Distributed Relationship Schemes for Trees

Cyril Gavoille Arnaud Labourel

University of Bordeaux, France

December 17-21, 2007, Sendai

Distributed bounded distance oracle

Problem

*Assign a piece of information (***label***) to each node of a graph such that* **distance** *between any two nodes at distance no more than* **k** *can be retrieved from the labels associated with the two nodes, without any other source of information.*

Distributed bounded distance oracle

Problem

*Assign a piece of information (***label***) to each node of a graph such that* **distance** *between any two nodes at distance no more than* **k** *can be retrieved from the labels associated with the two nodes, without any other source of information.*

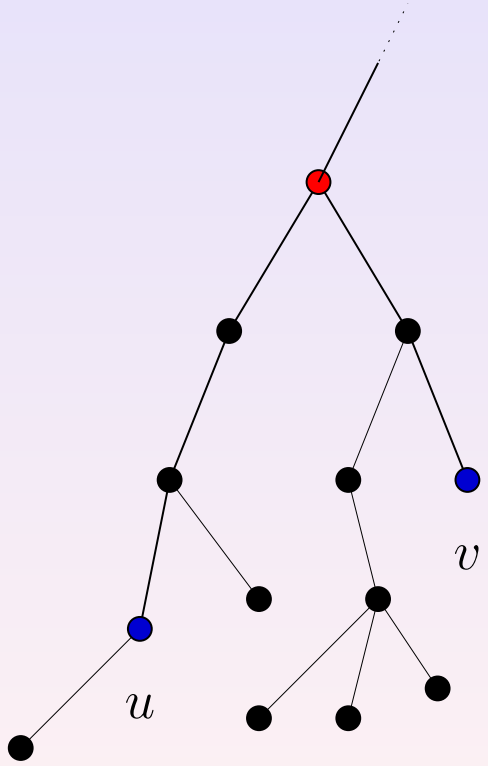
Goals

Minimize **label length** complexity, while maintaining a low time complexity for distance query, and for label assignment.

Bounded distance queries for trees

Definition

A pair (u, v) of nodes in a rooted tree is (k_1, k_2)-**related** if u and v are respectively at distance k_1 and k_2 from their **nca**.



(u, v) is $(3, 2)$-related

k-relationship scheme

Definition

A k-**relationship scheme** is a labeling scheme that can answer (from the labels) whether two nodes are (k_1, k_2)-related or not, for each $k_1, k_2 \leqslant k$.

E.g., 1-relationship schemes allow us to test:

k-relationship scheme

Definition

A k-**relationship scheme** is a labeling scheme that can answer (from the labels) whether two nodes are (k_1, k_2)-related or not, for each $k_1, k_2 \leqslant k$.

E.g., 1-relationship schemes allow us to test:

- identity: $(0, 0)$-related

k-relationship scheme

Definition

A k-**relationship scheme** is a labeling scheme that can answer (from the labels) whether two nodes are (k_1, k_2)-related or not, for each $k_1, k_2 \leqslant k$.

E.g., 1-relationship schemes allow us to test:

- identity: $(0, 0)$-related
- parent: $(0, 1)$-related

k-relationship scheme

Definition

A k-**relationship scheme** is a labeling scheme that can answer (from the labels) whether two nodes are (k_1, k_2)-related or not, for each $k_1, k_2 \leqslant k$.

E.g., 1-relationship schemes allow us to test:

- identity: $(0, 0)$-related
- parent: $(0, 1)$-related
- sibling: $(1, 1)$-related

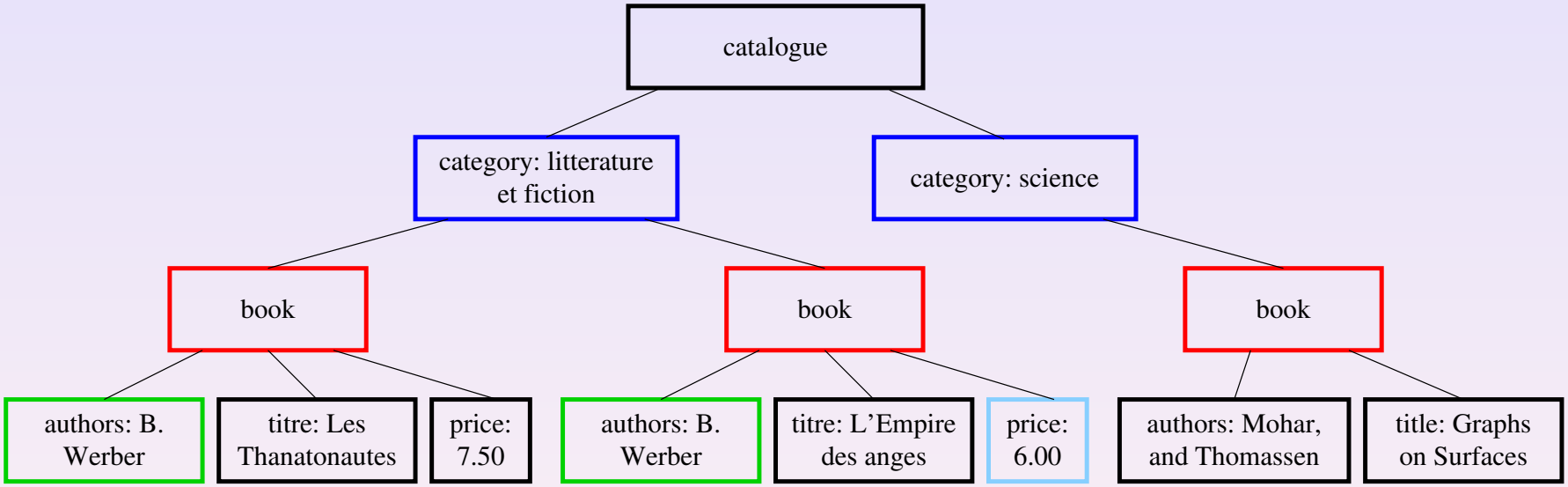
k-relationship scheme v.s. distances

Two nodes u, v are at **distance** $d \leqslant k$ iff there exists $i \in \{0, \ldots, k\}$ such that u, v are $(i, d-i)$-**related**.

$\Rightarrow$ a k-relationship scheme can be viewed as a distance oracle for distances $\leqslant k$ in trees.

Motivation: XML files & data-bases

XML files used for DB have a **tree** global structure.



XML search engine, and preprocessing

Wanted data-structures supporting structured queries like, search for a `book` of a given `category`, `author` and `price`.

XML search engine, and preprocessing

Wanted data-structures supporting structured queries like, search for a `book` of a given `category`, `author` and `price`.

Solution [Abiteboule et al. - SODA '01]

- Add to the XML file a big hash table containing all items.
- Associated with each entry w, the precomputed label(u) of each node u of the tree containing item w.
- Structured queries can be expressed as testing ancestry relationship, like k-relationship.

XML search engine, and preprocessing

Wanted data-structures supporting structured queries like, search for a `book` of a given `category`, `author` and `price`.

Solution [Abiteboule et al. - SODA '01]

- Add to the XML file a big hash table containing all items.
- Associated with each entry w, the precomputed label(u) of each node u of the tree containing item w.
- Structured queries can be expressed as testing ancestry relationship, like k-relationship.

Note: **one** byte saved in the label length does matter! each item ($\sim 10^8$) of the DB has to store a label.

Observation

$A_u[d] =$ ancestor of u at distance d

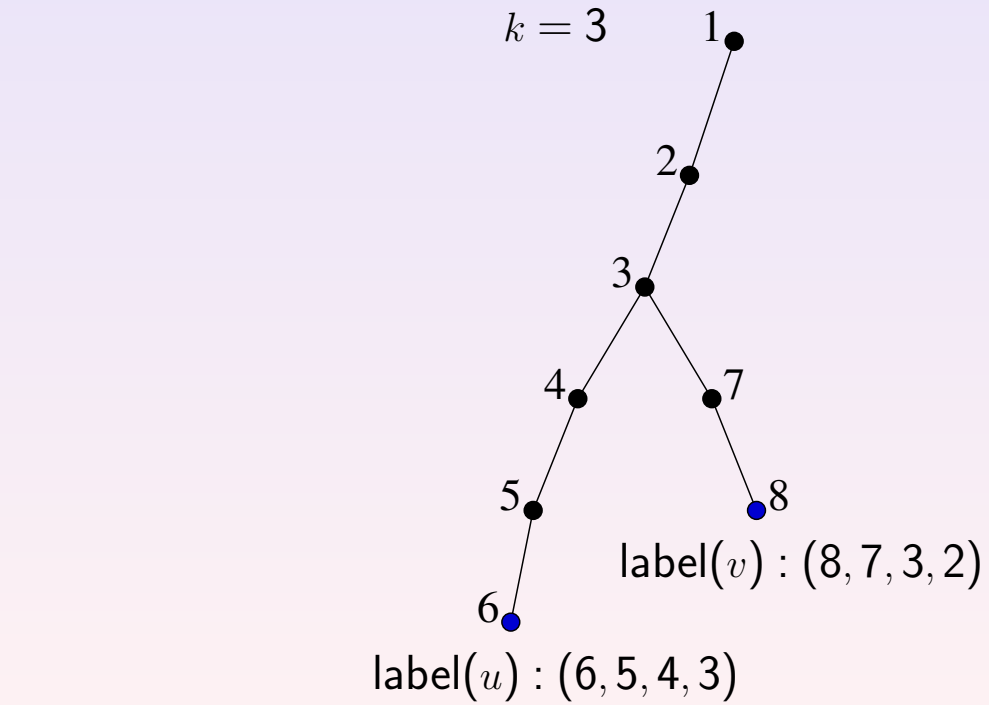
Fact

(u, v) is (k_1, k_2)-related iff $A_u[k_1] = A_v[k_2]$ and $A_u[k_1 - 1] \neq A_v[k_2 - 1]$.

Trivial solution

Version 1.0: $(k+1)\lceil \log n \rceil$-bit labels

- Associated with each node of the tree an **identifier**, a unique integer in $[0, n)$, $n =$ number of nodes.
- The **label** of u is just the sequence of the identifiers of $A_u[0], A_u[1], \cdots, A_u[k]$.



Best previous bounds

[Astrup, Bille, and Rauhe - SODA '03]

1. There is a k-relationship scheme with labels of $\log n + O(k^2 \cdot \log(k \log n))$ bits.

Best previous bounds

[Astrup, Bille, and Rauhe - SODA '03]

1. There is a k-relationship scheme with labels of $\log n + O(k^2 \cdot \log(k \log n))$ bits.
2. Every 1-relationship scheme requires labels of $\log n + \log \log n + \Omega(1)$ bits in the worst-case.

Best previous bounds

[Astrup, Bille, and Rauhe - SODA '03]

1. There is a k-relationship scheme with labels of $\log n + O(k^2 \cdot \log(k \log n))$ bits.
2. Every 1-relationship scheme requires labels of $\log n + \log \log n + \Omega(1)$ bits in the worst-case.

Note: `parent` query can be done with $\log n + O(\log^* n)$ bit labels [Alstrup et al. - FOCS '01], so `sibling` query costs!

Our results

Theorem

There is a k-relationship scheme using labels of $\log n + O(k \cdot \log(k \log(n/k)))$ bits.

Second order term	:	$k^2 \cdot \log(k \log n)$	$\rightarrow$	$k \cdot \log(k \log(n/k))$
For $k = 1$	:	$5 \log \log n$	$\rightarrow$	$2 \log \log n$

Remind: constants do matter!

Our scheme

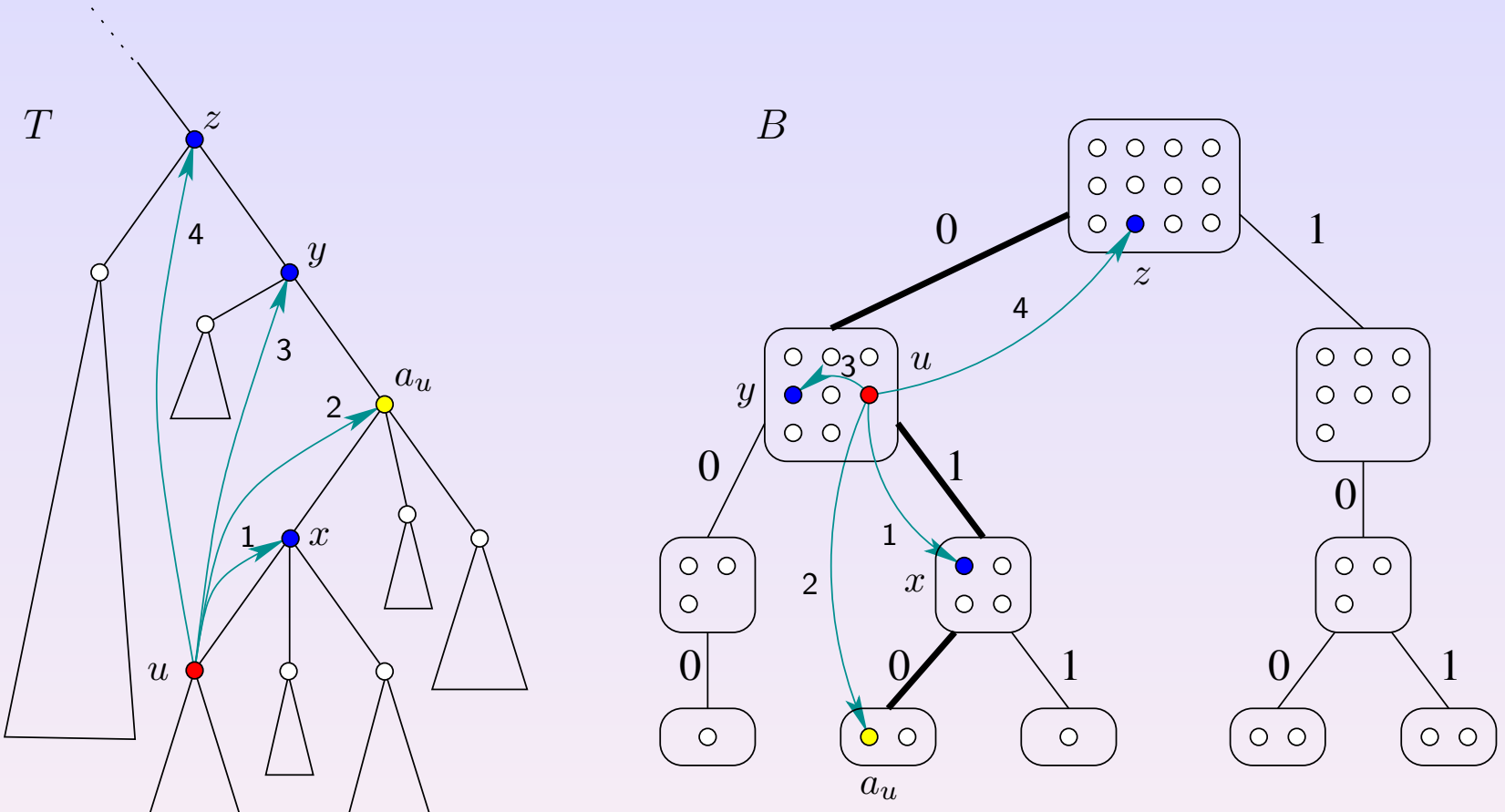
- Principle (similar to version 1.0): to store in the label of u **some** identifiers of $A_u[0], A_u[1], \cdots, A_u[k]$.
- Difficulty is in the selection of the identifiers so that they can be compressed. For that, we need a **specific** decomposition of the tree.

Key lemma

Lemma (k-ancestry decomposition)

Every tree T with n nodes has a node partition such that:

- *parts of the partition are nodes of a rooted* **binary tree** *B of depth $\leqslant \log(n/k)$;*
- *each part contains $\leqslant (k+1)\log(n/k)$ nodes of T; and*
- *for every $u \in V(T)$, the parts of its ancestors $A_u[0], A_u[1], \cdots, A_u[k]$ belongs to a path from the root to a leaf of B.*


T
z
y
a_u
x
u
4
3
2
1
B
z
y
u
x
a_u
0
1
4
3
2
1
0
1
0
1
0
1
0
1
0
1

How to get such a decomposition?

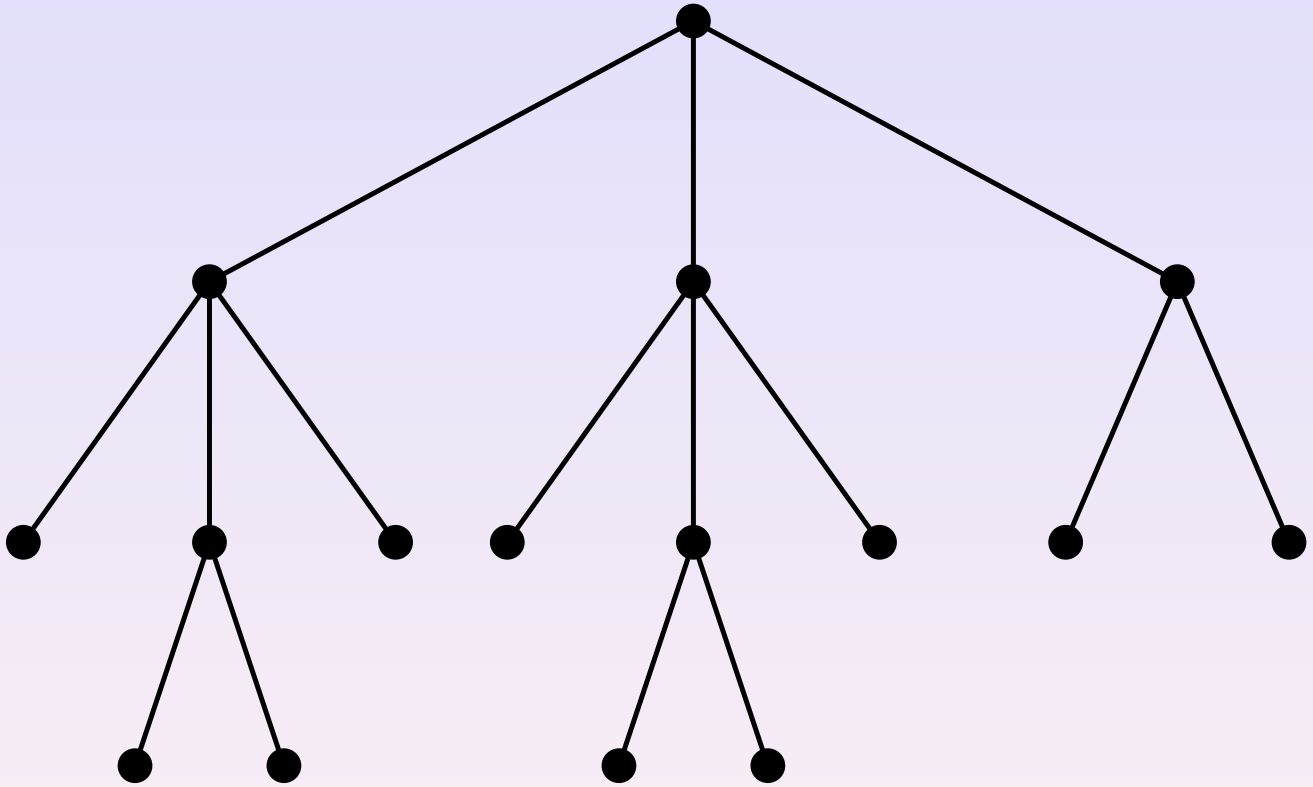
Step 1: Connect each node of T to its k closest ancestors.

$\Rightarrow$ We get an augmented graph T_k that is k-tree.

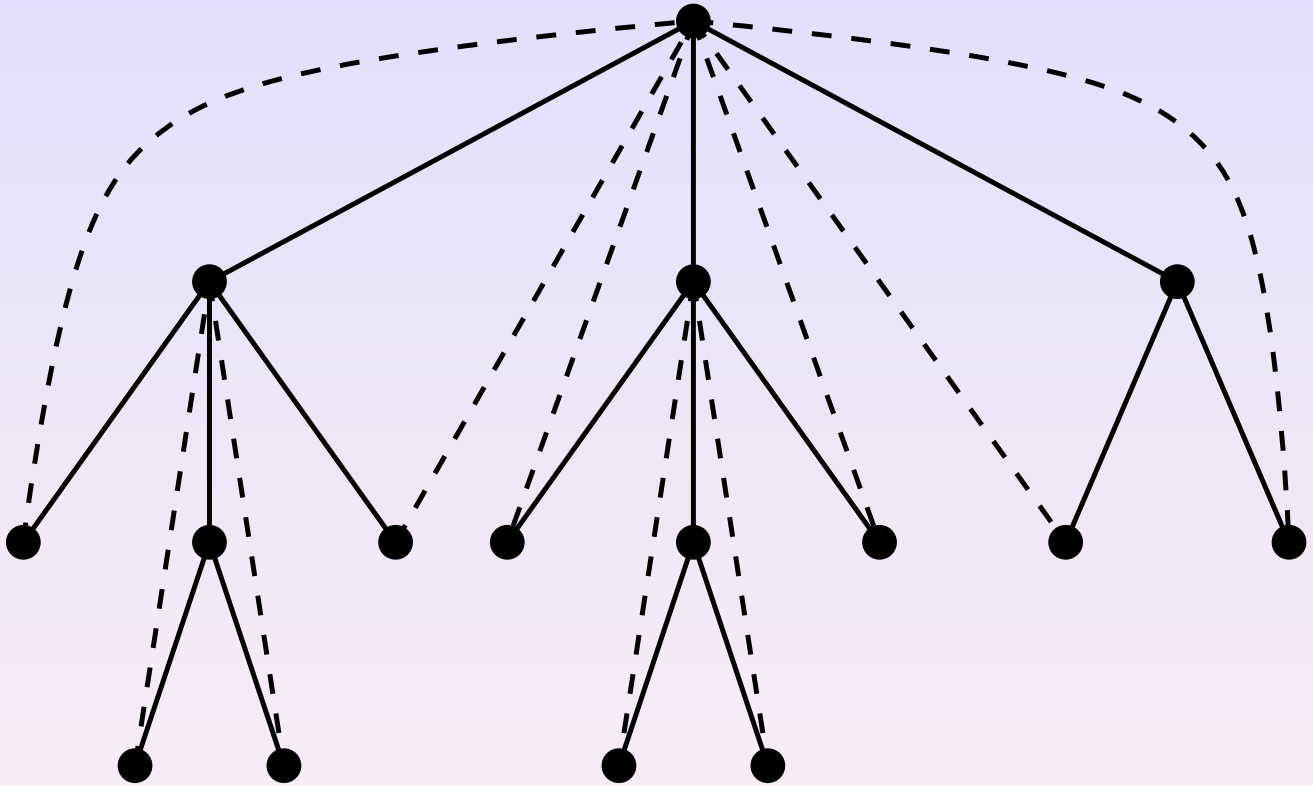
Step 2: Cut T_k recursively in two components of equal size, thanks to separators that cliques of size $k+1$.

$\Rightarrow$ We get a node partition of T, and a binary tree B where parts are composed of $\log(n/k)$ separators of size $k+1$.

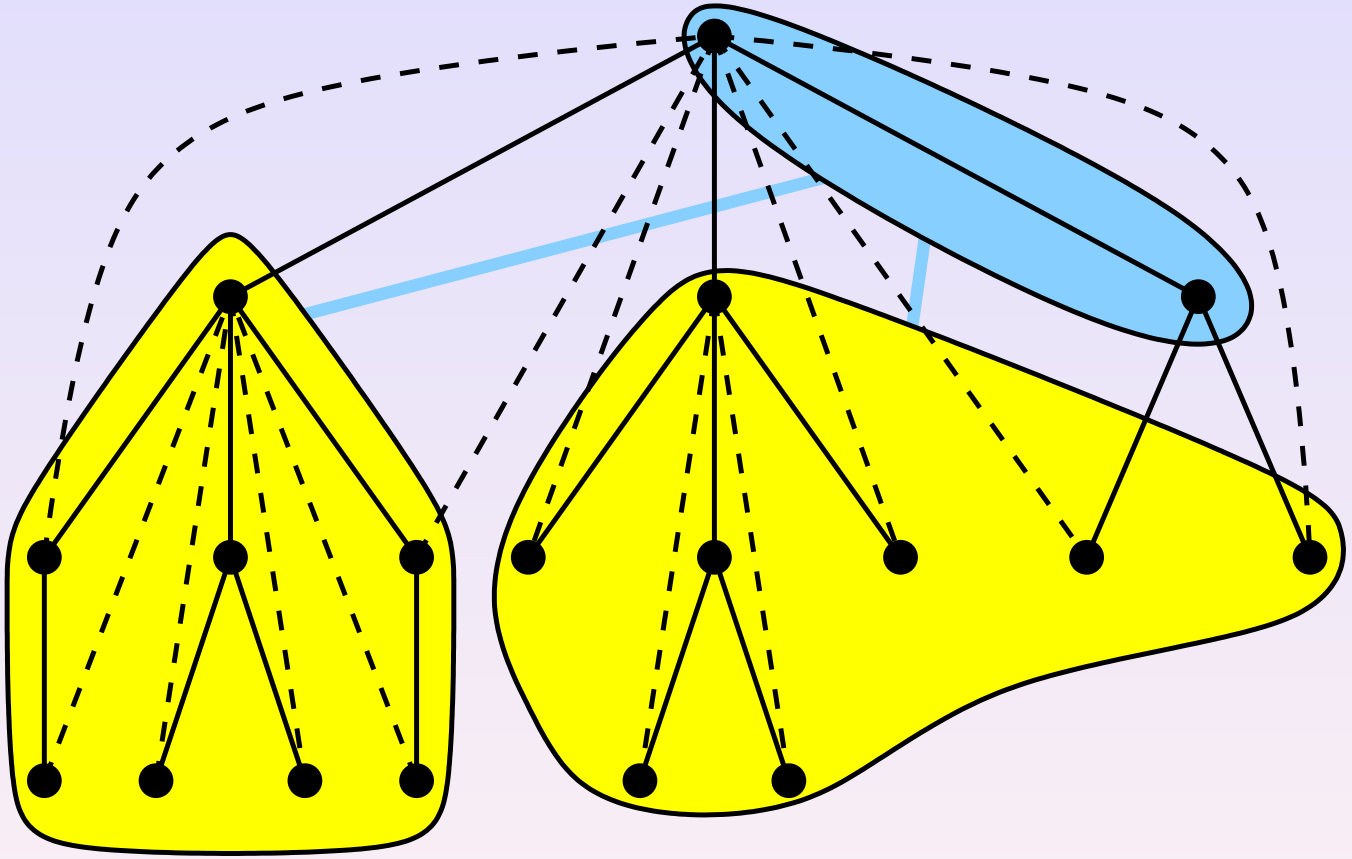
How to get such a decomposition?



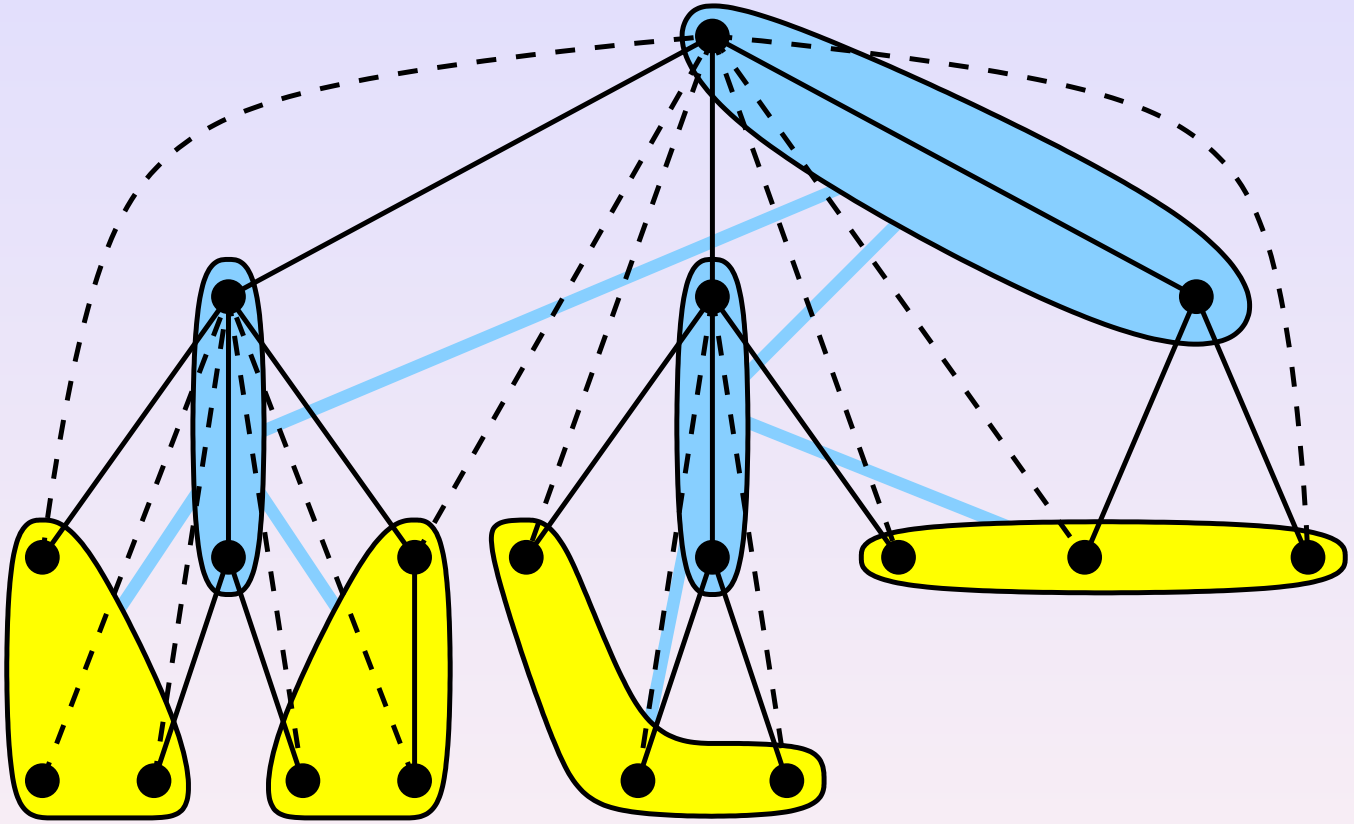
How to get such a decomposition?



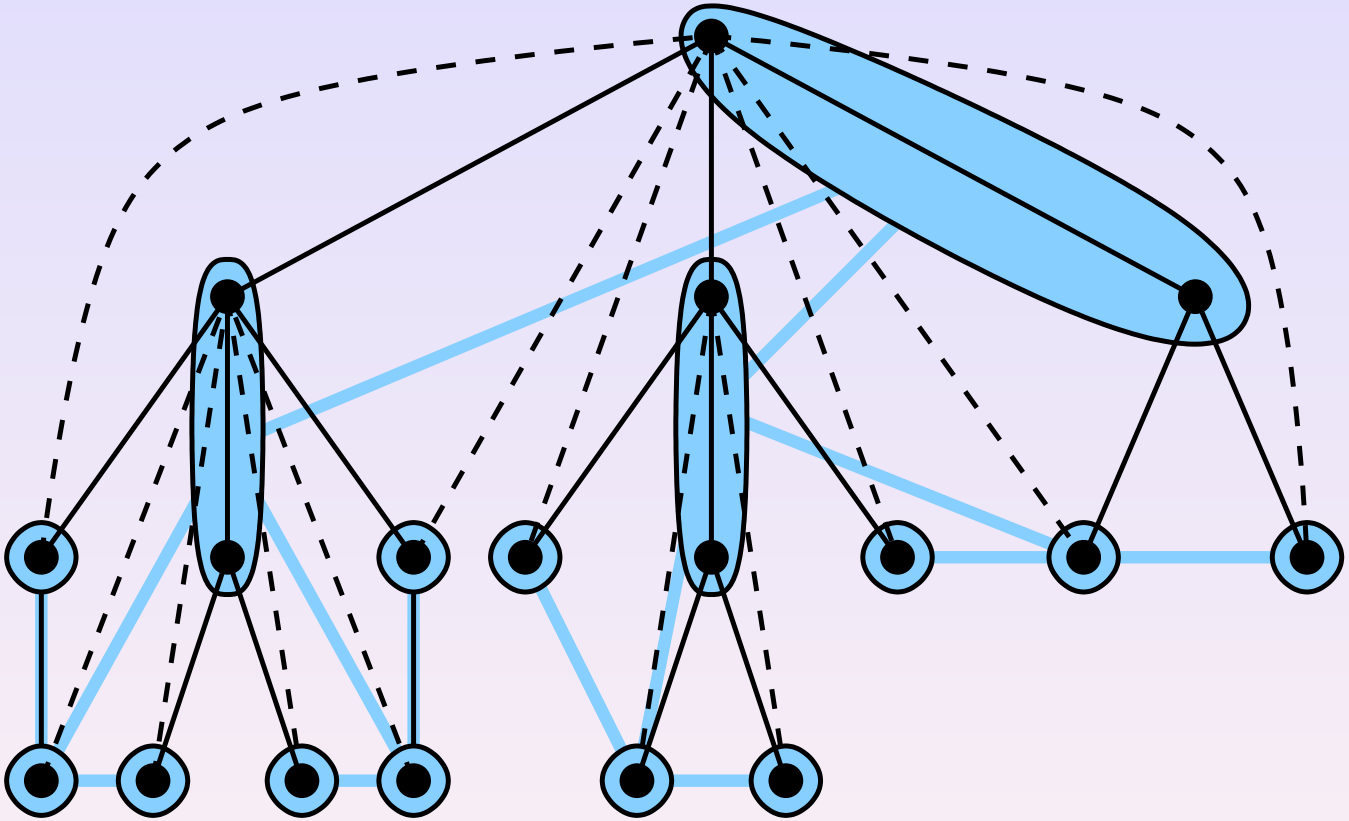
How to get such a decomposition?



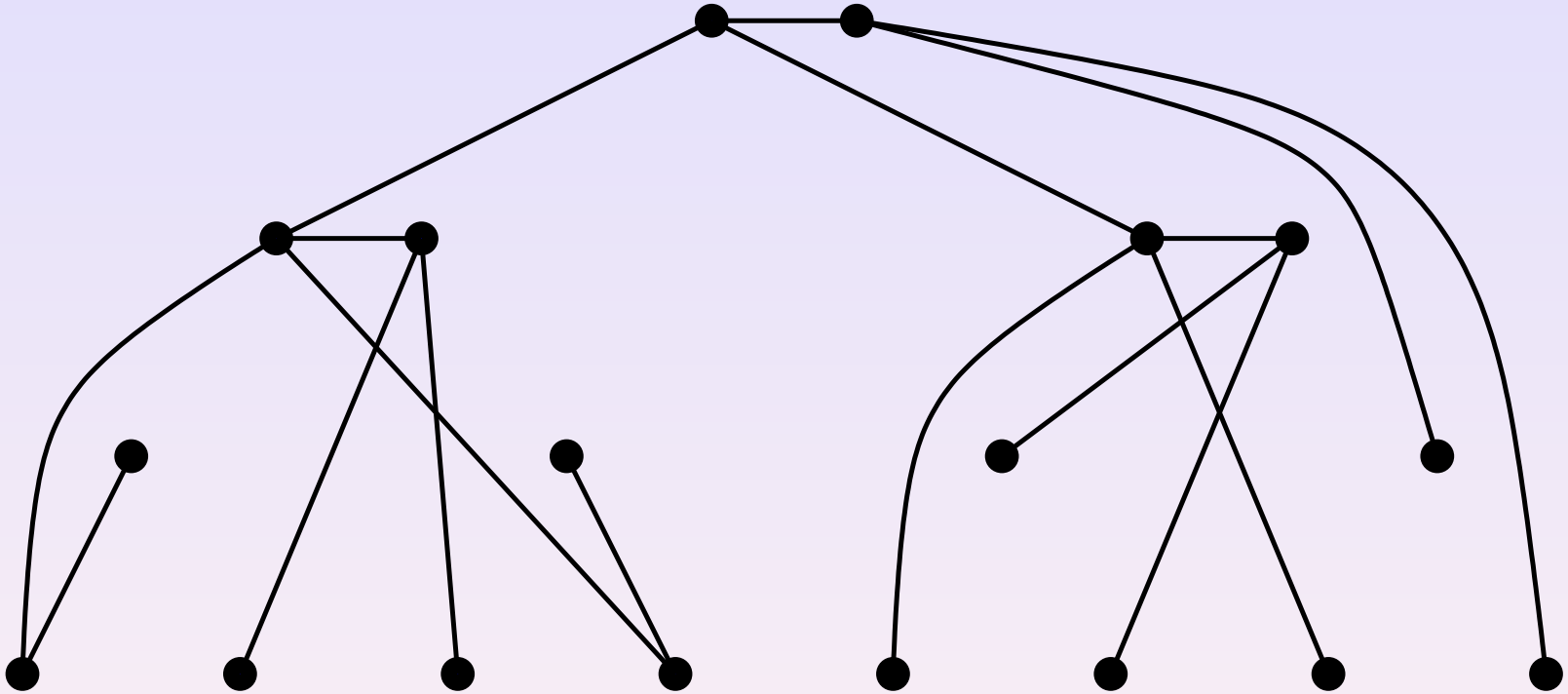
How to get such a decomposition?



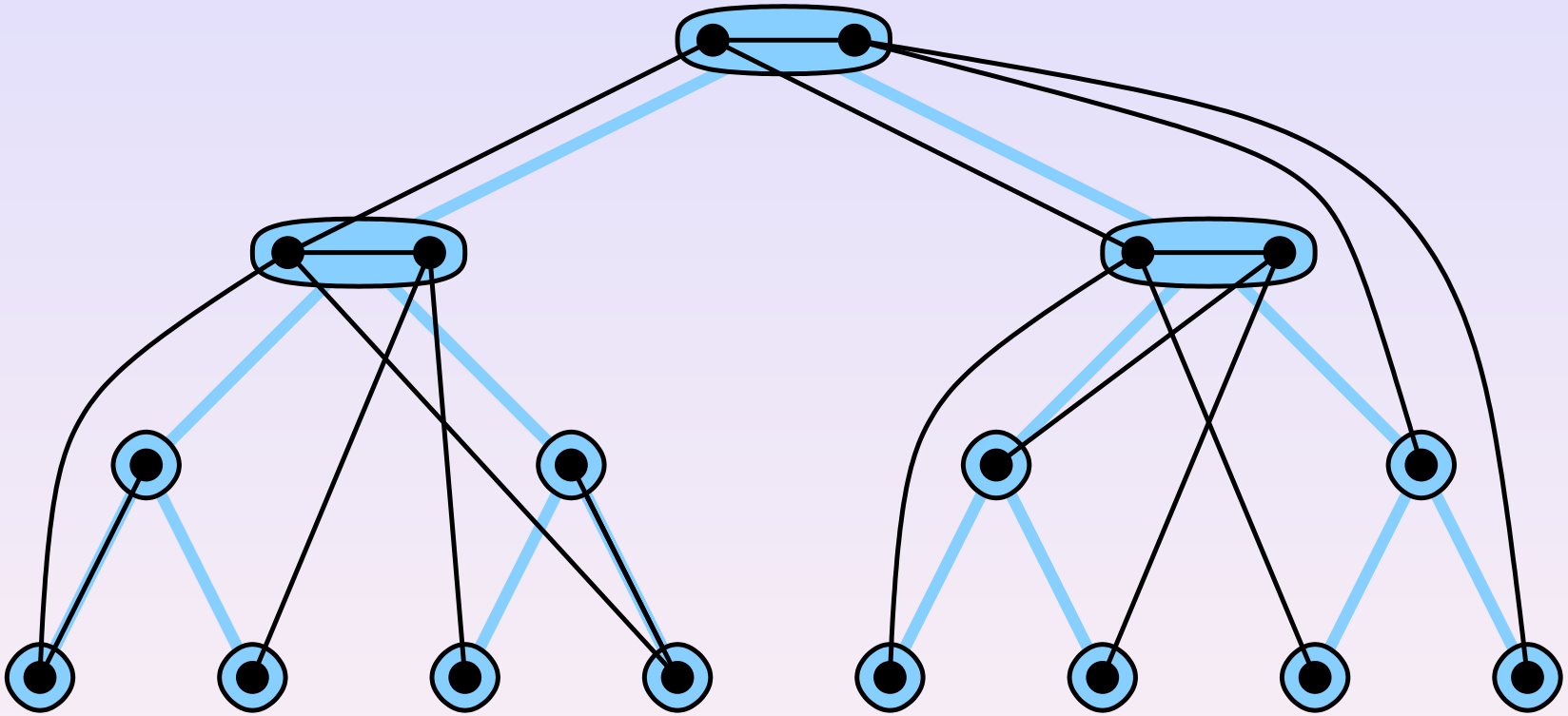
How to get such a decomposition?



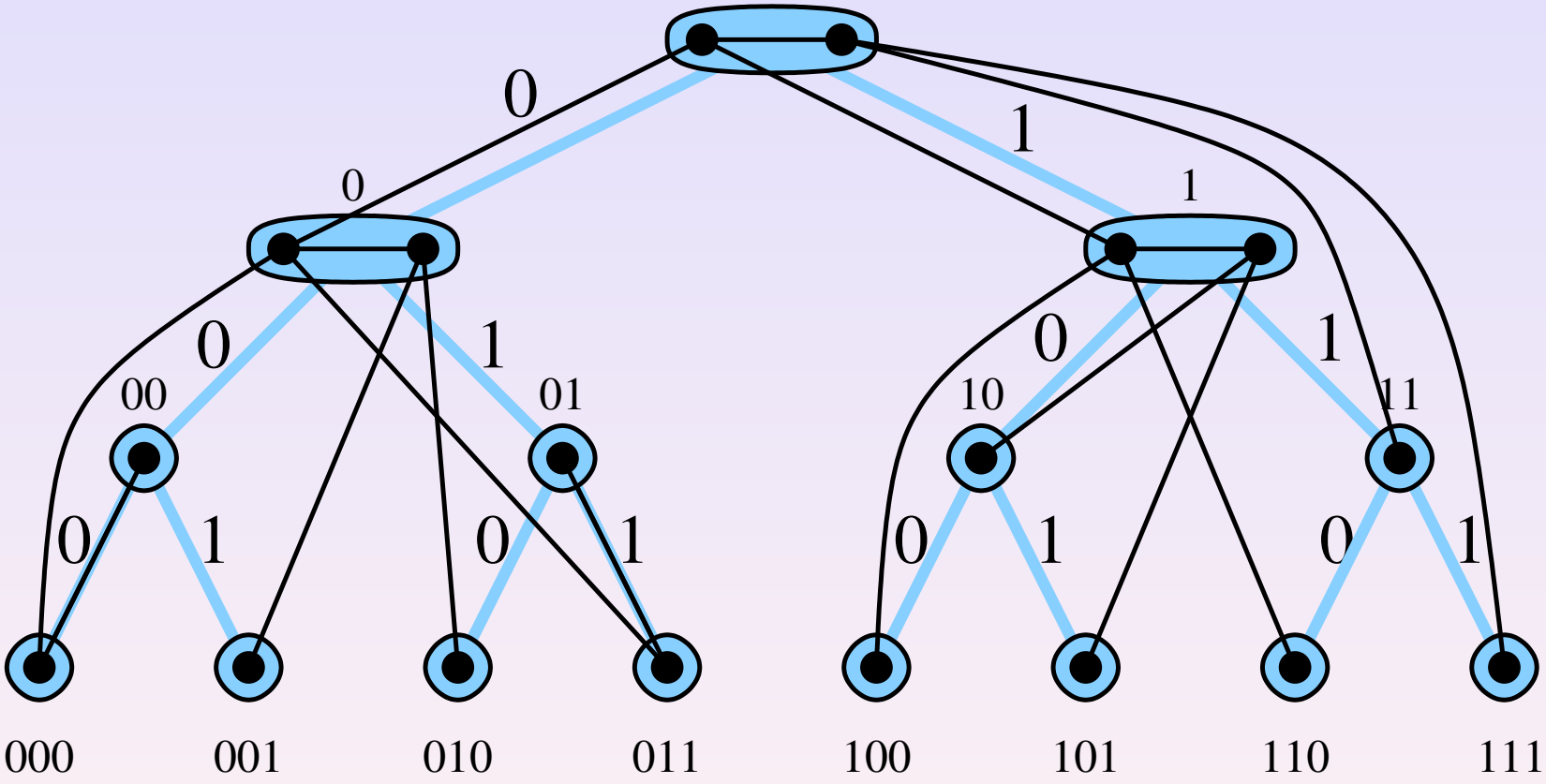
... and the decomposition of the tree



... and the decomposition of the tree



... and the decomposition of the tree



Identifiers and labels

Identifier of u: path from the root of B to the part of u, plus the position of u in its part.

Label of u: path w_u [$\leqslant \log(n/k)$ bits] from the root of B to the deepest part of an ancestor $A_u[0], A_u[1], \cdots, A_u[k]$, PLUS the depth of the part and the position of each of them [$O(k \cdot \log(k \log(n/k))$ bits].

Identifiers and labels

Identifier of u: path from the root of B to the part of u, plus the position of u in its part.

Label of u: path w_u [$\leqslant \log(n/k)$ bits] from the root of B to the deepest part of an ancestor $A_u[0], A_u[1], \cdots, A_u[k]$, PLUS the depth of the part and the position of each of them [$O(k \cdot \log(k \log(n/k))$ bits].

From label(u) one can extract **all** the identifiers of the nodes $A_u[0], A_u[1], \cdots, A_u[k]$ because the path of the part containing $A_u[i]$ is a prefix of w_u of length the depth of the part of $A_u[i]$. □

Open problems

Our scheme implies that distances in $o(\log n / \log\log n)$ can be computed with $\log n + o(\log n)$ bits per node.
One may ask:

- Design a labeling scheme with $\log n + o(\log n)$-bit labels for larger distances, say up to $c \log n$ for small constant $c \leqslant 1$.
- Design a labeling scheme with $\log n + o(\log n)$-bit labels for bounded distance in bounded **tree-width** graphs.
- Maintain dynamically our scheme with near optimal labels, say $O(\log n)$, and with low (amortized) update cost.

Thank You
for your attention