Optimal tree decompositions revisited: A simpler linear-time FPT algorithm

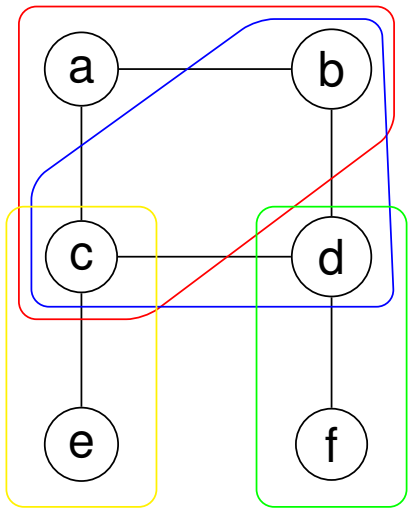
Ernst Althaus and Sarah Ziegler

The Main Result

- Bodlaender (1996): A linear-time algorithm for finding tree-decompositions of small treewidth
- This paper was cited more than 1.500 times
- The algorithm was declared as “to complicated for textbooks” several times
- We tried to simplify its description

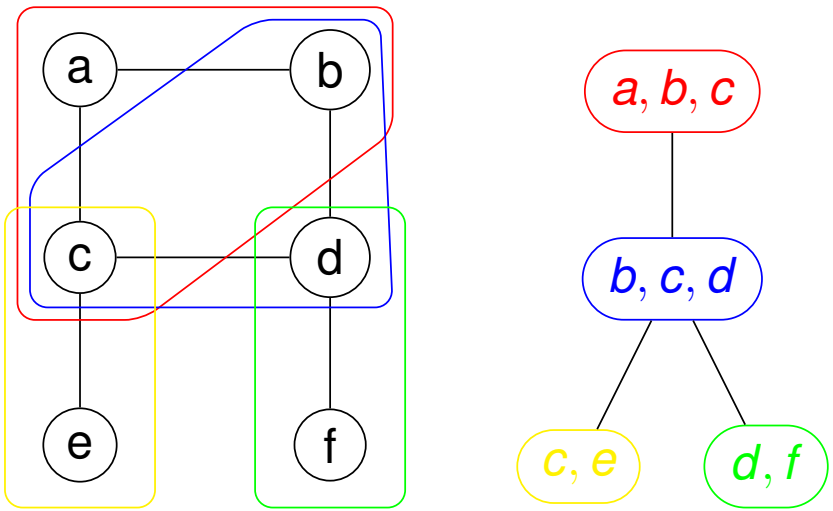
- **Motivation:** Many NP-hard problems are fixed-parameter tractable in the treewidth (defined next), i.e. allow an algorithm with a running time of $f(tw) \cdot \mathrm{poly}(|\mathrm{V}(\mathrm{G})|)$ if *tw* is the treewidth for some function *f*

Tree-Decompositions: Intuition



- Cover the graph (vertices and edges) with subsets of the vertices (called bags)
- . . . such that these subset are “tree like” (defined on the next slide)
- To simplify a bit, we assume $G = (V(G), E(G))$ to be connected

Tree-Decompositions: Definition



Graph Tree-Decomposition

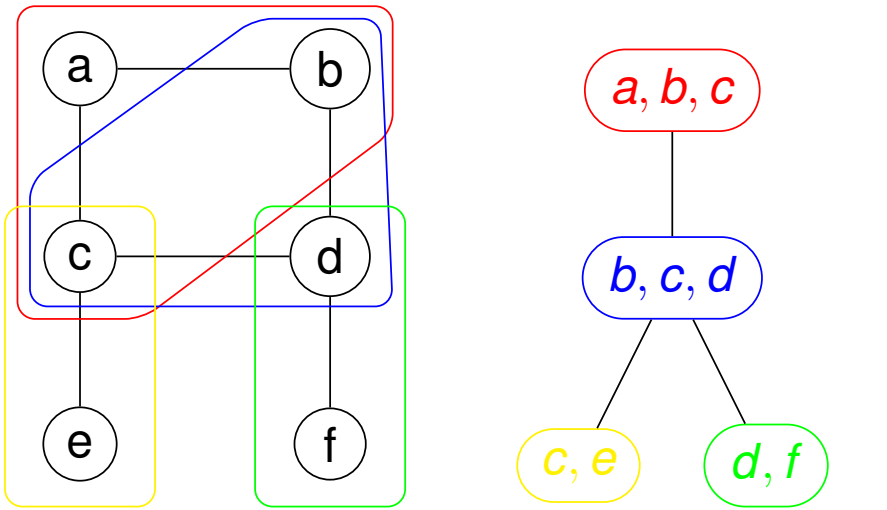
Tree-Decomposition (TD) for a graph $G = (V(G), E(G))$:

$(T, (X_t)_{v \in V(T)})$ for tree T with nodes $V(T)$ and $X_t \subseteq V$ with

Coverage: For each $uv \in E(G)$ there is $t \in V(T)$ with $u, v \in X_t$

Coherence: If $u \in X_{t'} \cap X_{t''}$ then $u \in X_t$ for all t on the path in T between t' and t''

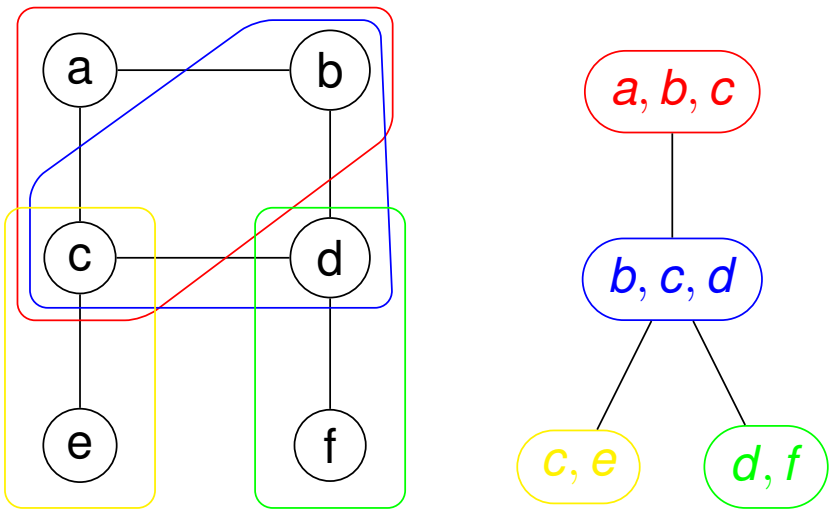
Tree-Decompositions: Names



Graph Tree-Decomposition

- The sets X_t are called bags
- Elements $V(T)$ are called nodes
- . . . whereas elements $V(G)$ are called vertices

Treewidth: Definition



Graph Tree-Decomposition

Width of a TD $(T, (X_t)_{v \in V(T)})$: Maximal size of a bag minus 1,
i.e. $\max_{t \in V(T)} |X_t| - 1$

Treewidth of a graph G: Minimal width of any TD of G

Fact: There is an optimal TD with at most $|V(G)|$ nodes

Nice Tree-Decompositions

- The tree is rooted, the bag of each leaf is $\emptyset$ and we have three kinds of non-leaf nodes:

  Introduce: t has exactly one child t' and

  $$X_t = X_{t'} \cup \{v\} \text{ for a vertex } v \in V(G) \setminus X_{t'}$$

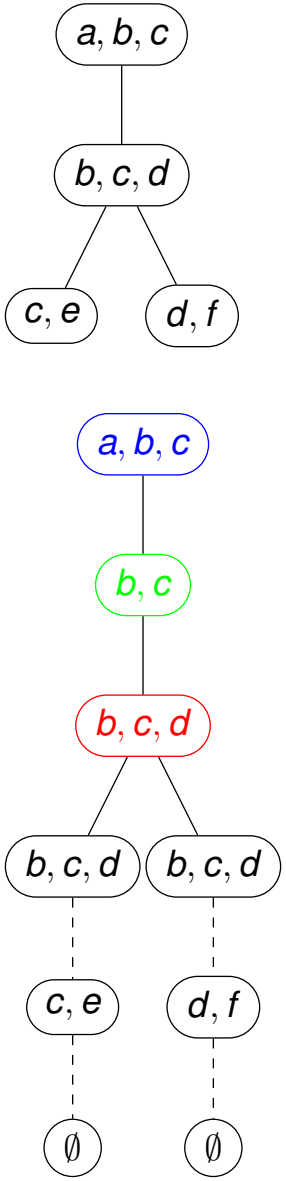
  Forget: t has exactly one child t' and

  $$X_t = X_{t'} \setminus \{v\} \text{ for a vertex } v \in X_{t'}$$

  Join: t has exactly two children t_1 and t_2 and

  $$X_t = X_{t_1} = X_{t_2}$$

- Each TD can be transformed to a nice one without increasing the width



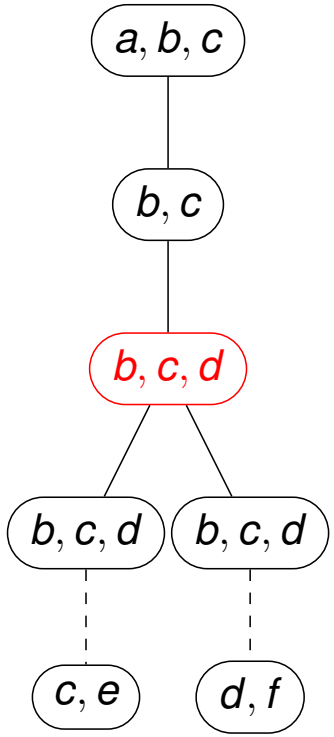
Current, Forgotten and (Un)seen Vertices

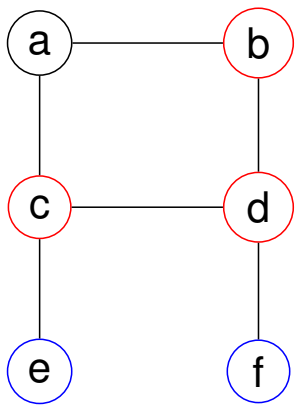
- Algorithms typically traverse the tree T bottom-up
- In node t, we call vertices

  Current: If they are in X_t (marked red)

  Seen: If they appear in some bag below t (unseen are marked black)

  Forgotten: If they are seen, but not current (marked blue)
- In a join-node, exactly the current nodes are seen in both subtrees





Typical Approach for TD-based-Algorithms

- For node $t \in V(T)$, enumerate all solutions restricted to the seen vertices (to be defined for the given problem)
- At the root, we have all solutions
- Show how to construct solutions for leaves and for join-, introduce-, and forget-nodes from one or two solutions of the children
- Let $J^t(s_1, s_2)$, $I^t(s)$, $F^t(s)$ be the solutions that can be constructed in join-, introduce-, and forget-nodes, respectively
- As there typically are exponentially many, we build equivalence classes and store only one solution per class (see next slide)

Building equivalence classes: necessary information

- For each solution s, only store the "necessary information" $NI(s)$:
  - We call s and s' with $NI(s) = NI(s')$ equivalent, denoted $s \equiv s'$
  - If $s \in J^t(s_1, s_2)$ can be constructed in bag t, $s'_1 \equiv s_1$ and $s'_2 \equiv s_2$ then there is $s' \in J^t(s'_1, s'_2)$ with $s \equiv s'$
  - $NI(s')$ can be computed from $NI(s'_1)$ and $NI(s'_2)$

  $\rightarrow$ modify J^T accordingly
  - Similarly for I^t, F^t and the leaves
- Intuitively: We only need to store the necessary information as
  - we can compute necessary informations of solutions bottom up
  - For each (complete) solution, at least one equivalent will be computed

Linear-time Algorithm for the treewidth

Today:

- Given a (not necessarily optimal) nice TD of width $\hat{tw}$ for $G = (V(G), E(G))$
- Enumerate all equivalence classes of (not necessarily nice) TDs with at most $|V(G)|$ nodes of width at most $\hat{tw}$
- ... in time $f(\hat{tw}) \cdot |V(G)|$

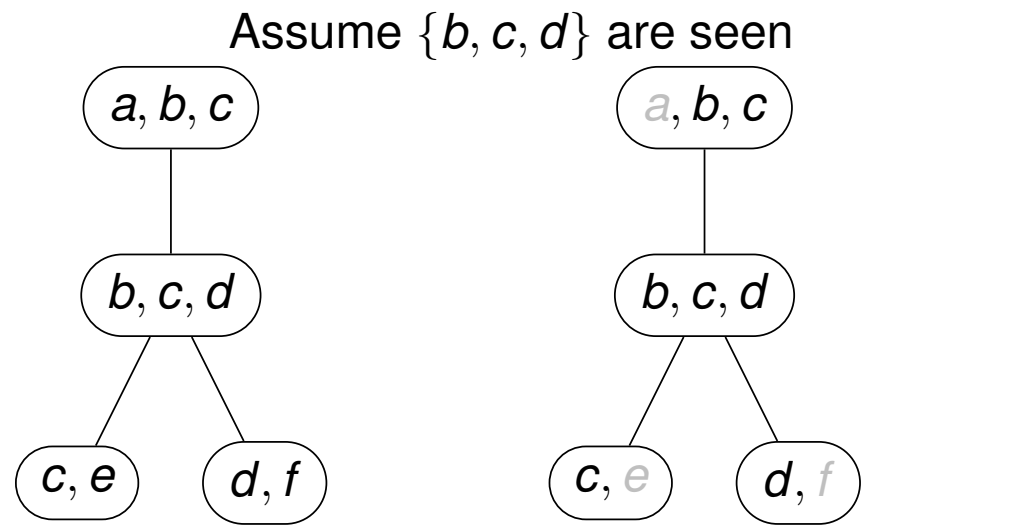
Bodlaender/Paper/Book of Downey and Fellows:

- ... this is the core for a linear-time algorithm for fixed treewidth graphs

Approach:

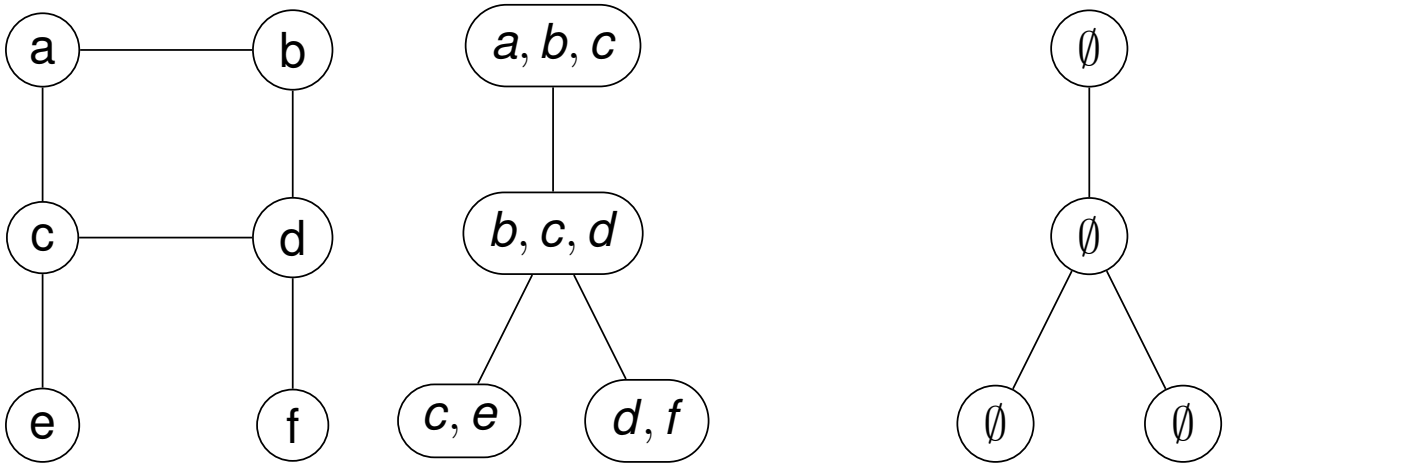
- Show how to construct all (exponentially many) TDs
- Show (in three steps) that we only need necessary information with at most $2^{\mathcal{O}(\hat{tw}^3)}$ equivalence classes

Restricting the TDs to the seen vertices



- Let $(T, (X_t)_{v \in V(T)})$ be a TD of G and $U \subseteq V$
- $(T, (X_t \cap U)_{v \in V(T)})$ is a TD of $G[U]$, called partial TD
- In node $\hat{t}$ of the given TD, we enumerate all TDs of $G[U]$ for the seen vertices U with at most $|V(G)|$ nodes

Enumerating Partial TDs: Leaves

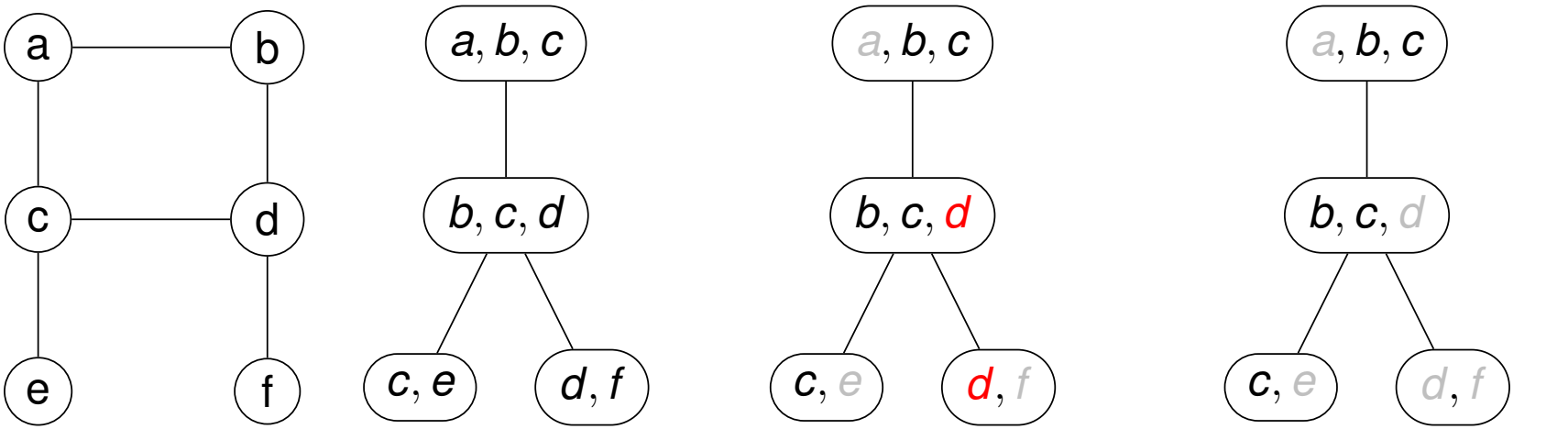


graph a TD at the root TD to be constructed at a leaf

- For any tree T with $|V(G)|$ nodes construct partial TD $(T, (\emptyset)_{t \in T})$

Enumerating Partial TDs: Introduce-Node

Assume $\{b, c, d\}$ are seen and d is introduced

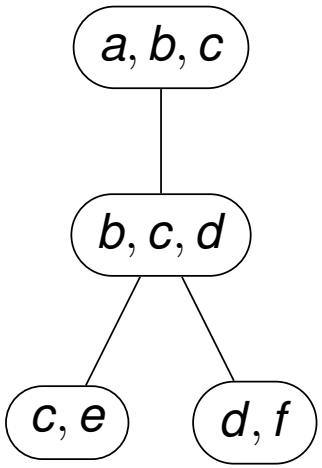


graph | a TD at the root | TD to be constructed | TD constructed for t'

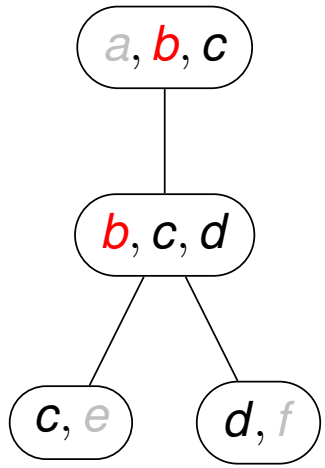
- Add the vertex introduced to the bags of a subtree such that all seen edges are covered

Enumerating Partial TDs: Join-Node

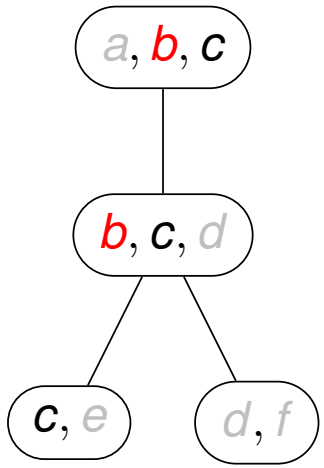
Assume $\{b, c, d\}$ are seen, b is current and c only in first subtree, d only in second



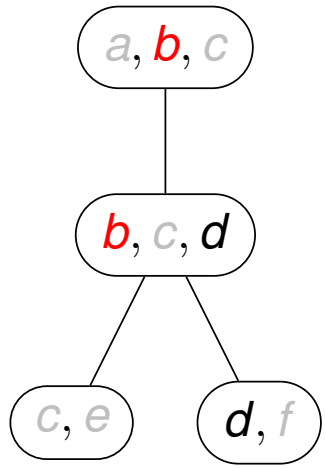
a TD at the root



TD to be constructed



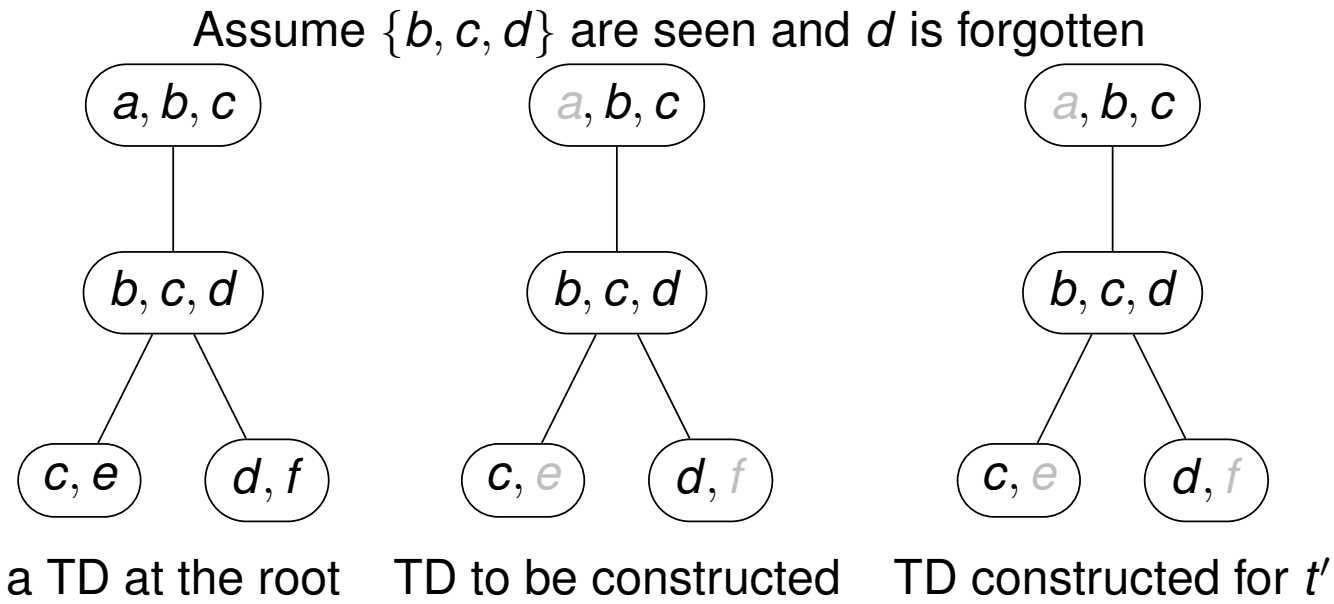
TD constructed for t_1



TD constructed for t_2

- We want to compute $J^t((T^1, (X_t^1)_{v \in V(T^1)}), (T^2, (X_t^2)_{v \in V(T^2)}))$
- Let U be the current vertices
- Only possible if $T^1 = T^2$ and $X_t^1 \cap U = X_t^2 \cap U$ for all $t \in V(T^1)$
- The only possibility is: $(T^1, (X_t^1 \cup X_t^2)_{t \in V(T)})$

Enumerating Partial TDs: Forget-Node

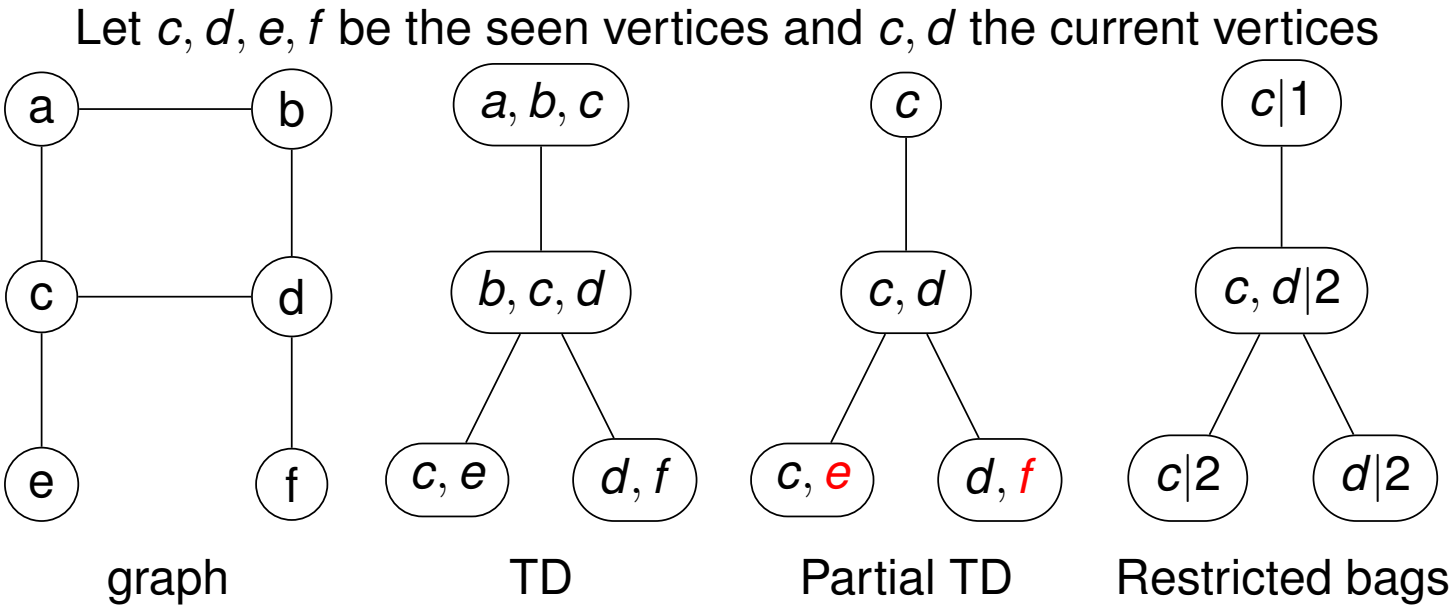


- As the seen vertices stay the same, all TDs are already constructed in t'
- In the following, we always ignore the leaves and the forget-nodes as they are the simplest

Defining equivalence classes

- The number of trees constructed in the leaves grows exponentially with $|V(G)|$
- For each node, we have roughly $|V(G)|^{\hat{tw}}$ many choices for the vertices of the bag
- We define classes such that the number of classes only depends on $\hat{tw}$:
  1. Restricted Bag Representation: Reduce the number of choices for the vertices of a bag to $\hat{tw} \cdot 2^{\hat{tw}}$
  2. The Core: Reduce the topologies to those with $\hat{tw}$ leaves (and hence $\hat{tw}$ inner nodes of degree at least three)
  3. The Compressed Core: Reduce the length of path of length 2 to roughly $2\hat{tw}2^{2\hat{tw}}$ (only sketched)

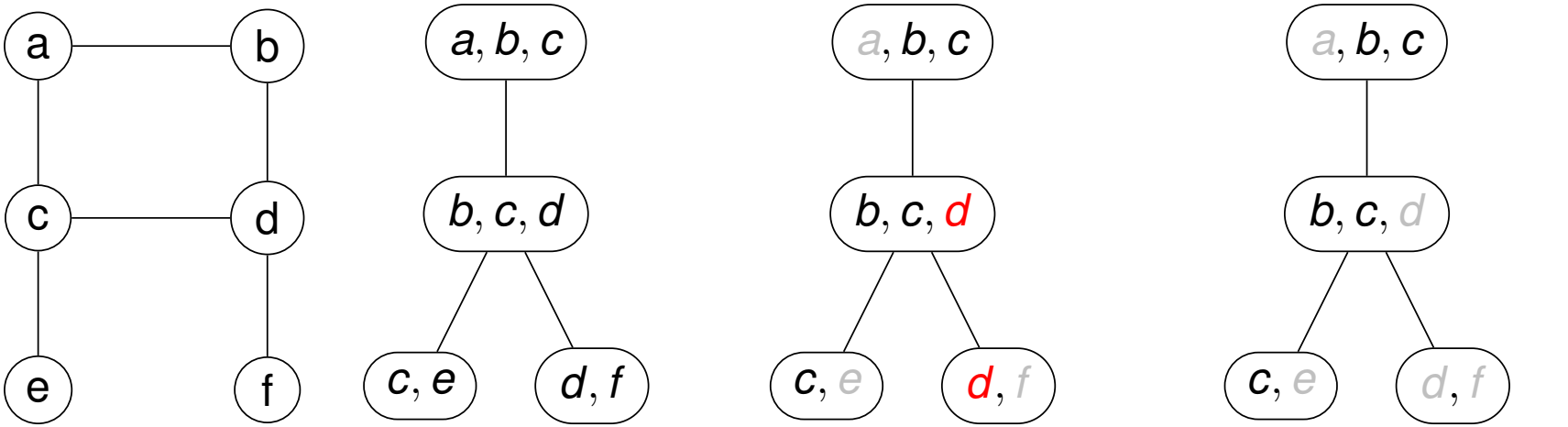
Restricted-Bag representation: Definition



- For each bag X_t of a TD $\mathcal{T} \in \mathcal{TD}^{\hat{t}}$ store only $X_t \cap U$ und $|X_t|$, where U are the current vertices
- Intuitively: we do not store all vertices of a bag, but only the current onces and the size of the bag

Reduced Bags: Introduce-Node

Assume $\{b, c, d\}$ are seen and d is introduced

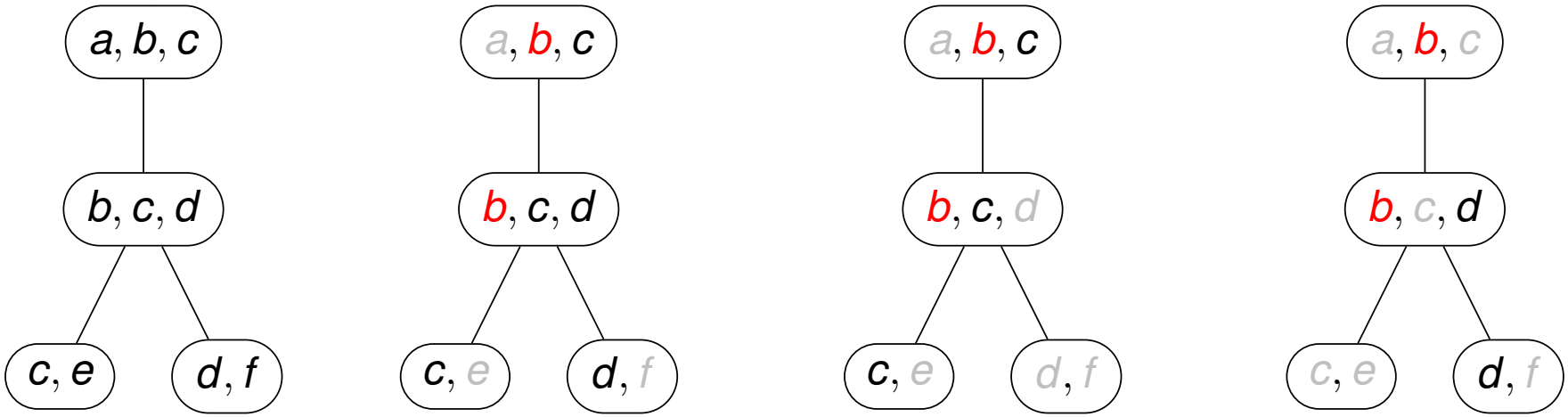


graph | a TD at the root | TD to be constructed | TD constructed for t'

- Recall: Add the vertex introduced to the bags of a subtree such that all seen edges are covered
- Now: This can still be checked, as there can not be an edge from the vertex introduced to a forgotten vertex (as this edge will not be covered)

Reduced Bags: Join-Node

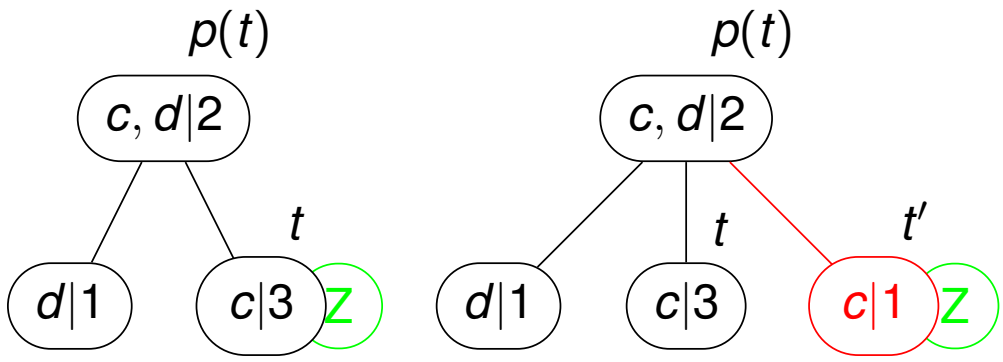
Assume $\{b, c, d\}$ are seen, b is current and c only in first subtree, d only in second



a TD at the root | TD to be constructed | TD constructed for t_1 | TD constructed for t_2

- Recall:
  - Only possible if $T^1 = T^2$ and $X_t^1 \cap U = X_t^2 \cap U$ for all $t \in V(T^1)$
  - The only possibility is: $(T^1, (X_t^1 \cup X_t^2)_{t \in V(T)})$
- Instead of $X_t^1 \cup X_t^2$, we only store $(X_t^1 \cup X_t^2) \cap U = X_t^1 \cap U$ and its size $|X_t^1 \cup X_t^2| = |X_t^1| + |X_t^2| - |X_t^1 \cap U|$

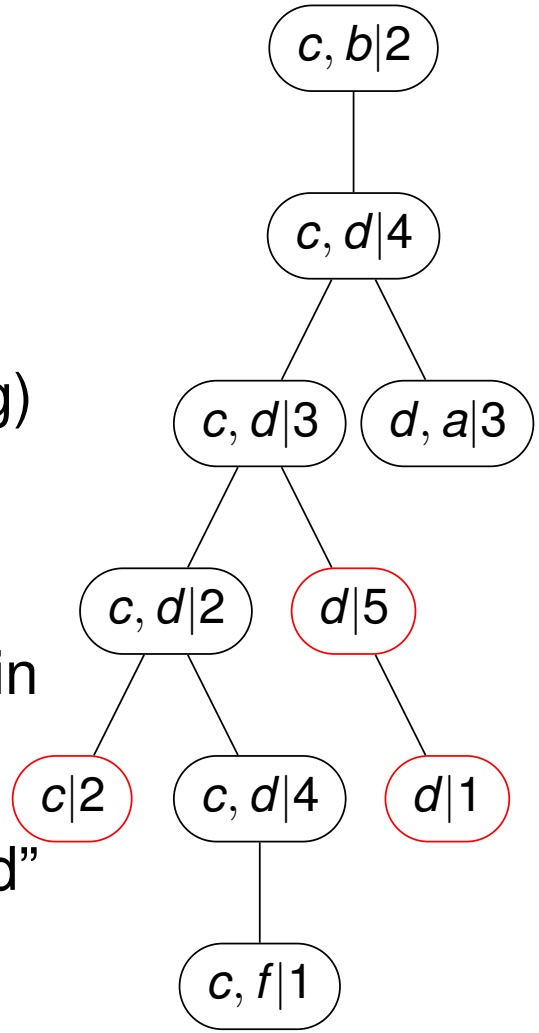
The Core: Structural Property



- Let $\hat{t}$ be a node of the given TD, t be a leaf of the constructed TD and U the current vertices
- Assume $X_t \cap U \subseteq X_{p(t)} \cap U$ and X_t contains at least one forgotten vertex
- There is a minimal TD such that X_t does not get an additional vertex
- Summary: A leaf whose current vertices are a subset of those of the parent either has no forgotten or no unseen vertices

The Core: Definition

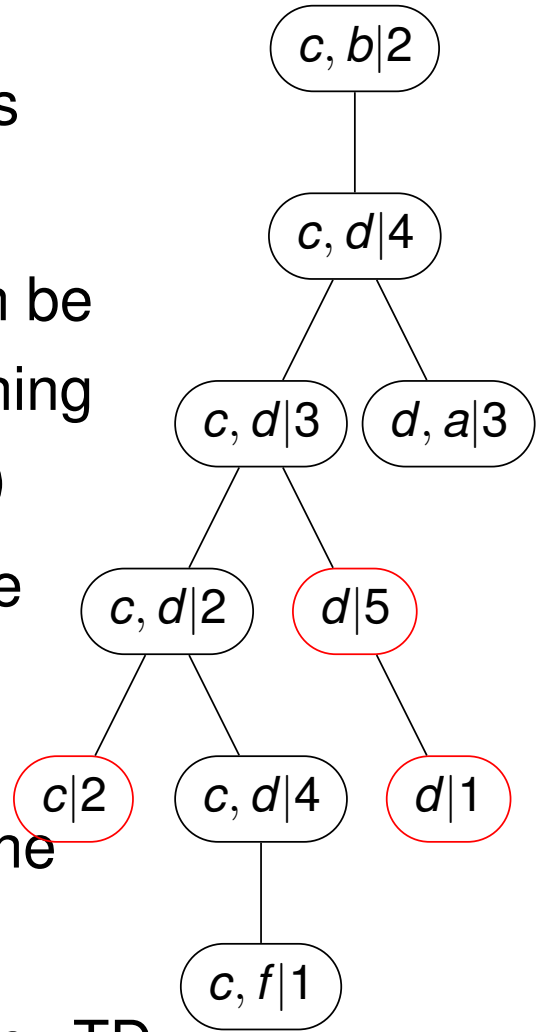
- Let U be the current vertices
- Recursively remove all leaves t with $X_t \cap U \subseteq X_{p(t)} \cap U$
  (restricted bag of parent contains restricted bag)
- The core depends on the current vertices
- Notice that we also remove leaves which contain only current and unseen vertices
- In the constructions of J, I, F, we have to "readd" those leaves



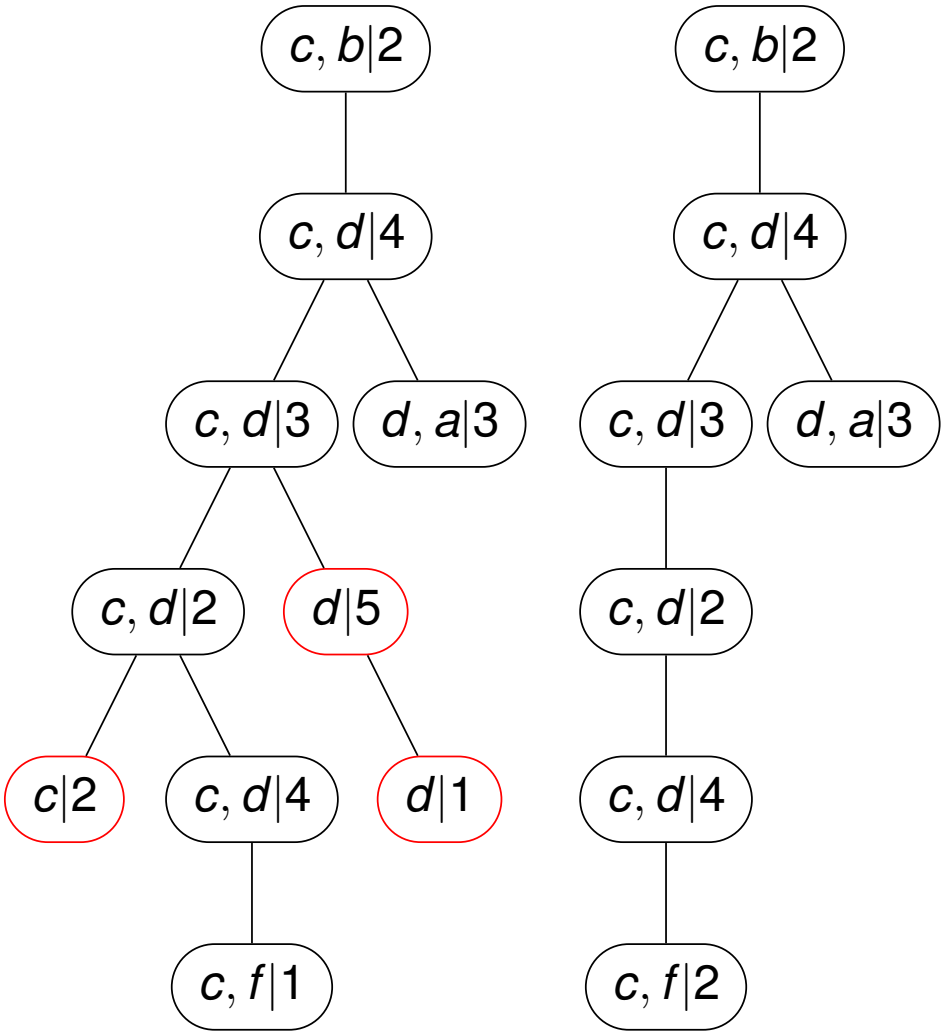
The Core: Compensation in the construction

- When constructing the core, we remove some nodes (leaves whose current vertices in the bags are subsets of those of the parent)
- We compensate by considering all TD which can be obtained by iteratively adding leaves only containing current vertices (sufficient by structural property)
- This constructs all TD which could have been the origin of the core
- But this constructs more TDs than the origin of the core

→ no problem, all such additions can be made to any TD



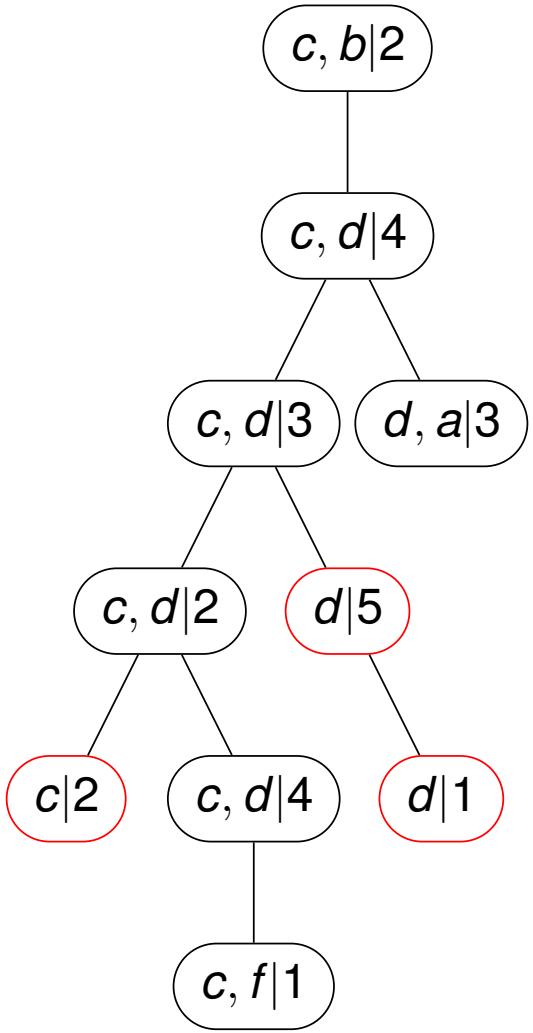
The Core: Example and Number of Leaves



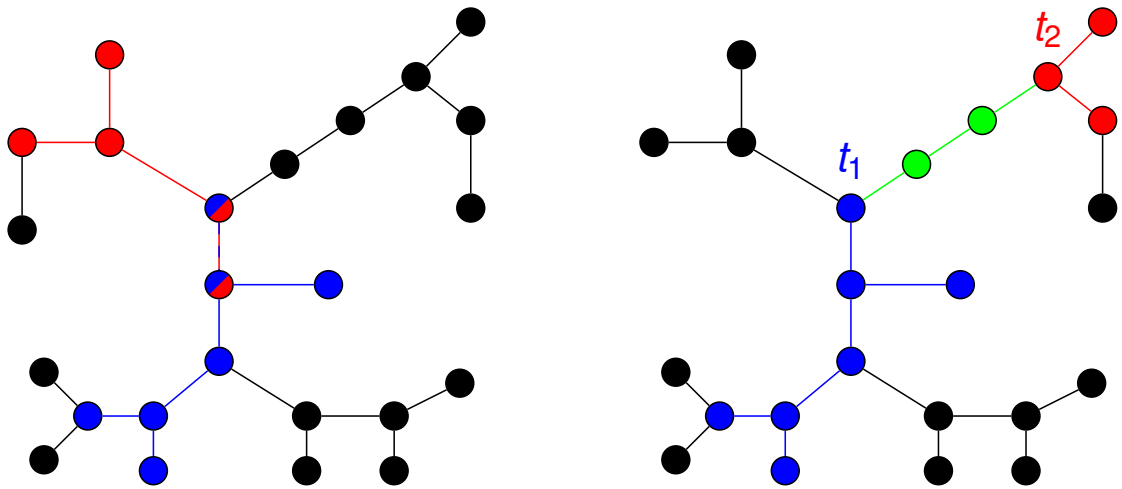
- Every leaf contains a vertex which is not in the parent
- This vertex is contained only in the leaf
- Hence, there are at most $\hat{tw} + 1$ leaves

The Core: Construction for join-nodes

- The restricted bags in $\mathcal{T}^1$ und $\mathcal{T}^2$ (and the constructed $\mathcal{T}$) have to be the same
- Hence, the cores are the same
- Hence, we can do the construction directly on the cores

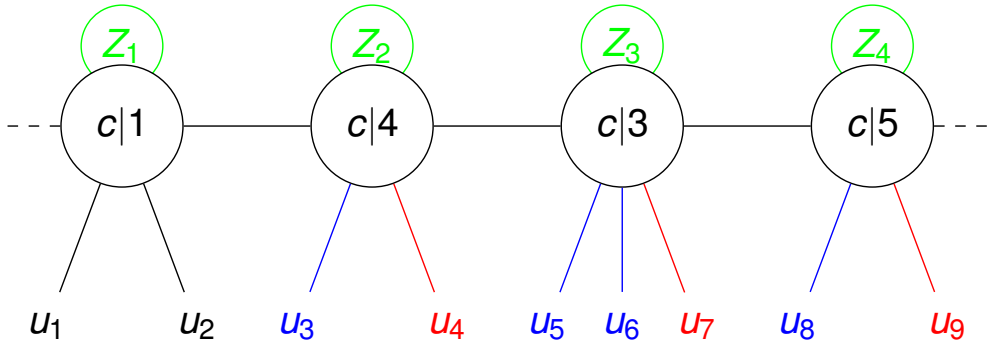


The Core: Construction for introduce-nodes



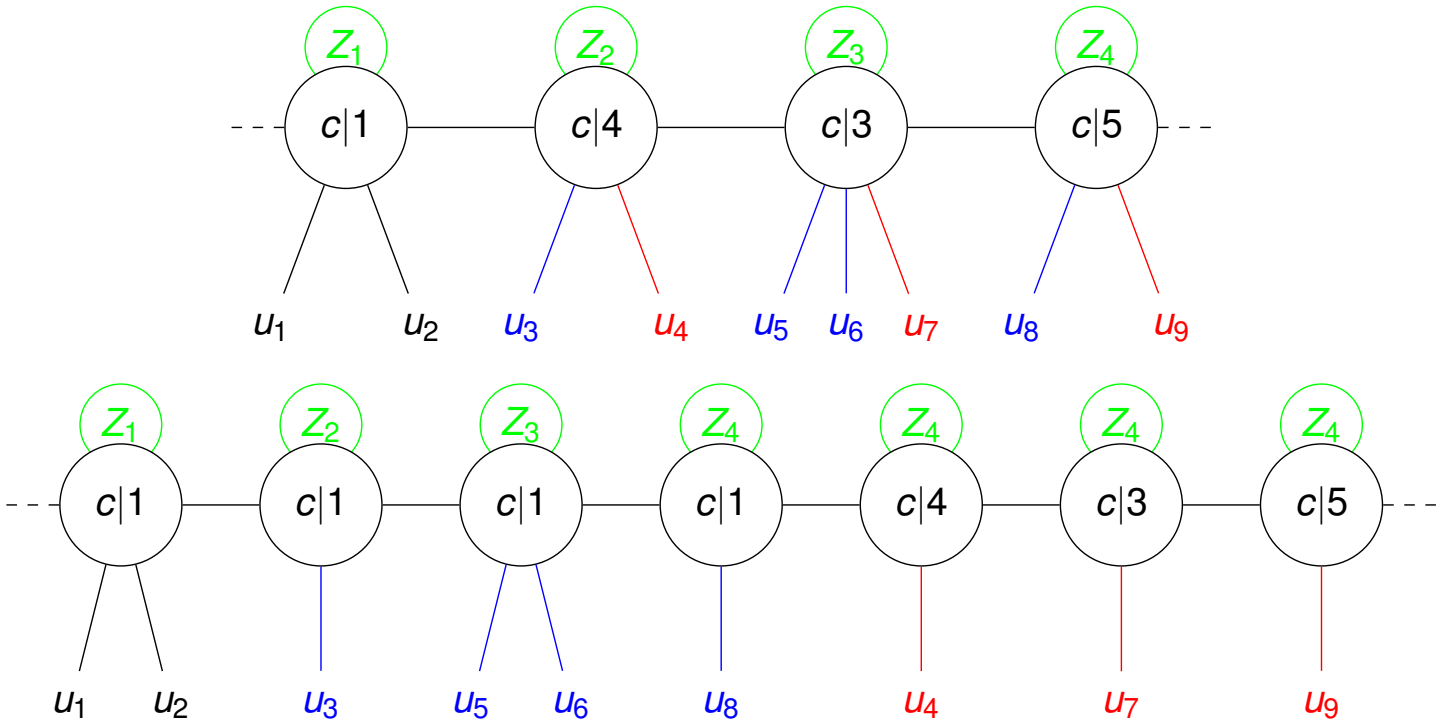
- Case 1: v is introduced to at least one bag of the core
  - The core does not change
- Case 2: v is not introduced to any bag of the core
  - The core will have exacly one bag containing v
  - . . . which is attached to the core by a path whose bags are nested subsets
  - We have to enumerate all possibilities

Compressed Core: Structural Property



- We want to reduce the length of path of nodes of degree 2 to a number bounded in $\hat{tw}$
- There can be at most $2\hat{tw} + 2$ different bags on such path
- Consider a subpath $t_1, \ldots t_k$ of nodes with the same bag and let $a_1, \ldots, a_k$ be the sizes of the bags
- Assume there are $i < j$ such that $a_i < a_\ell \leq a_j$ for all $i < \ell < j$
- Then there is a TD such that the same set of vertices is added to all t_ℓ $(i \leq \ell \leq j)$

“Proof”



- Blue Edges: No forgotten vertices in the bags
- Red Edges: No unseen vertices in the bags

Compressed Core

- Definition: Iteratively remove all nodes whose sizes correspond to
  - $a_{i+1}, \ldots, a_{j-1}$ of an integer sequence with $a_i \leq a_\ell \leq a_j$
  - a_i, if $a_i = a_{i+1}$
- The resulting sequences of sizes is unique sequence called the typical sequence of $a_1, \ldots, a_k$
- Notice: in the lemma we had "$a_i < \ldots$" and not "$a_i \leq \ldots$"
- In the construction for join-, introduce-, and forget-nodes, we have to consider all possible origins of the typcial sequences
- . . . and have to show that TDs with all resulting typical sequences can be constructed, no matter which integer sequence was the origin

Finalizing the Argument

- The number of typical sequences of $t\hat{w}$ different integers is at most $2^{2t\hat{w}}$
- Hence: the number of compressed cores is bounded in $t\hat{w}$ (in $2^{\mathcal{O}(t\hat{w}^3)}$)
- Furthermore all computations can be done in $2^{\mathcal{O}(t\hat{w}^3)}$ per node of the given TD
- This gives a linear-time algorithm for small treewidth graphs

Summary

- We sketched a well-known linear-time algorithm to compute the treewidth of graphs of small treewidth
  - Enumerate all tree-decompositions
  - . . . but only one per equivalence class
  - Number of equivalence classes is bounded in the treewidth
- This gives a much simplified description
- This algorithm is not practical
- . . . but of high theoretical interest (as many FPT-results are based on it)
- Only recently a linear-time approximation algorithm for the treewidth with is only single exponential in tw has been found (Bodlaender et-al, 2016: A $c^k n$ 5-Approximation Algorithm for Treewidth)