Complexity of maintaining (canonical) tree-like decompositions

Frank Fuhlbrück

April 2nd, 2014

Dynamic complexity vs. static complexity

- L belongs to static class $\mathcal{C}$ if there is a machine/circuit/formula to decide $I \stackrel{?}{\in} L$ for a given instance I (string/structure)

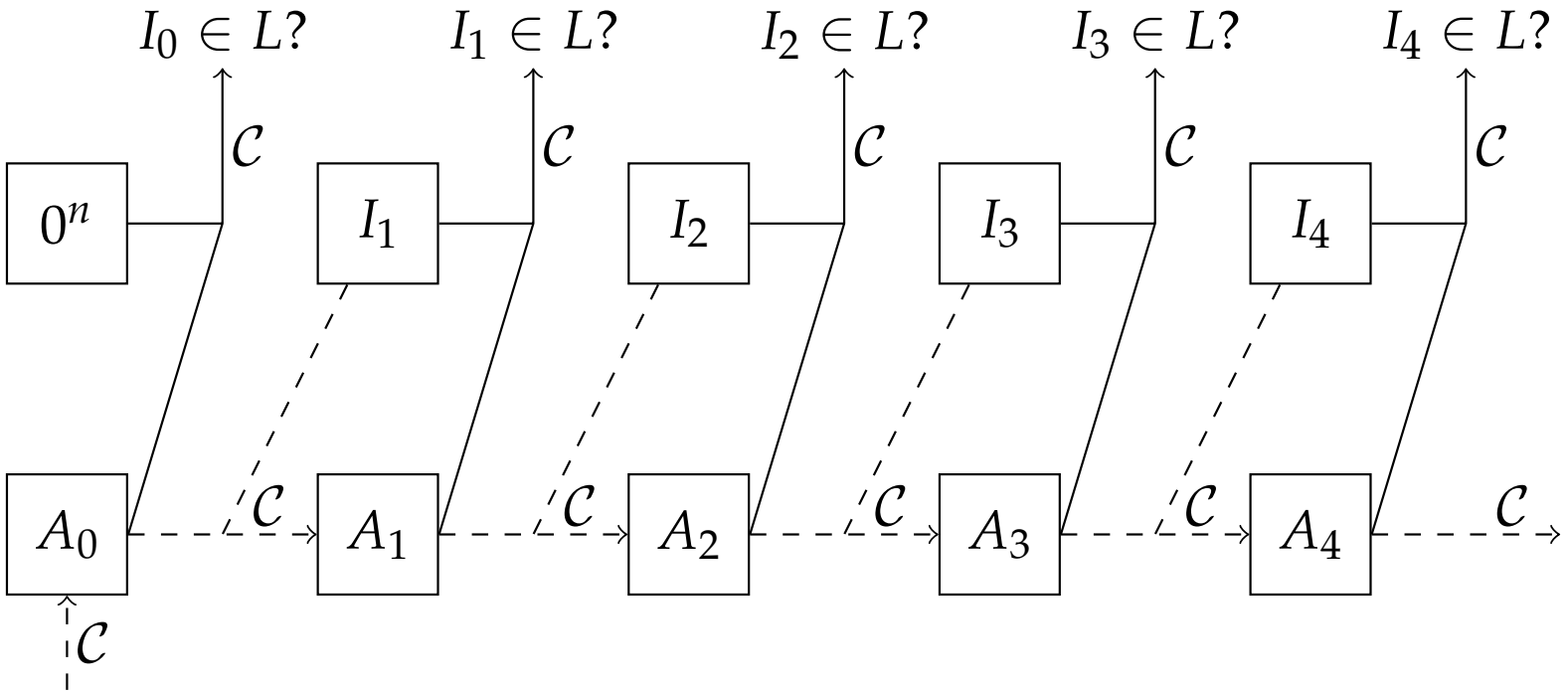
Dynamic complexity vs. static complexity

- L belongs to static class $\mathcal{C}$ if there is a machine/circuit/formula to decide $I \stackrel{?}{\in} L$ for a given instance I (string/structure)
- L belongs to a dynamic class Dyn$\mathcal{C}$ if there is a machine/circuit/formula to update an auxiliary string/structure and to decide $I \stackrel{?}{\in} L$ using the auxiliary information [MSVT94; PI97].

Dynamic complexity vs. static complexity

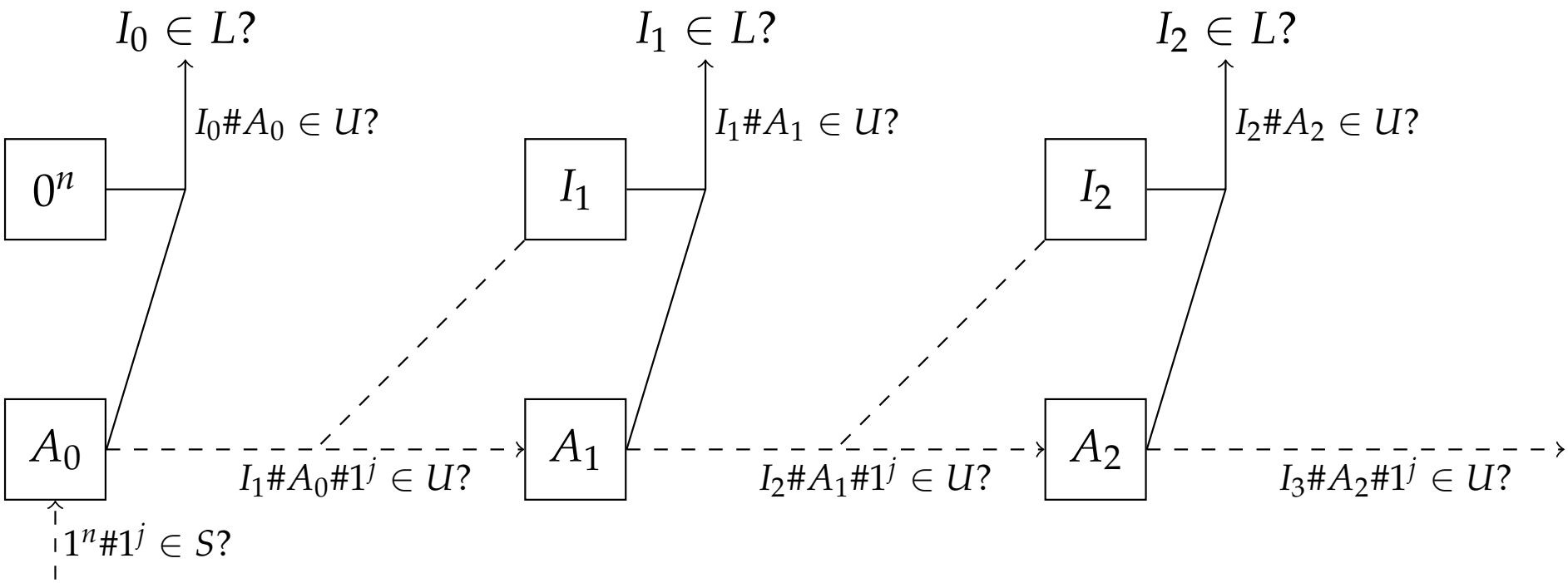
- L belongs to static class $\mathcal{C}$ if there is a machine/circuit/formula to decide $I \stackrel{?}{\in} L$ for a given instance I (string/structure)
- L belongs to a dynamic class Dyn$\mathcal{C}$ if there is a machine/circuit/formula to update an auxiliary string/structure and to decide $I \stackrel{?}{\in} L$ using the auxiliary information [MSVT94; PI97].
- The auxiliary information stems from a series of instances $I_0, \ldots, I_l = I$ differing only slightly, i.e. hamming distance $\Delta(I_i, I_j) = 1$ (strings) or a single tuple inserted/removed (structures).
- For graphs: a single edge inserted or deleted

Dynamic complexity illustrated



I_i - Instances, $\Delta(I_i, I_{i+1}) = 1$
A_i - Auxiliary information

Dynamic complexity illustrated - details



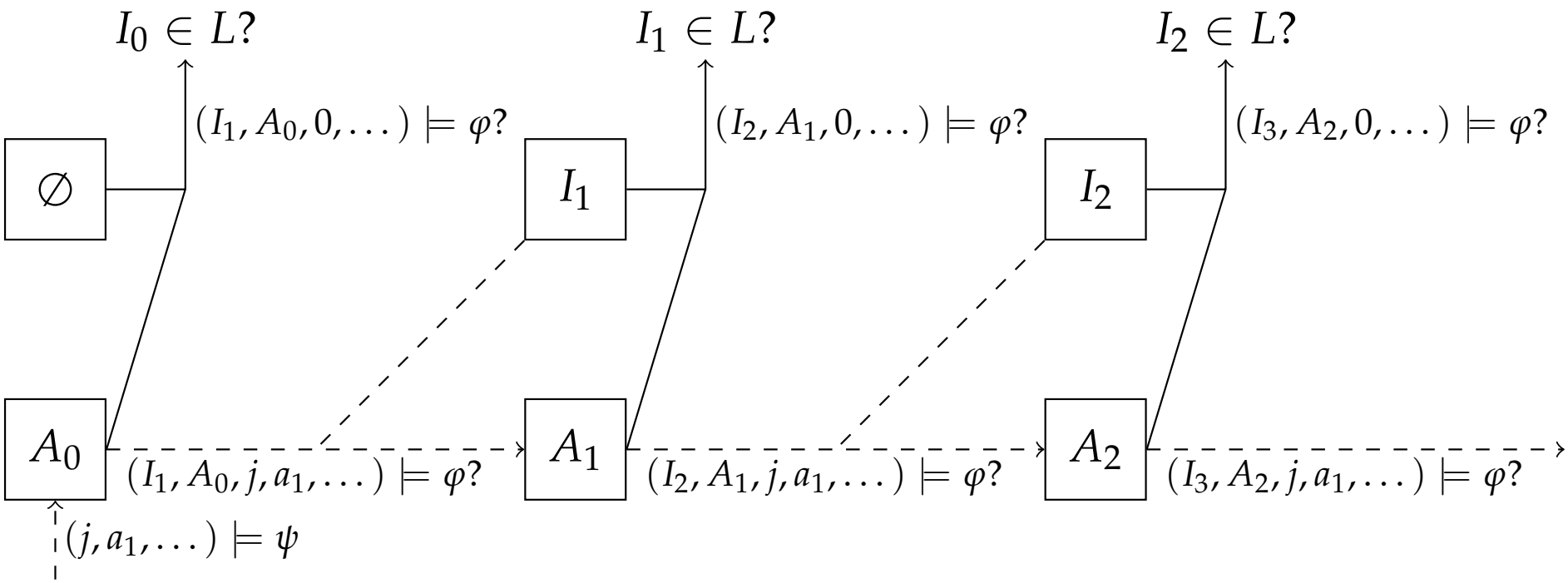
$\{U, S\} \subseteq \mathcal{C}$

I_i - Instances, $\Delta(I_i, I_{i+1}) = 1$

A_i - Auxiliary string:

jth bit of A_i: $A_{i,j} = 1 \Leftrightarrow I_i\#A_{i-1}\#1^j \in U$

Dynamic complexity illustrated - details



$$\{\varphi, \psi\} \subseteq \Phi$$

I_i - Instances, one tuple changed from I_i to I_{i+1}

A_i - Auxiliary structure:

$$(a_1, \dots, a_m) \in R_j^{A_i} \Leftrightarrow (I_i, A_{i-1}, j, a_1, \dots, a_m, \dots) \models \varphi$$

Important dynamic classes

- DynFO: dynamic first order (without bit predicate, arithmetic or order)

Important dynamic classes

- DynFO: dynamic first order (without bit predicate, arithmetic or order)
- Nevertheless, DynFO $\approx$ DynAC^0: we can simulate addition and an order relation on the elements used so far in DynFO

Important dynamic classes

- DynFO: dynamic first order (without bit predicate, arithmetic or order)
- Nevertheless, DynFO $\approx$ DynAC^0: we can simulate addition and an order relation on the elements used so far in DynFO
- Reachability [PI97] and computing distances [Meh13] in undirected graphs is in DynFO

Important dynamic classes

- DynFO: dynamic first order (without bit predicate, arithmetic or order)
- Nevertheless, DynFO $\approx$ DynAC^0: we can simulate addition and an order relation on the elements used so far in DynFO
- Reachability [PI97] and computing distances [Meh13] in undirected graphs is in DynFO
- Reachability in directed graphs is in DynTC^0 [Hes03].

Important dynamic classes

- DynFO: dynamic first order (without bit predicate, arithmetic or order)
- Nevertheless, DynFO $\approx$ DynAC^0: we can simulate addition and an order relation on the elements used so far in DynFO
- Reachability [PI97] and computing distances [Meh13] in undirected graphs is in DynFO
- Reachability in directed graphs is in DynTC^0 [Hes03].
- further interesting: DynNC^i, DynL, ...

Important dynamic classes

- DynFO: dynamic first order (without bit predicate, arithmetic or order)
- Nevertheless, DynFO $\approx$ DynAC^0: we can simulate addition and an order relation on the elements used so far in DynFO
- Reachability [PI97] and computing distances [Meh13] in undirected graphs is in DynFO
- Reachability in directed graphs is in DynTC^0 [Hes03].
- further interesting: DynNC^i, DynL, ...
- but DynP is just P

Bounded reductions

- Dyn$\mathcal{C}$ is not known to be closed under "F$\mathcal{C}$-reductions" in general

Bounded reductions

- Dyn$\mathcal{C}$ is not known to be closed under "F$\mathcal{C}$-reductions" in general
- e.g. there are problems in DynFO that are P-complete under first order reductions [MSVT94; PI97]

Bounded reductions

- Dyn$\mathcal{C}$ is not known to be closed under "F$\mathcal{C}$-reductions" in general
- e.g. there are problems in DynFO that are P-complete under first order reductions [MSVT94; PI97]
- therefore, we define:

Definition

$A \leq_{b\mathcal{C}} B$ if there is an $A \leq_{\mathcal{C}} B$ reduction function f with

$$\forall c \exists c' : \Delta(I, J) \leq c \Rightarrow \Delta(f(I), f(J)) \leq c' .$$

That means every input bit influences only a constant number of output bits.

Known Results for Dynamic Canonization

- canonization of trees is in DynFO [Ete98]

Known Results for Dynamic Canonization

- canonization of trees is in DynFO [Ete98]
- canonization of planar 3-connected graphs in DynFO^+ (polynomial time precomputation) [Meh13]

Tree-depth

Definition ([NO06; BDK12])

Let G be a graph with connected components $C_1, \ldots, C_l$. Then

$$\mathrm{td}(G) = \begin{cases} 1 & |V(G)| = 1 \\ \max_{i \in [l]} \mathrm{td}(G[C_i]) & l > 1 \\ 1 + \min_{v \in V(G)} \mathrm{td}(G - v) & \text{else} \end{cases} .$$

Tree-depth

Definition ([NO06; BDK12])

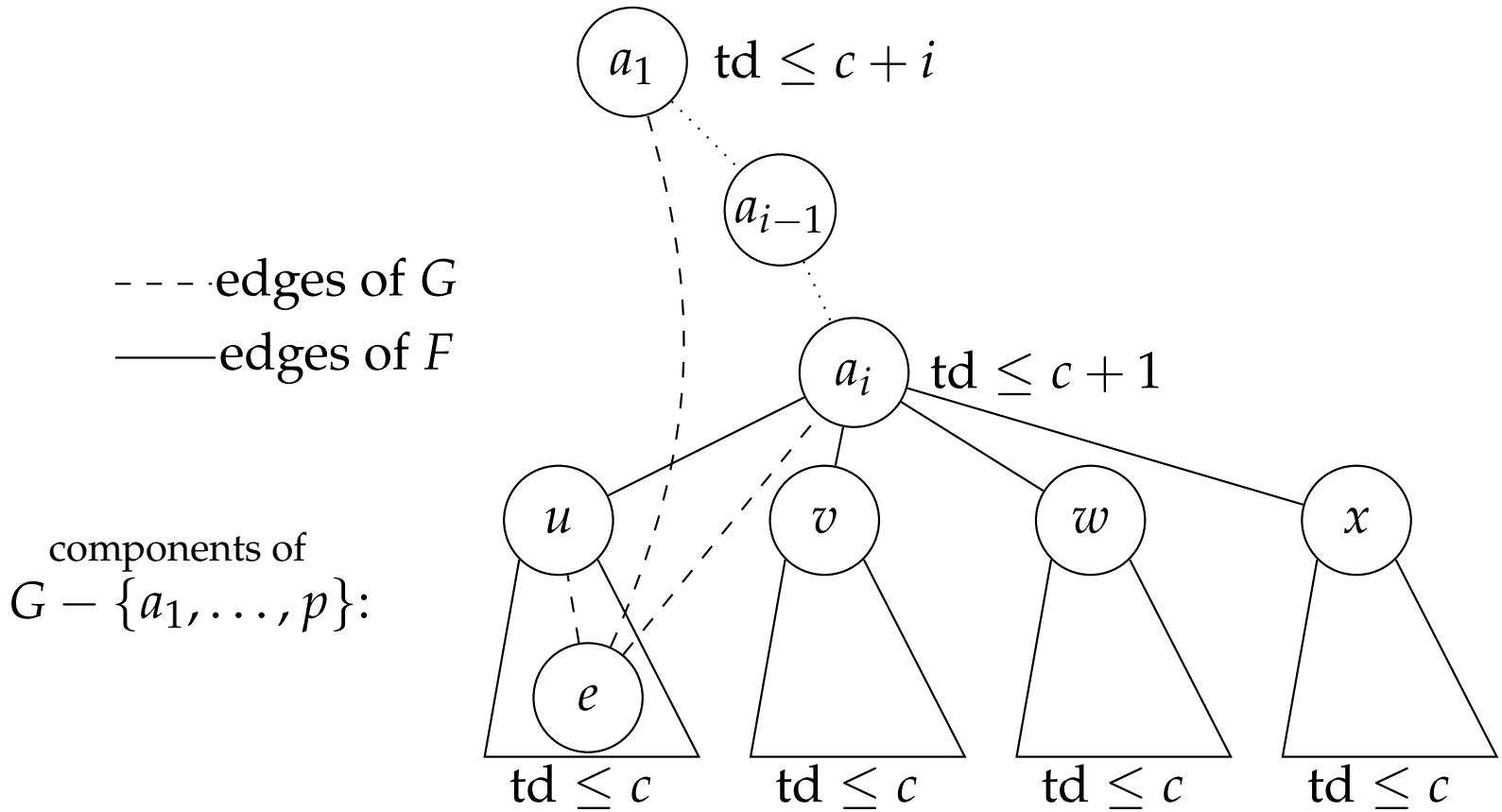
Let G be a graph with connected components $C_1, \ldots, C_l$. Then

$$\mathrm{td}(G) = \begin{cases} 1 & |V(G)| = 1 \\ \max_{i \in [l]} \mathrm{td}(G[C_i]) & l > 1 \\ 1 + \min_{v \in V(G)} \mathrm{td}(G - v) & \text{else} \end{cases}.$$

A rooted forest $(F, r_1, \ldots, r_l)$ is a tree-depth decomposition of G if $V(G) = V(F)$ and

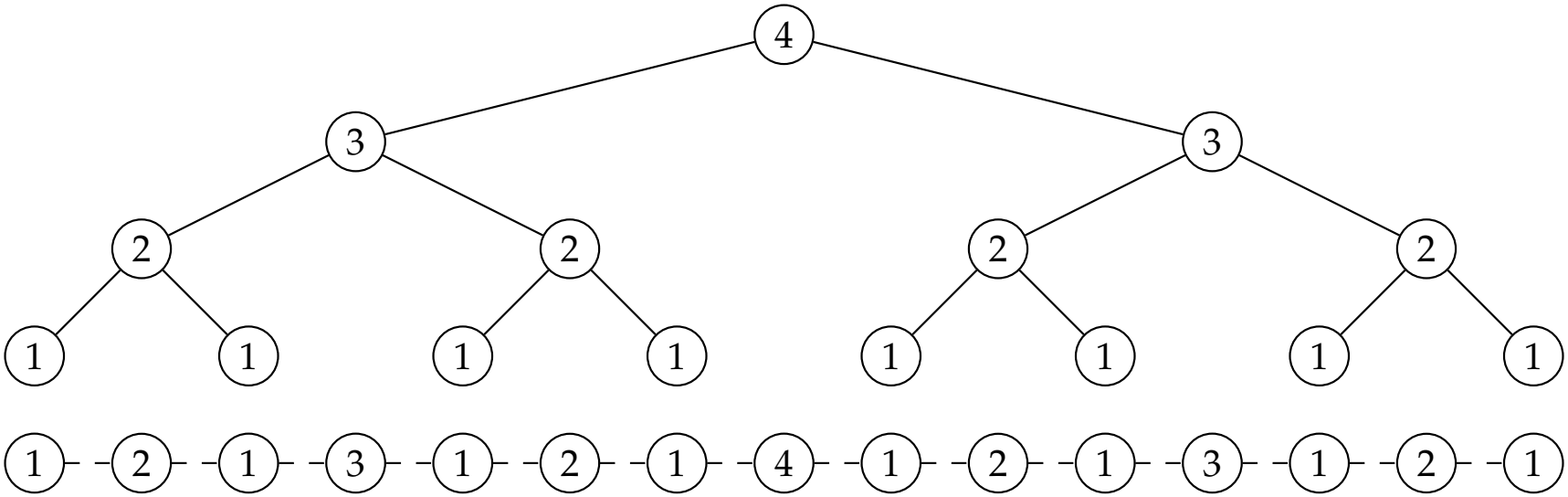
$$\begin{cases} r_1 = v & V(G) = \{v\} \\ \forall 1 \leq i \leq l : (F[C_i], r_i) \text{ is a tree-depth dec. of } G[C_i] & l > 1 \\ (F - r_1, r'_1, \ldots, r'_{l'}) \text{ is a tree-depth dec. of } G - r_1 & \text{else} \\ \text{where } N_F(r_1) = \{r'_1, \ldots, r'_{l'}\} \end{cases}.$$

Tree-depth decomposition - illustrated



Every neighbor of e in G lies on path from e to a_1 in F.

Tree-depth decomposition of a path



A path of length 14 has tree-depth 4.

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children
  - $\Rightarrow$ use isomorphism order on tree depth dec. from [BDK12]

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children
  - $\Rightarrow$ use isomorphism order on tree depth dec. from [BDK12]
  - basically Lindell[Lin92], but use adjacency to path from root as first criterion

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children
  - $\Rightarrow$ use isomorphism order on tree depth dec. from [BDK12]
  - basically Lindell[Lin92], but use adjacency to path from root as first criterion
  - $\mathrm{adj}(a_1, \ldots, a_i) = b_1 \ldots b_{i-1}$ where $\forall j < i : b_j = 1 \Leftrightarrow \{a_j, a_i\} \in E(G)$

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children
  - $\Rightarrow$ use isomorphism order on tree depth dec. from [BDK12]
  - basically Lindell[Lin92], but use adjacency to path from root as first criterion
  - $\mathrm{adj}(a_1, \ldots, a_i) = b_1 \ldots b_{i-1}$ where $\forall j < i : b_j = 1 \Leftrightarrow \{a_j, a_i\} \in E(G)$
- updating sorted list in first order:

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children
  - $\Rightarrow$ use isomorphism order on tree depth dec. from [BDK12]
  - basically Lindell[Lin92], but use adjacency to path from root as first criterion
  - $\mathrm{adj}(a_1, \ldots, a_i) = b_1 \ldots b_{i-1}$ where $\forall j < i : b_j = 1 \Leftrightarrow \{a_j, a_i\} \in E(G)$
- updating sorted list in first order:
  - remove changed items (vertices representing components)

Maintaining a canonical tree-depth dec.

- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children
  - $\Rightarrow$ use isomorphism order on tree depth dec. from [BDK12]
  - basically Lindell[Lin92], but use adjacency to path from root as first criterion
  - $\mathrm{adj}(a_1, \ldots, a_i) = b_1 \ldots b_{i-1}$ where $\forall j < i : b_j = 1 \Leftrightarrow \{a_j, a_i\} \in E(G)$
- updating sorted list in first order:
  - remove changed items (vertices representing components)
  - update indices (subtract c if index $> c$ removed indices)

Maintaining a canonical tree-depth dec.

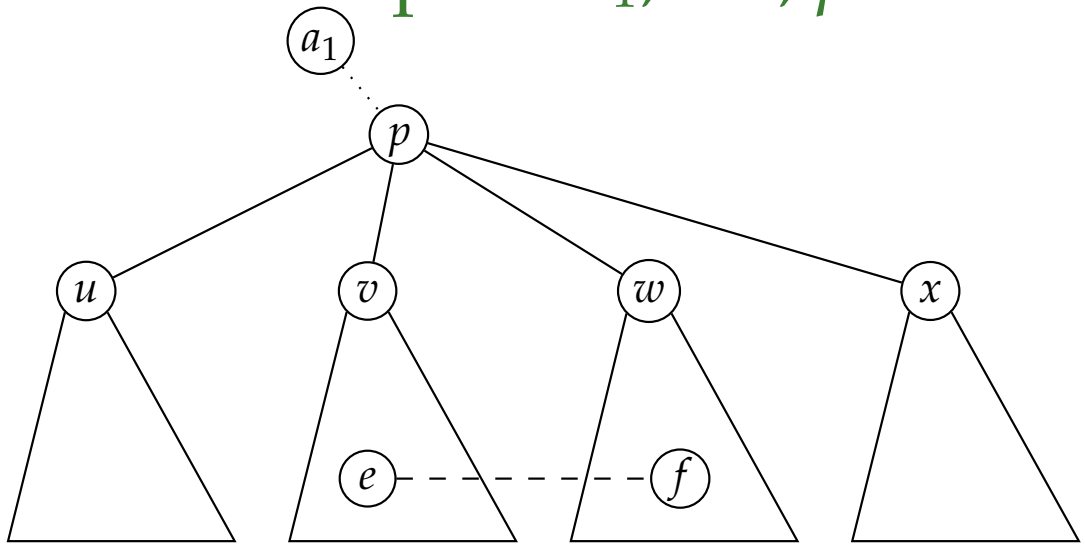
- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children
  - $\Rightarrow$ use isomorphism order on tree depth dec. from [BDK12]
  - basically Lindell[Lin92], but use adjacency to path from root as first criterion
  - $\mathrm{adj}(a_1, \ldots, a_i) = b_1 \ldots b_{i-1}$ where $\forall j < i : b_j = 1 \Leftrightarrow \{a_j, a_i\} \in E(G)$
- updating sorted list in first order:
  - remove changed items (vertices representing components)
  - update indices (subtract c if index $> c$ removed indices)
  - insert changed items at minimal position

Maintaining a canonical tree-depth dec.

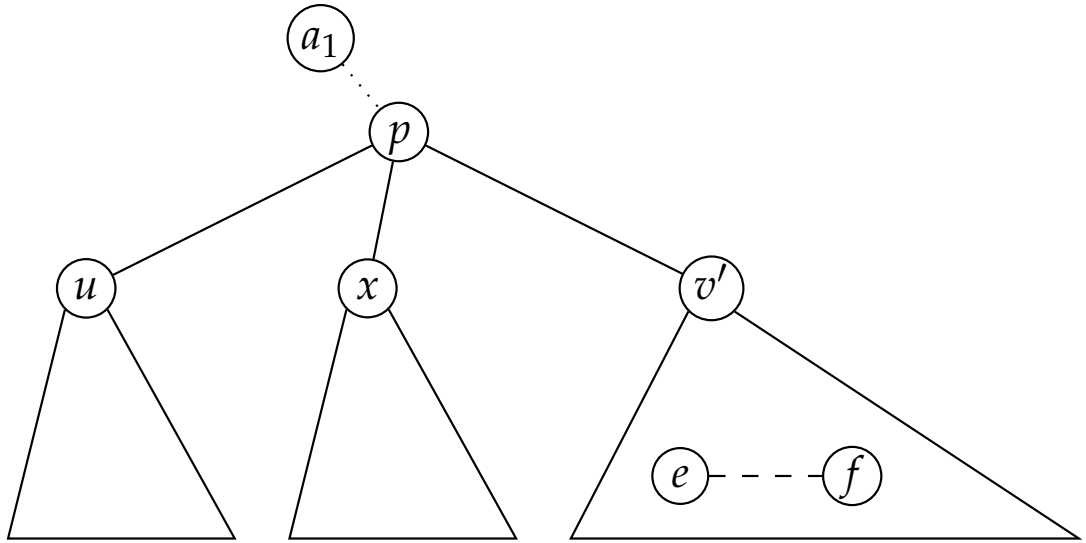
- For graphs with $\mathrm{td}(G) \leq k$
  - we can keep a can. decomposition of $G - \{a_1, \ldots, a_i\}$ for every i-tuple $(a_1, \ldots, a_i)$ (order matters!)
  - we can update every level *consecutively* (bottom-up)
- Main task for a canonical tree is sorting the children
  - $\Rightarrow$ use isomorphism order on tree depth dec. from [BDK12]
  - basically Lindell[Lin92], but use adjacency to path from root as first criterion
  - $\mathrm{adj}(a_1, \ldots, a_i) = b_1 \ldots b_{i-1}$ where $\forall j < i : b_j = 1 \Leftrightarrow \{a_j, a_i\} \in E(G)$
- updating sorted list in first order:
  - remove changed items (vertices representing components)
  - update indices (subtract c if index $> c$ removed indices)
  - insert changed items at minimal position
  - update indices (add c if item $>$ c removed items)

≤ 2 components below a fixed path $a_1, \ldots, p$

canonical dec. with root path $a_1, \ldots, p, \ldots$



canonical dec. with root path $a_1, \ldots, p, \ldots$



Using a canonical tree-depth dec. for isomorphism

- uniquely color every vertex a_i with the child indices of its ancestors $a_1, \ldots, a_{i-1}$ in the tree-depth dec. (like in [Meh13], but simpler)

Using a canonical tree-depth dec. for isomorphism

- uniquely color every vertex a_i with the child indices of its ancestors $a_1, \ldots, a_{i-1}$ in the tree-depth dec. (like in [Meh13], but simpler)
  - the simulation of numbers has to consistent in both graphs

Using a canonical tree-depth dec. for isomorphism

- uniquely color every vertex a_i with the child indices of its ancestors $a_1, \ldots, a_{i-1}$ in the tree-depth dec. (like in [Meh13], but simpler)
  - the simulation of numbers has to consistent in both graphs
- add $\mathrm{adj}(a_1, \ldots, a_i)$ to this color

Using a canonical tree-depth dec. for isomorphism

- uniquely color every vertex a_i with the child indices of its ancestors $a_1, \ldots, a_{i-1}$ in the tree-depth dec. (like in [Meh13], but simpler)
  - the simulation of numbers has to consistent in both graphs
- add $\mathrm{adj}(a_1, \ldots, a_i)$ to this color
- check if for each vertex of a graph G_1 there is a vertex in G_2 with the same color and vice versa

Using a canonical tree-depth dec. for isomorphism

- uniquely color every vertex a_i with the child indices of its ancestors $a_1, \ldots, a_{i-1}$ in the tree-depth dec. (like in [Meh13], but simpler)
  - the simulation of numbers has to consistent in both graphs
- add $\mathrm{adj}(a_1, \ldots, a_i)$ to this color
- check if for each vertex of a graph G_1 there is a vertex in G_2 with the same color and vice versa

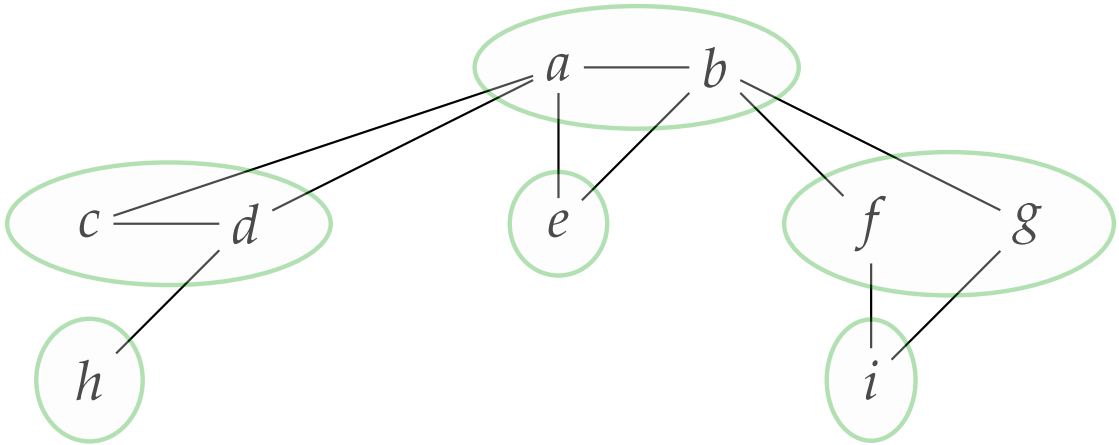
Using a canonical tree-depth dec. for isomorphism

- uniquely color every vertex a_i with the child indices of its ancestors $a_1, \ldots, a_{i-1}$ in the tree-depth dec. (like in [Meh13], but simpler)
  - the simulation of numbers has to consistent in both graphs
- add $\mathrm{adj}(a_1, \ldots, a_i)$ to this color
- check if for each vertex of a graph G_1 there is a vertex in G_2 with the same color and vice versa

Theorem

GraphIso for graphs of bounded tree-depth is in DynFO (*and* TC^0).

Tree distance decompositions

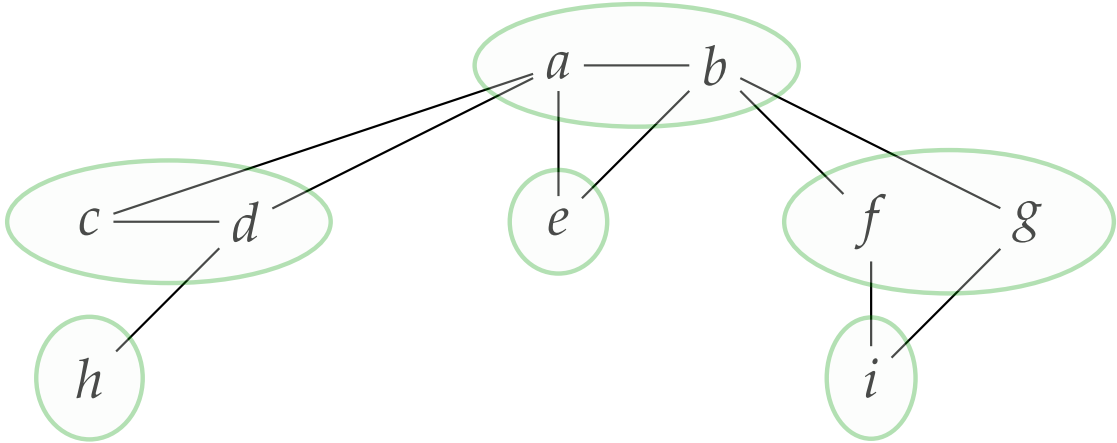


Definition

Let G be a graph. A (*minimal*) *tree distance decomposition* of G is a triple (B, T, r), where

1. T is a tree and $r \in V(T)$,
2. $B : V(T) \to \wp(V(G))$ such that $\mathrm{rng}(B)$ is a partition of $V(G)$ and
3. for all $e \in E(G)$ $\exists t, u \in V(T)$ such that $e \subseteq B_t \cup B_u$ and $t = u$ or $\{t, u\} \in E(T)$ and

Tree distance decompositions

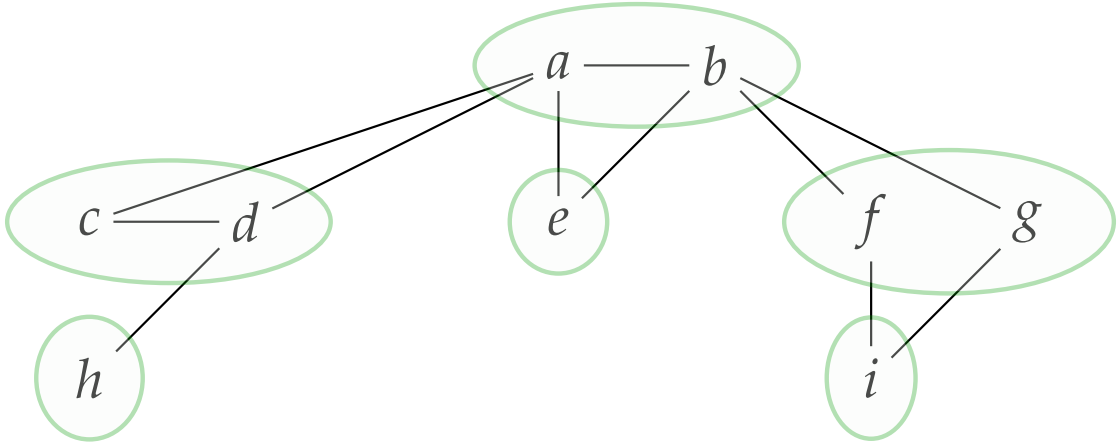


Definition

Let G be a graph. A (*minimal*) *tree distance decomposition* of G is a triple (B, T, r), where

4. for all $t \in V(T)$ and all $v \in B_t$: $\min_{u \in B_r} \mathrm{d}(u, v) = \mathrm{d}(r, t)$ and

Tree distance decompositions

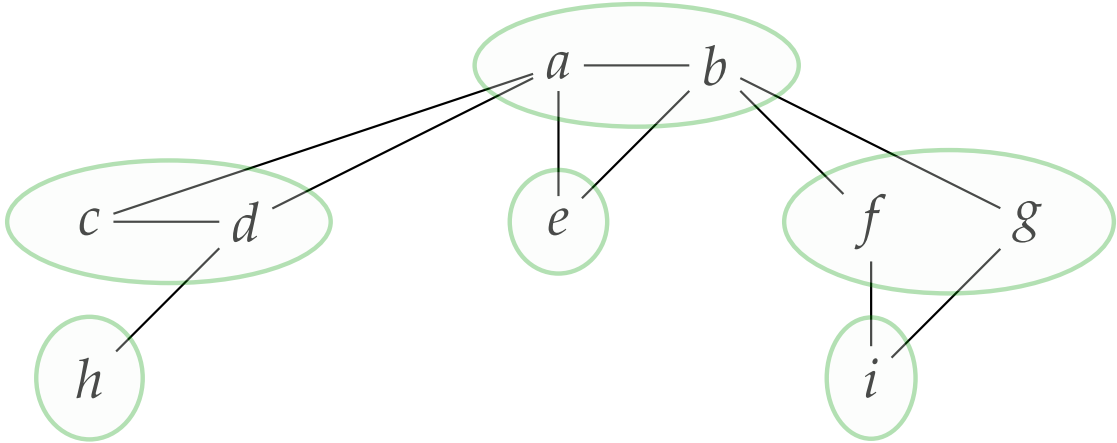


Definition

Let G be a graph. A (*minimal*) *tree distance decomposition* of G is a triple (B, T, r), where

5. (minimality) for all $t \in V(T)$: $G[\bigcup_{t' \in V(T_t)}]$ is connected, where T_t is the subtree rooted at t.

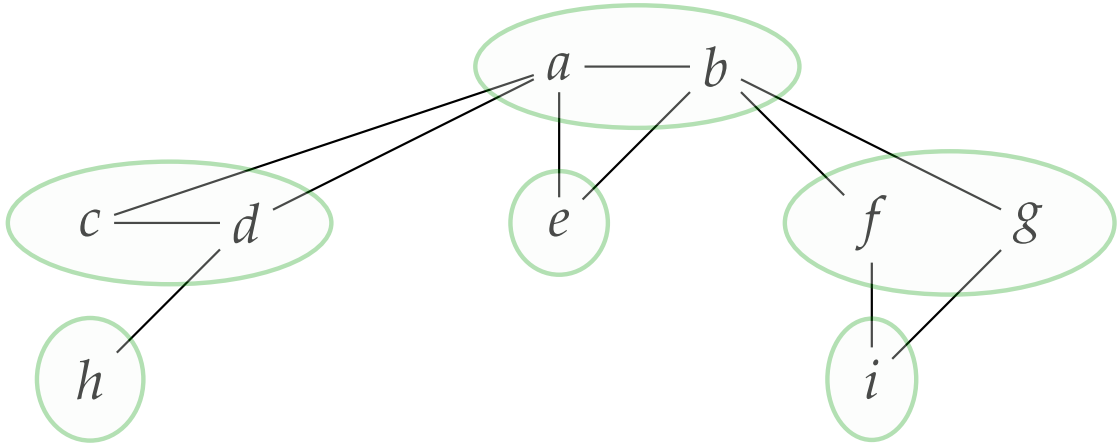
Tree distance decompositions



Definition

Let G be a graph. A (*minimal*) *tree distance decomposition* of G is a triple (B, T, r), where ...
It is a *path distance decomposition* if T is a path.
The *width* of a tree distance decomposition is $\max_{t \in V(T)} |B_t|$. The *path/tree distance width* $\mathrm{pdw}(G)/\mathrm{tdw}(G)$ is the minimal width of a path/tree distance decomposition of G.

Maintaining a minimal tree distance decomposition

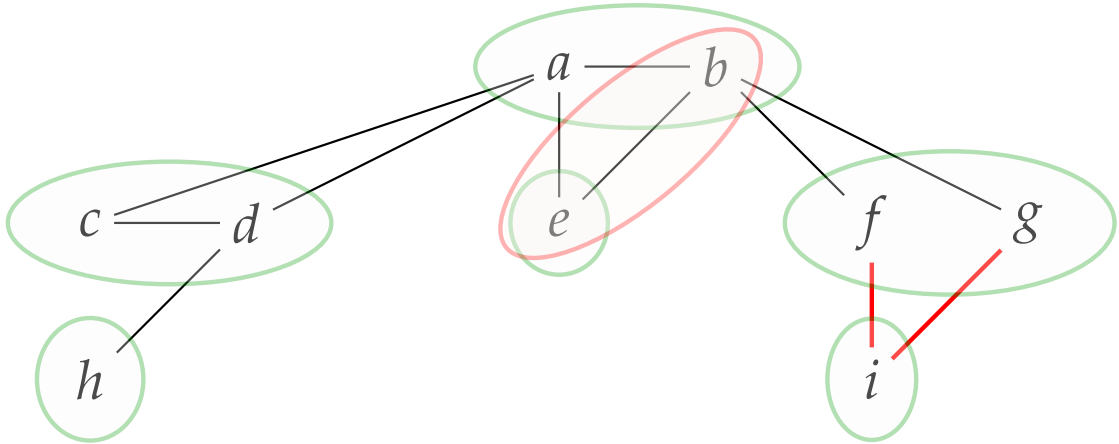


What constitutes a bag w.r.t. to root bag $R = \{a, b\}$?

We need a predicate that tells us for every k-element set (henceforth "bag") whether it is a bag.

Computing pairwise distances is in DynFO [Meh13] and and the minimum can be expressed in first order.

Maintaining a minimal tree distance decomposition



What constitutes a bag w.r.t. to root bag $R = \{a, b\}$?

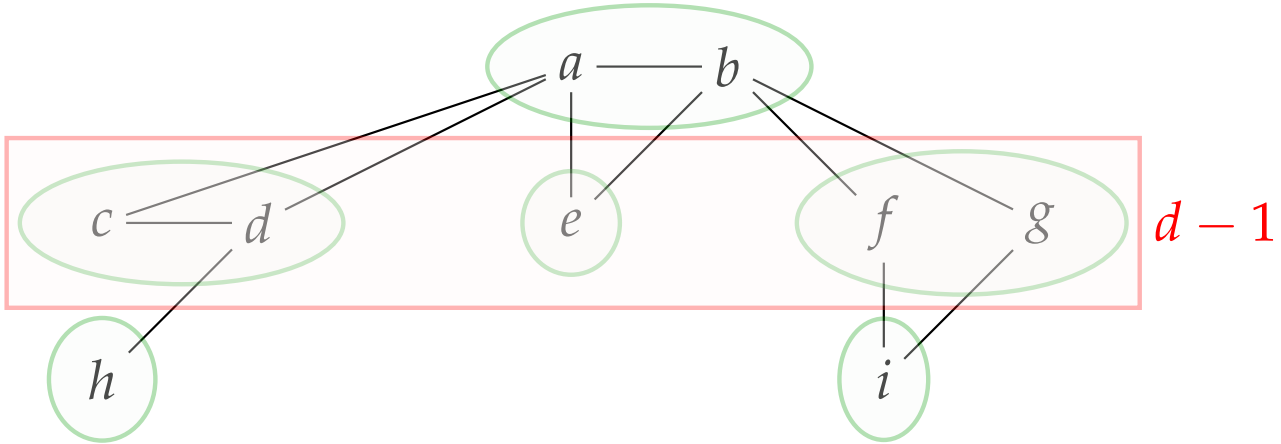
1. What is in the same component below some "bag" P?

$\text{inComp}_{R,P}(u, v) \Leftrightarrow u$ is reachable from v

in $G - P$ but from no vertex in R

e.g. $\text{inComp}_{\{a,b\},\{b,e\}}(f, g)$ holds

Maintaining a minimal tree distance decomposition



What constitutes a bag w.r.t. to root bag $R = \{a, b\}$?

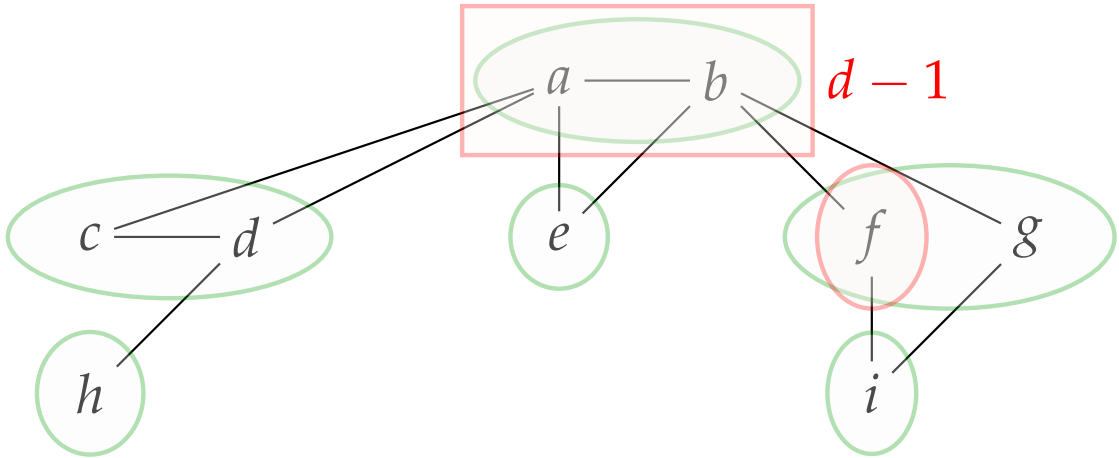
2. What is in one comp. below all "bags" with distance $d - 1$?

$$\text{inCompLvl}_{R,d}(u, v) \Leftrightarrow \forall B, |B| \leq k \wedge$$

$$v \in B \Rightarrow d(R, v) = d - 1 : \text{inComp}(R, B, u, v)$$

e.g. $\text{inCompLvl}_{\{a,b\},2}(h, i)$ does not hold

Maintaining a minimal tree distance decomposition



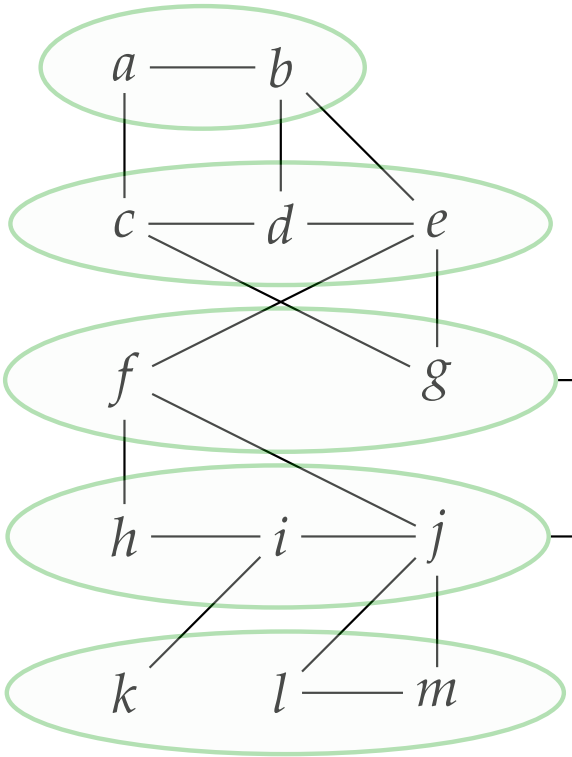
What constitutes a bag w.r.t. to root bag $R = \{a, b\}$?

3. Which "bags" are really bags?

$$\begin{aligned}\text{isBag}(R, B) \Leftrightarrow \exists d : (\forall v \in B : d(R, v) = d) \wedge \\ (\forall u, v \in B : \text{inCompLvl}(R, d, u, v)) \wedge \\ \forall u : (\exists v : v \in B, d(R, u) = d, \text{inCompLvl}(R, d, u, v)) \Rightarrow u \in B\end{aligned}$$

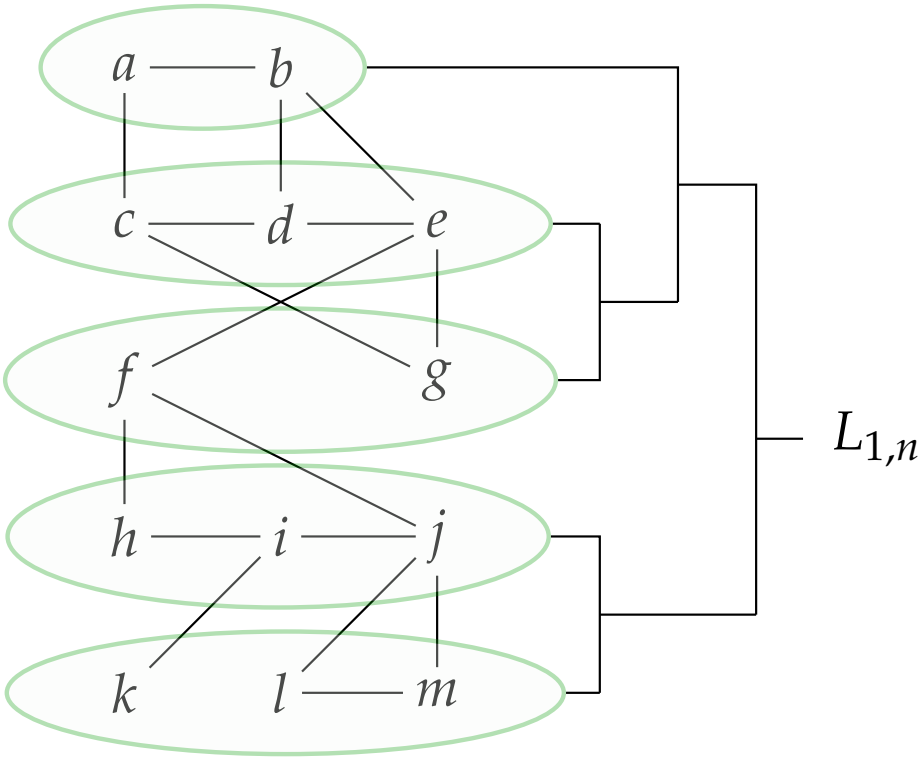
e.g. $\text{isBag}_{\{a,b\}}(\{f\})$ does not hold, because g is reachable from f in $G - \{a, b\}$

Canonization of PDDs in NC^1



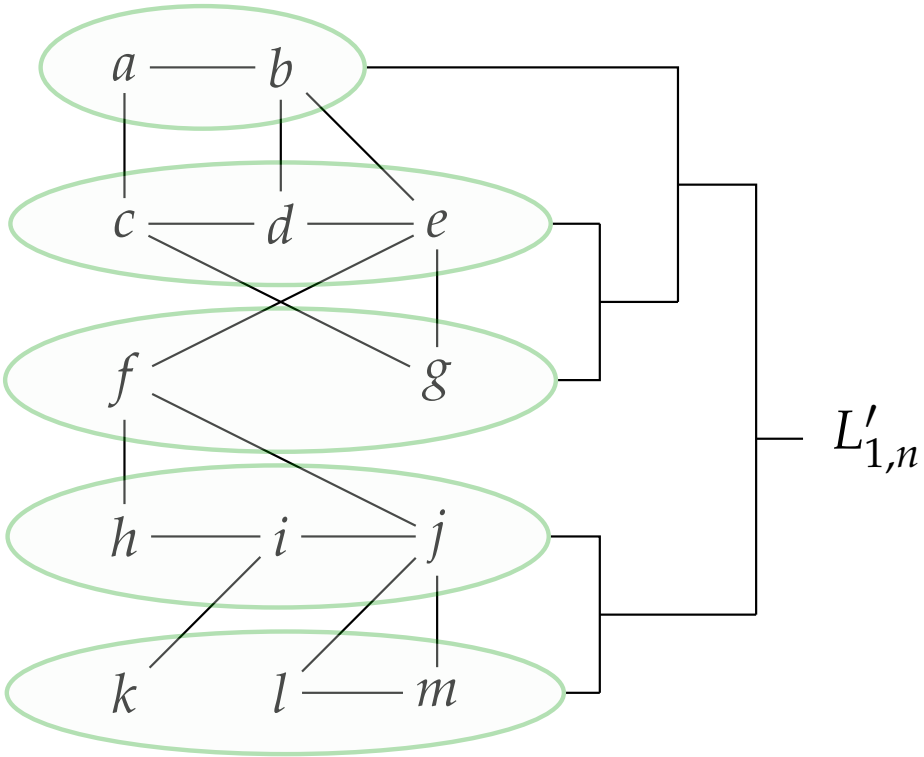
$$L_{i,i+1} = \{(\varphi, \psi) \mid \exists \text{lab.} \chi \text{ of } G[B_i \cup B_{i+1}] : \varphi = \chi|_{B_i}, \chi \text{ is minimal s.t. } \chi|_{B_{i+1}} = \psi\}$$

Canonization of PDDs in NC^1



$$L_{i,j} = \{(\varphi, \psi) \mid \exists (\varphi, \psi') \in L_{i,k}, (\varphi', \psi) \in L_{k+1,j} : (\psi', \varphi') \in L_{k,k+1}\}$$

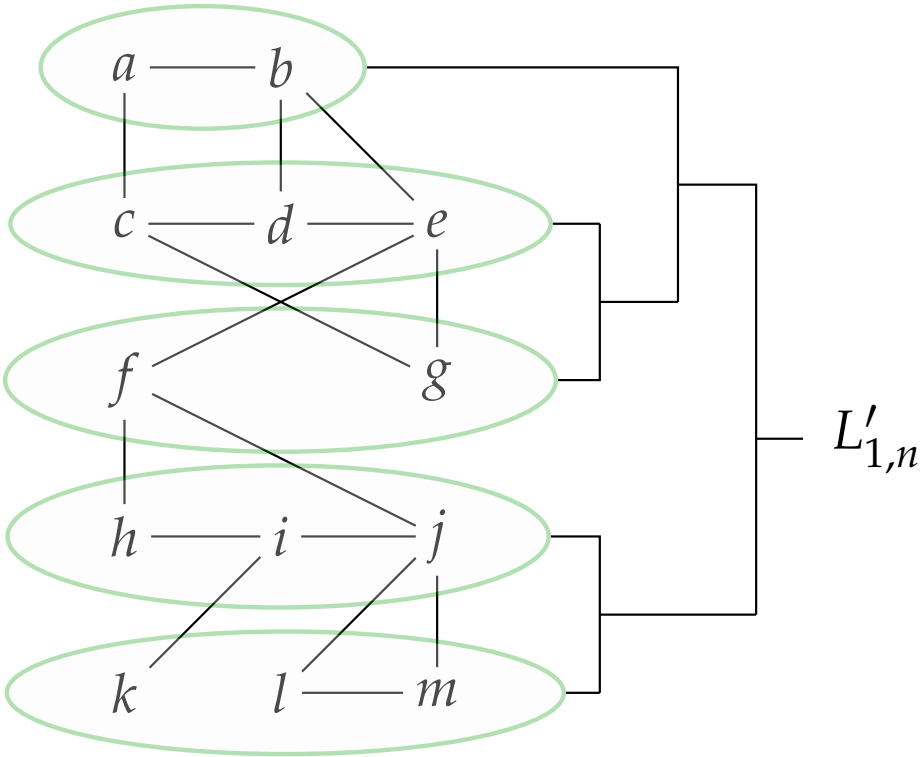
Canonization of PDDs in NC^1



$$L'_{i,j} = \left\{ (\varphi, \psi) \mid \exists (\varphi, \psi') \in L'_{i,k}, (\varphi', \psi) \in L'_{k+1,j} : (\psi', \varphi') \in L_{k,k+1} \right\}$$

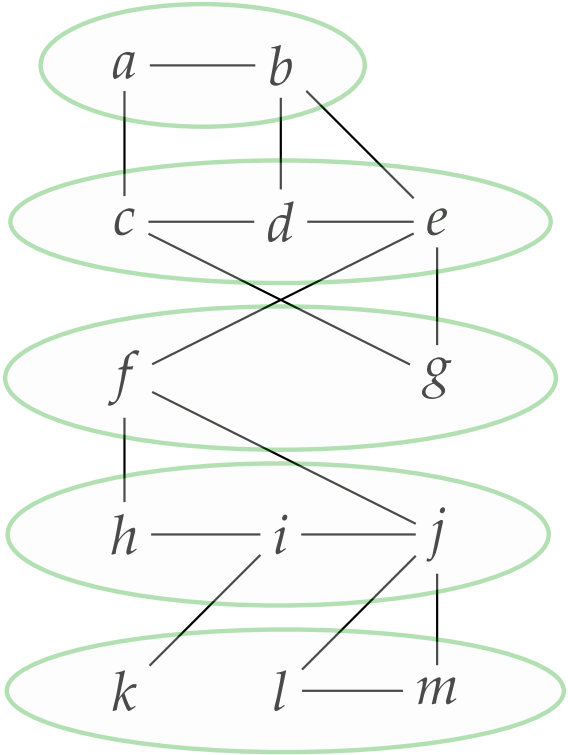
where $L'_{i,i+1}$ depends on $L_{i,n}$ and $L_{i+1,n}$

Canonization of PDDs in NC^1



Now we choose $\varphi_1 \in \mathrm{dom}(L'_{1,n})$
and *independently* one φ_i from each $L'_{1,i}(\varphi_1)$.

Canonization of PDDs in NC^1



Theorem

GraphIso for graphs of bounded path-distance width is in DynNC^1.

Maintaining a canonical k-tree-representation

- only (dis)connecting new simplicial vertices is allowed (instead of add/delete single edge)

Maintaining a canonical k-tree-representation

- only (dis)connecting new simplicial vertices is allowed (instead of add/delete single edge)
- k-tree canonization reduces to colored tree canonization with k colors [GG13; KK09]

Maintaining a canonical k-tree-representation

- only (dis)connecting new simplicial vertices is allowed (instead of add/delete single edge)
- k-tree canonization reduces to colored tree canonization with k colors [GG13; KK09]
- (dis)connecting a simplicial vertex changes at most 2 edges in this colored tree

Maintaining a canonical k-tree-representation

- only (dis)connecting new simplicial vertices is allowed (instead of add/delete single edge)
- k-tree canonization reduces to colored tree canonization with k colors [GG13; KK09]
- (dis)connecting a simplicial vertex changes at most 2 edges in this colored tree
- thus this gives us a bounded first order reduction to tree canonization [Ete98]

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization
  - However, as seen, many static results carry over to lower dynamic classes.

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization
  - However, as seen, many static results carry over to lower dynamic classes.
- Relevance of Dyn$\mathcal{C}$

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization
  - However, as seen, many static results carry over to lower dynamic classes.
- Relevance of Dyn$\mathcal{C}$
  - practical: unclear (except DynFO and DynTC^0)

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization
  - However, as seen, many static results carry over to lower dynamic classes.
- Relevance of Dyn$\mathcal{C}$
  - practical: unclear (except DynFO and DynTC^0)
  - research: focus on data structures

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization
  - However, as seen, many static results carry over to lower dynamic classes.
- Relevance of Dyn$\mathcal{C}$
  - practical: unclear (except DynFO and DynTC^0)
  - research: focus on data structures
  - research: raises new subquestions (PDD canonization in NC^1 is not interesting in the static setting)

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization
  - However, as seen, many static results carry over to lower dynamic classes.
- Relevance of Dyn$\mathcal{C}$
  - practical: unclear (except DynFO and DynTC^0)
  - research: focus on data structures
  - research: raises new subquestions (PDD canonization in NC^1 is not interesting in the static setting)
- Open questions regarding isomorphism

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization
  - However, as seen, many static results carry over to lower dynamic classes.
- Relevance of Dyn$\mathcal{C}$
  - practical: unclear (except DynFO and DynTC^0)
  - research: focus on data structures
  - research: raises new subquestions (PDD canonization in NC^1 is not interesting in the static setting)
- Open questions regarding isomorphism
  - Kinds of tree-like decompositions for which we can get below DynL?

Conclusions

- Few things are known (that is: explicitly stated) for dynamic complexity of graph isomorphism / canonization
  - However, as seen, many static results carry over to lower dynamic classes.
- Relevance of Dyn$\mathcal{C}$
  - practical: unclear (except DynFO and DynTC^0)
  - research: focus on data structures
  - research: raises new subquestions (PDD canonization in NC^1 is not interesting in the static setting)
- Open questions regarding isomorphism
  - Kinds of tree-like decompositions for which we can get below DynL?
  - For bounded treewidth in DynL? (and later in L?)

Bibliography I

[BDK12] A. Bouland, A. Dawar and E. Kopczyński. *On tractable parameterizations of graph isomorphism*. Parameterized and Exact Computation. Springer, 2012, pp. 218–230.

[Ete98] K. Etessami. *Dynamic tree isomorphism via first-order updates to a relational database*. Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. ACM. 1998, pp. 235–243.

[GG13] A. Gainer-Dewar and I. M. Gessel. *Counting unlabeled k-trees*. arXiv:1309.1429 (2013).

[Hes03] W. Hesse. *The dynamic complexity of transitive closure is in* DynTC^0. Theoretical Computer Science 296.3 (2003), pp. 473–485.

[KK09] J. Köbler and S. Kuhnert. *The isomorphism problem for k-trees is complete for logspace*. Mathematical Foundations of Computer Science 2009. Springer, 2009, pp. 537–548.

[Lin92] S. Lindell. *A logspace algorithm for tree canonization*. Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. ACM. 1992, pp. 400–404.

[Meh13] J. C. Mehta. *Dynamic Complexity of Planar 3-connected Graph Isomorphism*. arXiv:1312.2141 (2013).

Bibliography II

[MSVT94] P. B. Miltersen, S. Subramanian, J. S. Vitter and R. Tamassia. *Complexity models for incremental computation*. Theoretical Computer Science 130.1 (1994), pp. 203–236.

[NO06] J. Nešetřil and P. Ossona de Mendez. *Tree-depth, subgraph coloring and homomorphism bounds*. European Journal of Combinatorics 27.6 (2006), pp. 1022–1041.

[PI97] S. Patnaik and N. Immerman. *Dyn-FO: A Parallel, Dynamic Complexity Class*. Journal of Computer and System Sciences 55.2 (1997), pp. 199–209.