannach, Max and Sebastian Berndt (2019). "Practical access to dynamic programming on tree decompositions". In: *Algorithms* 12.8, p. 172.

Bodlaender, Hans L (1992). *A tourist guide through treewidth*. Vol. 92. Unknown Publisher.

Bibliography II

- Bodlaender, Hans L (2005). "Discovering treewidth". In: *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, pp. 1–16.
- Cygan, Marek et al. (2015). *Parameterized algorithms*. Vol. 5. 4. Springer.
- Dell, Holger et al. (2018). "The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration". In: *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- Easley, David, Jon Kleinberg, et al. (2010). *Networks, crowds, and markets*. Vol. 8. Cambridge university press Cambridge.
- Gao, Hang and Wenyu Gao (2018). "Kernelization for maximum happy vertices problem". In: *Latin American Symposium on Theoretical Informatics*. Springer, pp. 504–514.

Bibliography III

- Li, Angsheng and Pan Peng (2011). "Community structures in classical network models". In: *Internet Mathematics* 7.2, pp. 81–106.
- — (2012). "The small-community phenomenon in networks". In: *Mathematical Structures in Computer Science* 22.3, pp. 373–407.
- Marchal, Lambertus (2012). "Treewidth: structural properties and algorithmic insights". In.
- Strasser, Ben (2017). "Computing tree decompositions with flowcutter: PACE 2017 submission". In: *arXiv preprint arXiv:1709.08949*.
- Tamaki, Hisao (2019). "Positive-instance driven dynamic programming for treewidth". In: *Journal of Combinatorial Optimization* 37.4, pp. 1283–1311.

Bibliography IV

Zhang, Peng and Angsheng Li (2015). "Algorithmic aspects of homophyly of networks". In: *Theoretical Computer Science* 593, pp. 117–131.

KU LEUVEN