Compact and Distributed Data Structures

Cyril Gavoille

(LaBRI, University of Bordeaux)

44th Ecole de Printemps en Informatique Théorique (EPIT)

- Distributed Computing -

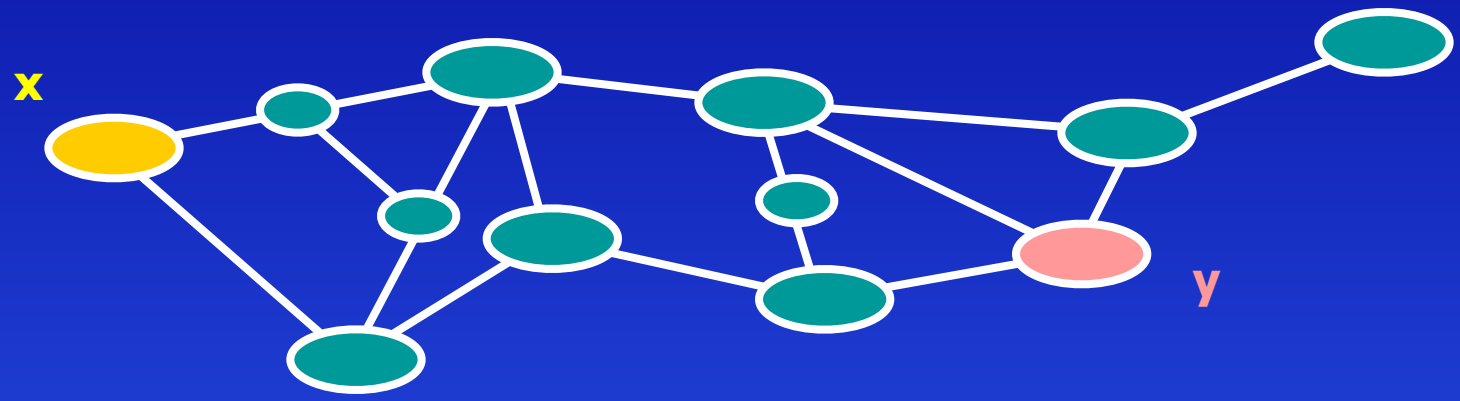
Île de Porquerolles, May 15-19, 2017

Information & Locality

Understanding what information are needed to achieve a computational task is a central question not only in DC (eg., data structure theory, communication complexity,…)

The ultimate goal in **Labeling Schemes** is to understand how localized and how much information are required to solve a given task on a network.

Task1: Routing in a physical network



Routing query: next hop to go from x to y?

- pre-processing to compute routing information
- a node x stores only routing information involving x

⇒ distributed data structure

Task2: Ancestry in rooted trees

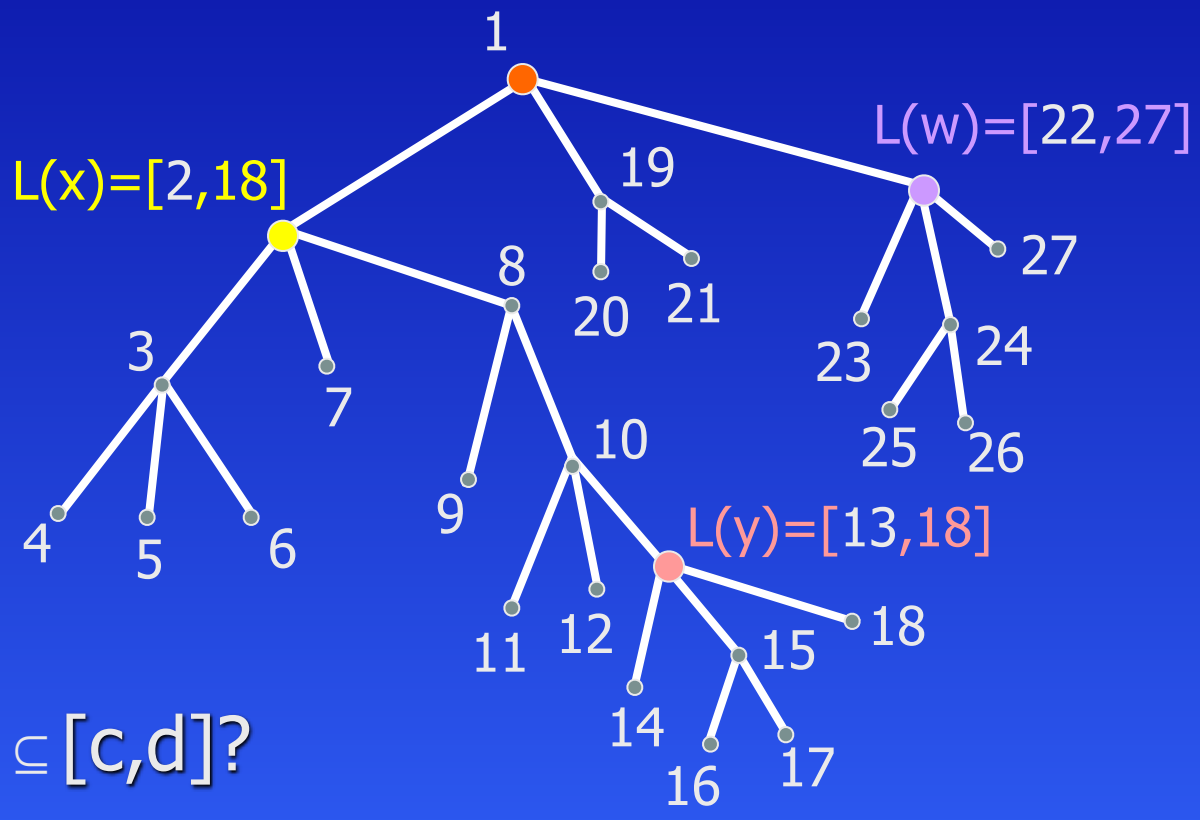
Motivation: [Abiteboul,Kaplan,Milo '01]

The <TAG> … </TAG> structure of a huge XML data-base is a rooted tree. Some queries are **ancestry** relations in this tree.

Ex: Is <"distributed computing"> descendant of <booktitle>?

Use compact index for fast query XML search engine. Here the constants do matter. Saving 1 byte of fast memory on each entry of the index table is important. Here n is large, ~ 10^9.

Folklore solution: DFS labelling



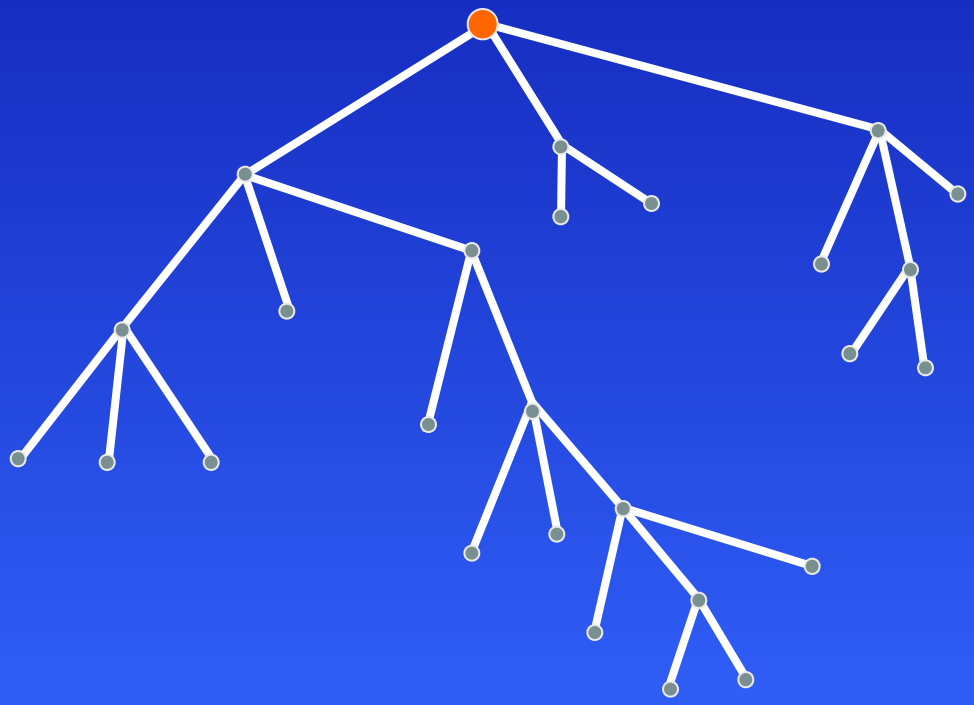
$[a,b] \subseteq [c,d]$?

$\Rightarrow$ 2logn bit labels

Best solution: $\log n + \theta(\log\log n)$ bit labels

[Alstrup,Rauhe – Siam J.Comp. ’06]

[Fraigniaud,Korman – STOC’10]



A distributed data structure

- Get the labels of nodes involved in the query
- Compute/decode the answer from the labels
- No other source of information is required

Some labelling schemes

- Adjacency
- Distance (exact or approximate)
- First edge on a (near) shortest path
- Ancestry, parent, nca, sibling relations in trees
- Edge/vertex connectivity, flow
- Proof labelling systems
- …

Agenda

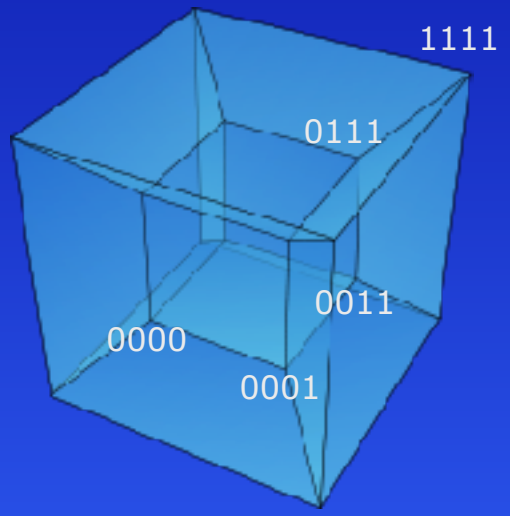
1. Distance Labelling in General Graphs
2. Distance Labelling in Trees
3. Routing in Trees
4. Nearest Common Ancestor Labelling
5. Forbidden-Set Labelling
6. Distance in Planar Graphs
7. Distance in Minor-Free Graphs

Agenda

1. **Distance Labelling in General Graphs**
2. Distance Labelling in Trees
3. Routing in Trees
4. Nearest Common Ancestor Labelling
5. Forbidden-Set Labelling
6. Distance in Planar Graphs
7. Distance in Minor-Free Graphs

The Distance Labelling Problem

Given a graph, find a labelling of its nodes such that the distance between any two nodes can be computed by inspecting only their labels.



Subject to:

- label the nodes of every graph of a family (scheme),
- using short labels (size measured in bits), and
- with a fast distance decoder (algorithm)

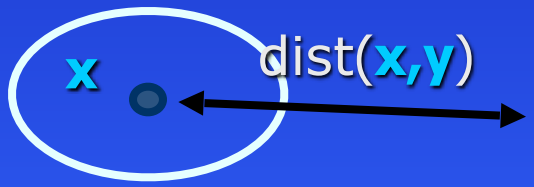
Motivation
[Peleg '99]

If a short label (say of poly-logarithmic size) can be added to the address of the destination, then routing to any destination can be done without routing tables and with a "limited" number of messages.

message header=hop-count

Motivation
[Peleg '99]

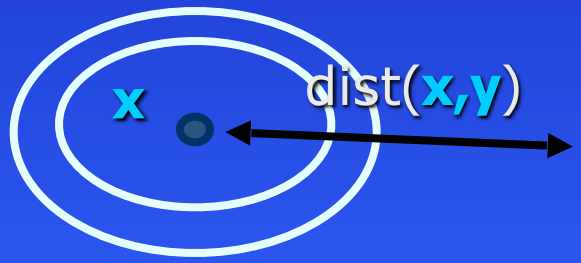
If a short label (say of poly-logarithmic size) can be added to the address of the destination, then routing to any destination can be done without routing tables and with a "limited" number of messages.



message header=hop-count

Motivation
[Peleg '99]

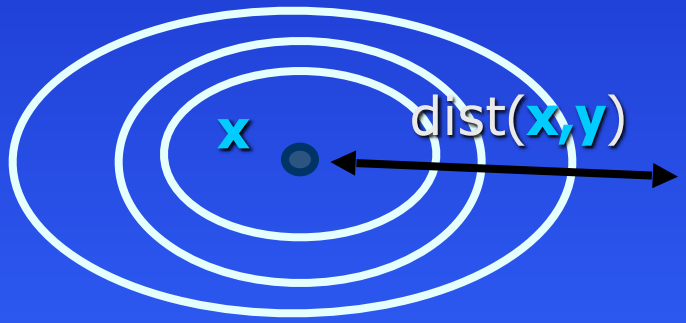
If a short label (say of poly-logarithmic size) can be added to the address of the destination, then routing to any destination can be done without routing tables and with a "limited" number of messages.



message header=hop-count

Motivation
[Peleg '99]

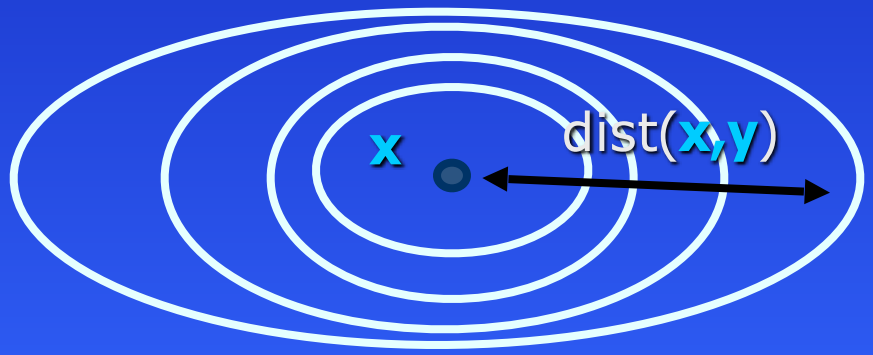
If a short label (say of poly-logarithmic size) can be added to the address of the destination, then routing to any destination can be done without routing tables and with a "limited" number of messages.



message header=hop-count

Motivation
[Peleg '99]

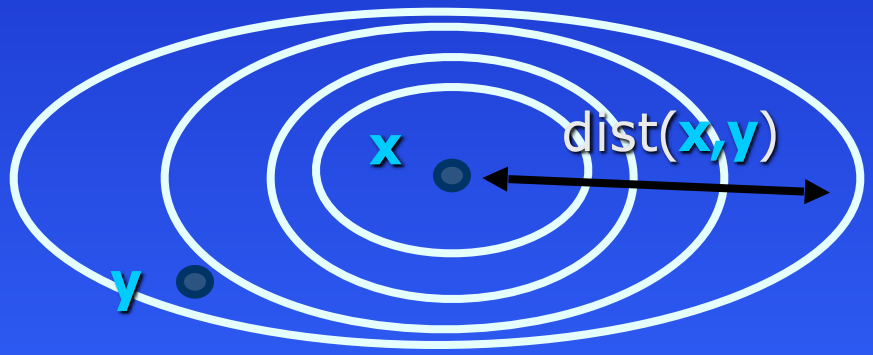
If a short label (say of poly-logarithmic size) can be added to the address of the destination, then routing to any destination can be done without routing tables and with a "limited" number of messages.



message header=hop-count

Motivation
[Peleg '99]

If a short label (say of poly-logarithmic size) can be added to the address of the destination, then routing to any destination can be done without routing tables and with a "limited" number of messages.



message header=hop-count

Label Size:
a trivial upper bound

There is a labelling scheme using labels of $O(n \log n)$ bits for every (unweighted) graph G with n nodes, and constant time decoding.

$$L_G(i) = (i, [\text{dist}(i,1), \ldots, \text{dist}(i,j), \ldots, \text{dist}(i,n)])$$

➟ distance vector

Label Size:
a trivial lower bound

No labelling scheme can guarantee labels of less than $0.5n$ bits for all n-node graphs (whatever the distance decoder complexity is)

Proof. The sequence $\langle L_G(1),\ldots,L_G(n)\rangle$ and the decoder $\delta(.,.)$ is a representation of G on $n\cdot k+O(1)$ bits if each label has size k: i adjacent to j iff $\delta(L_G(i),L_G(j))=1$.

$$n\cdot k + O(1) \geq \log_2(\#\text{graphs}(n))=n\cdot(n-1)/2$$

∎

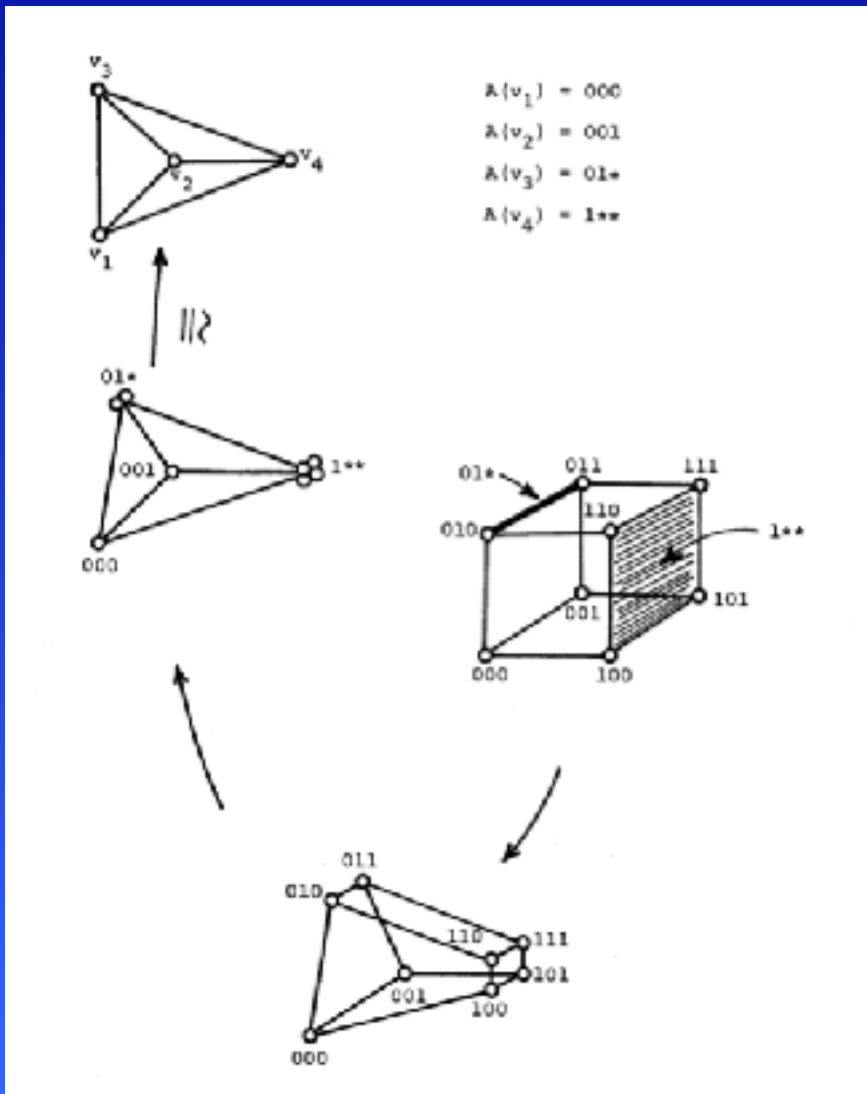
Squashed Cube Dimension
[Graham,Pollack '71]

Labeling: word over {0,1,*}
Decoder: Hamming distance
(where *=don't care)
(graphs must be connected)

$SC_{dim}(G) \geq \max\{n^+, n^-\}$

$n^{+/-}$=#positive/negative eigen-values of the distance matrix of G

$SC_{dim}(K_n)=n-1$



Squashed Cube Dimension
[Winkler '83]

Theorem. Every connected n-node graph has squashed cube dimension at most n-1.

Therefore, for the family of all connected n-node graphs:

Label size: $O(n)$ bits, in fact $n\log_2 3 \sim 1.58n$ bits

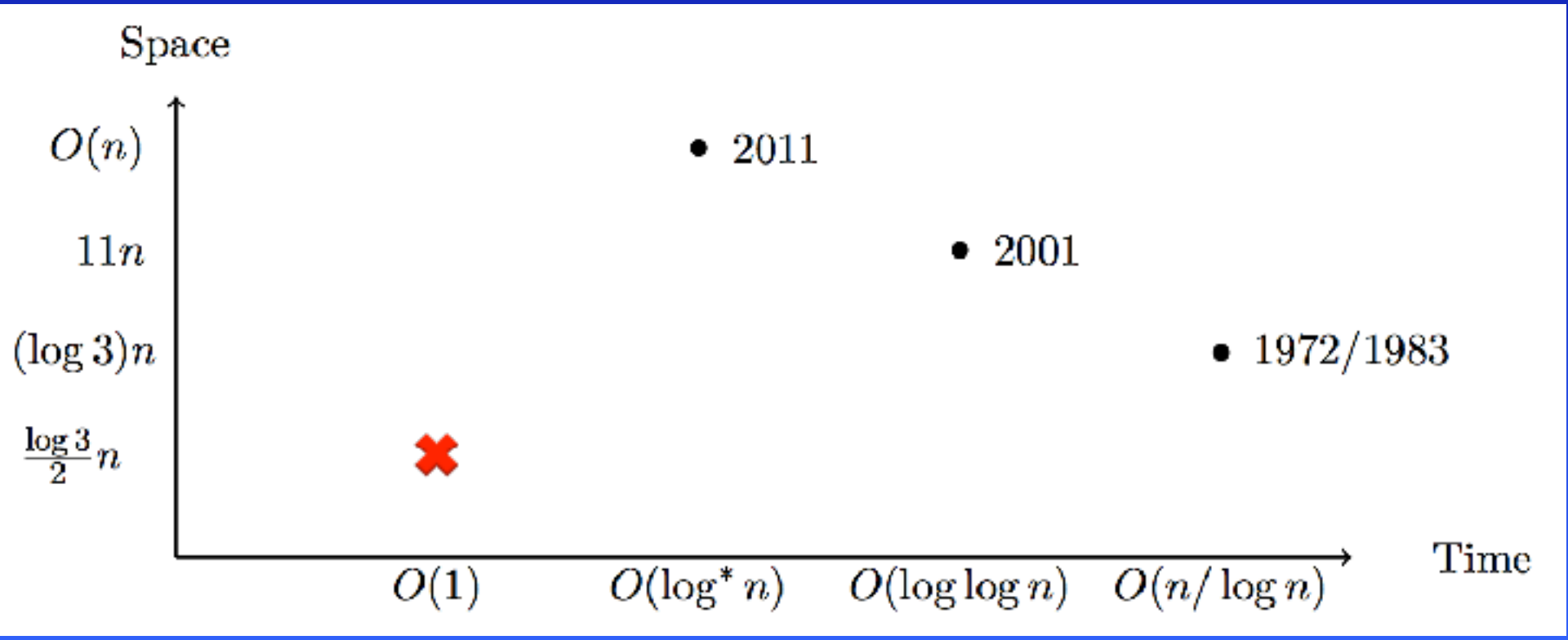
Decoding time: $O(n/\log n)$ in the RAM model

Rem: all graphs = connected graphs + $O(\log n)$ bits

Current best solution

Label size: $n(\log_2 3)/2 \sim 0.793n$ bits

Decoding time: O(1)



Agenda

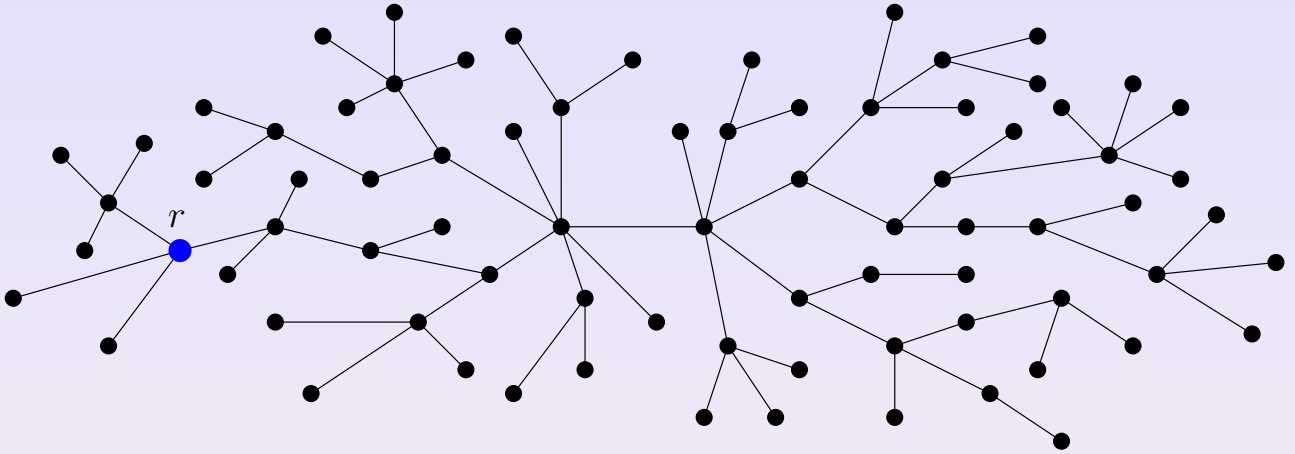
1. Distance Labelling in General Graphs
2. **Distance Labelling in Trees**
3. Routing in Trees
4. Nearest Common Ancestor Labelling
5. Forbidden-Set Labelling
6. Distance in Planar Graphs
7. Distance in Minor-Free Graphs

Un oracle simple pour les arbres

Idée générale : compresser et « localiser » les informations

Un oracle simple pour les arbres

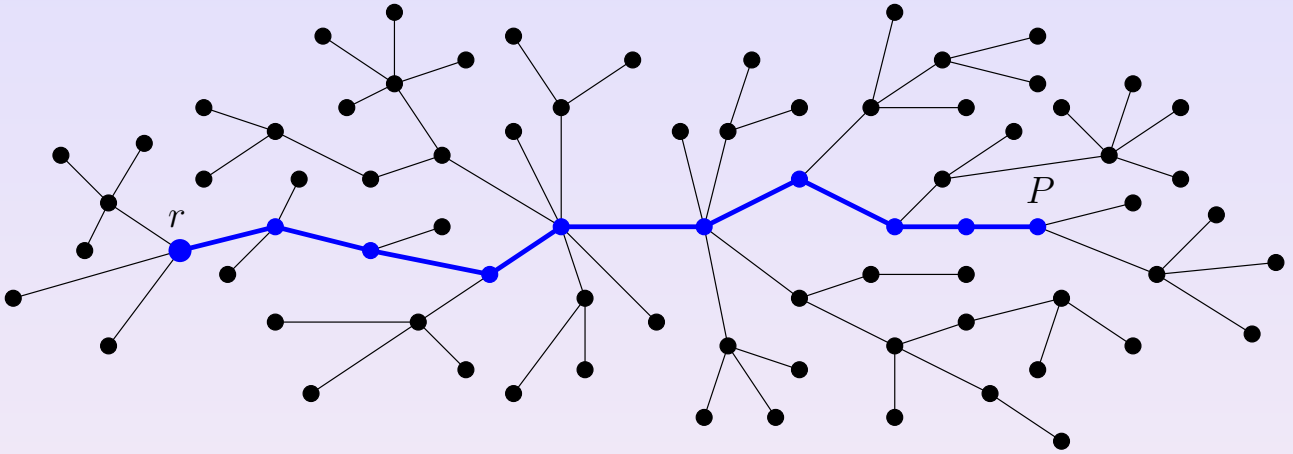
Idée générale : compresser et « localiser » les informations



1. Choisir un nœud r arbitraire comme racine de T

Un oracle simple pour les arbres

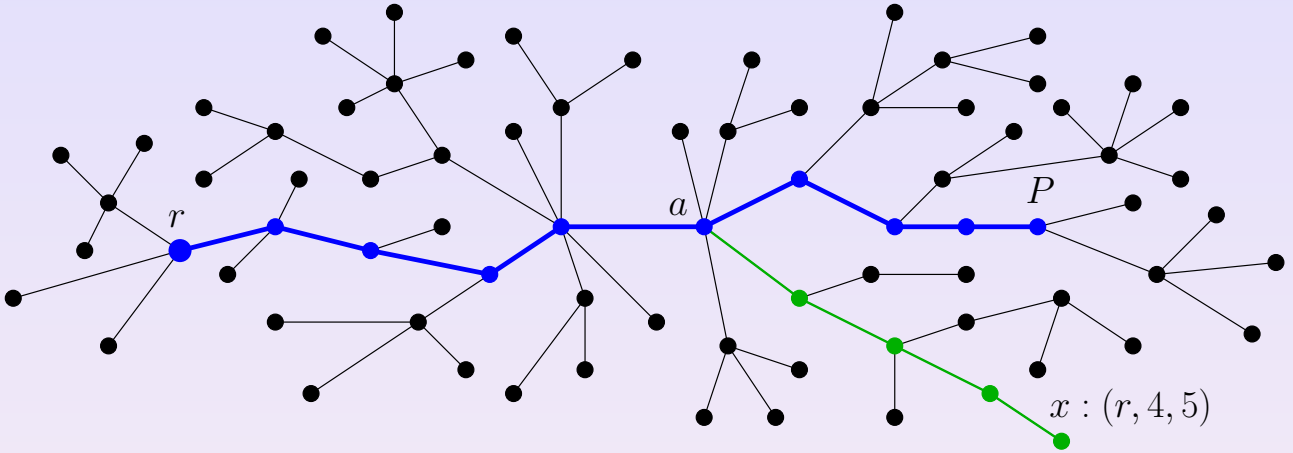
Idée générale : compresser et « localiser » les informations



1. Choisir un nœud r arbitraire comme racine de T
2. Prendre un chemin P qui « coupe en deux » T

Un oracle simple pour les arbres

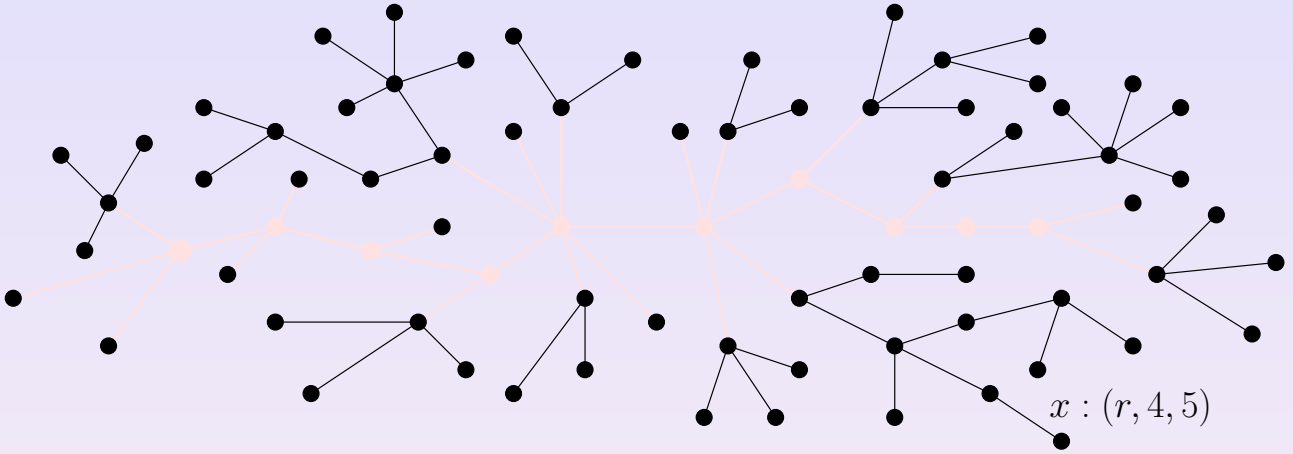
Idée générale : compresser et « localiser » les informations



1. Choisir un nœud r arbitraire comme racine de T
2. Prendre un chemin P qui « coupe en deux » T
3. Le nœud x stocke (r, d, h) où $d = d_T(x, a)$ et $h = d_T(r, a)$

Un oracle simple pour les arbres

Idée générale : compresser et « localiser » les informations



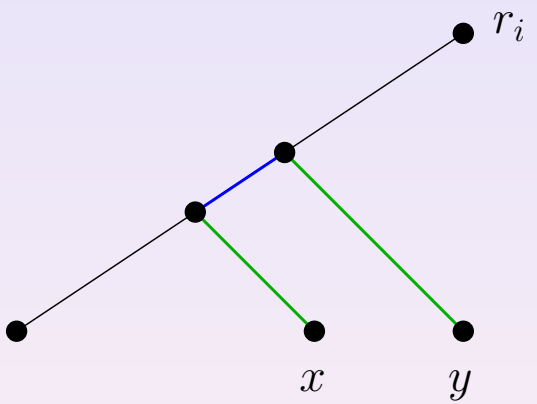
1. Choisir un nœud r arbitraire comme racine de T
2. Prendre un chemin P qui « coupe en deux » T
3. Le nœud x stocke (r, d, h) où $d = d_T(x, a)$ et $h = d_T(r, a)$
4. Recommencer avec les nœuds restant de $T \setminus P$

Décodage de la distance et analyse

Le nœud x stocke : $(r_1, d_1, h_1), \ldots, (r_k, d_k, h_k)$ où $d_k = 0$

Distance entre x et y :

1. Calculer le plus grand i tq $r_i(x) = r_i(y)$
2. renvoyer $d_i(x) + d_i(y) + |h_i(x) - h_i(y)|$

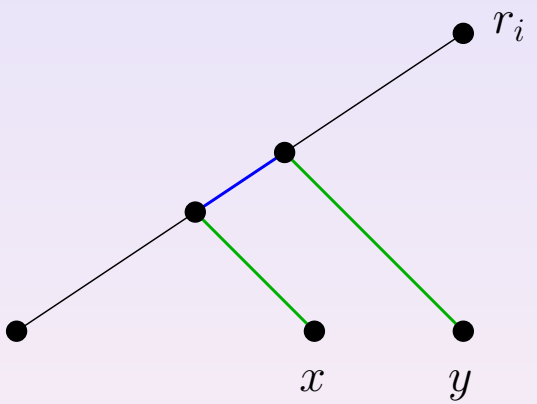


Décodage de la distance et analyse

Le nœud x stocke : $(r_1, d_1, h_1), \ldots, (r_k, d_k, h_k)$ où $d_k = 0$

Distance entre x et y :

1. Calculer le plus grand i tq $r_i(x) = r_i(y)$
2. renvoyer $d_i(x) + d_i(y) + |h_i(x) - h_i(y)|$



Complexité : $O(k) = O(\log n)$ [$O(1)$ possible]

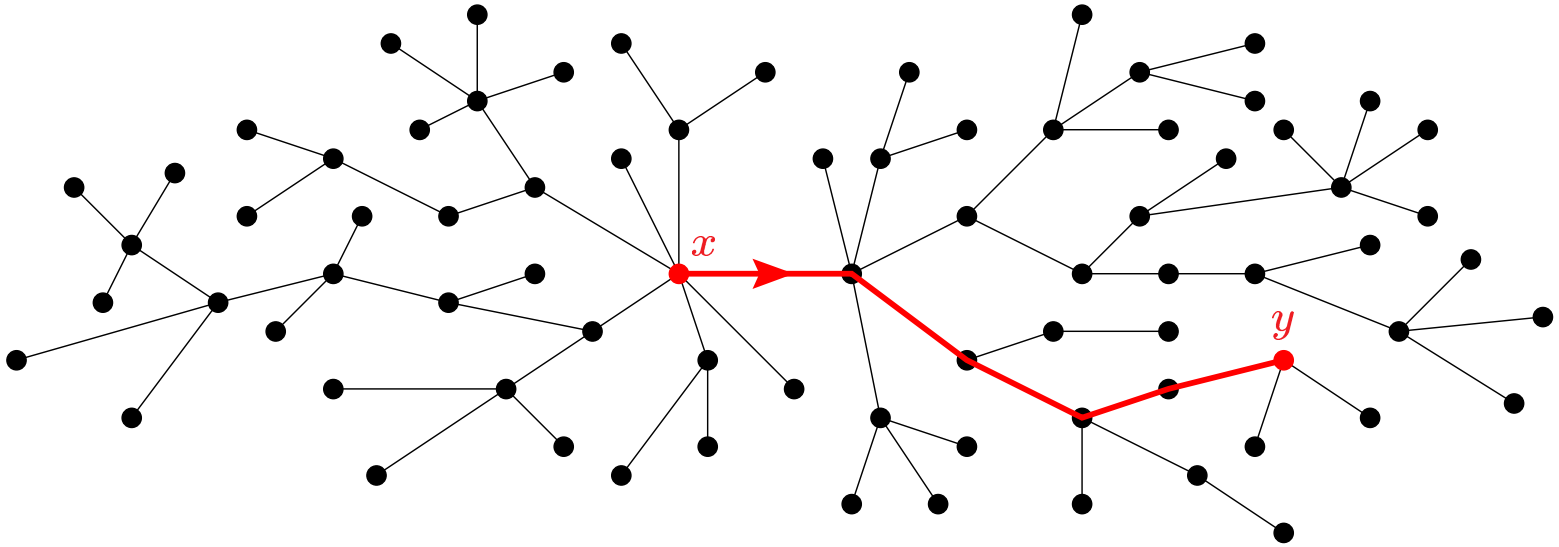
Pré-calcul : $O(nk) = O(n \log n)$ [$O(n)$ possible]

Agenda

1. Distance Labelling in General Graphs
2. Distance Labelling in Trees
3. **Routing in Trees**
4. Nearest Common Ancestor Labelling
5. Forbidden-Set Labelling
6. Distance in Planar Graphs
7. Distance in Minor-Free Graphs

Problem

To route a message from x to y in an n-node tree, $\forall\ x, y$

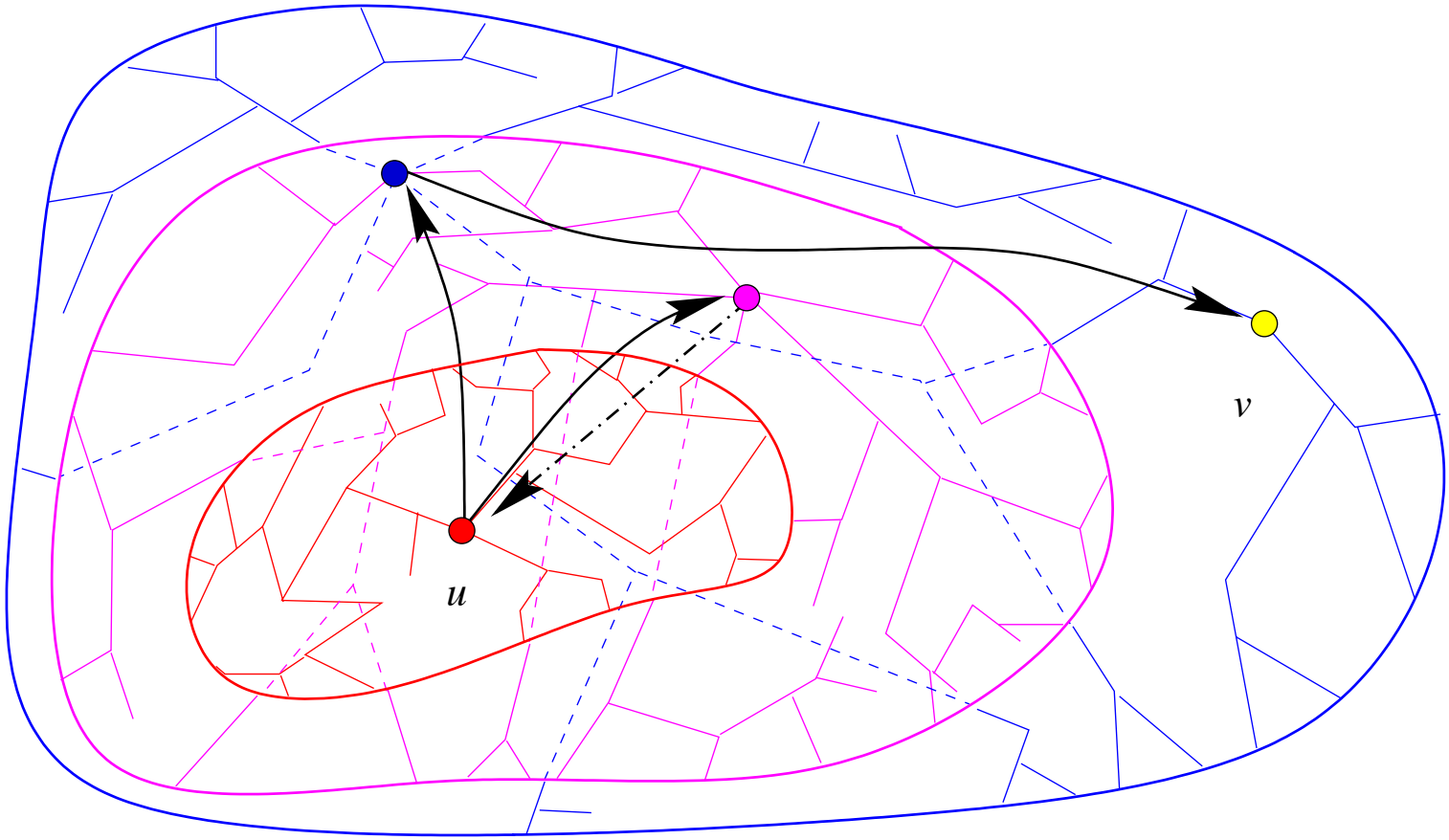


Constraints :

- shortest paths;
- local and compact routing tables;
- fast routing decision.

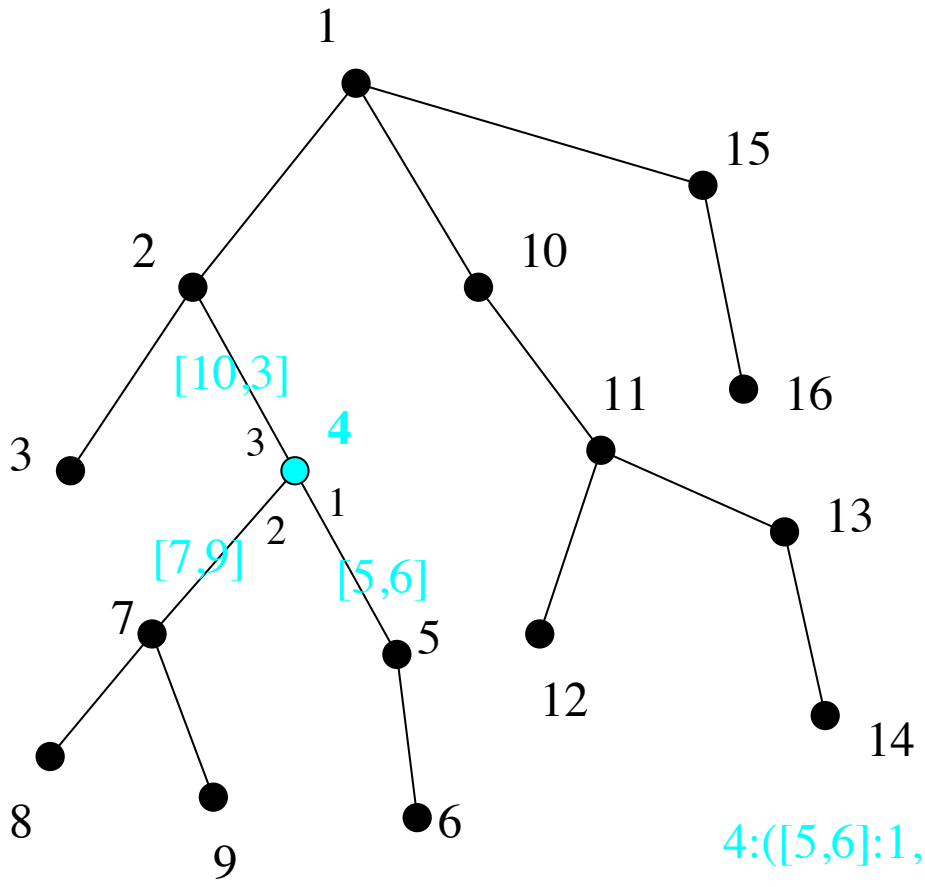
A Tool for Universal Routing Schemes

Hierarchical routing schemes applicable to <u>all</u> the networks
[Awerbuch,Cowen,Eilam,Frederickson,Peleg,Upfal,...]



Standard Techniques

[Santoro-Khatib/van Leeuwen-Tan]



- @ = DFS numbering
- interval routing based
- $O(n \log n)$ bits in total!

4:([5,6]:1,[7,9]:2,[10,3]:3)

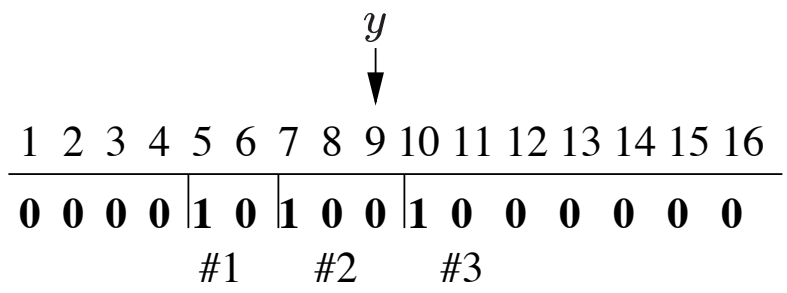
4:([5,6]:1,[7,9]:2,[10,3]:3)

Binary search $\Rightarrow O(\log n)$ routing time

Memory Space $\Rightarrow O(d \log n)$ bits, d = degree

Problem: Space $\Omega(n)$ bits whenever $d \geqslant n/\log n$

Another solution:

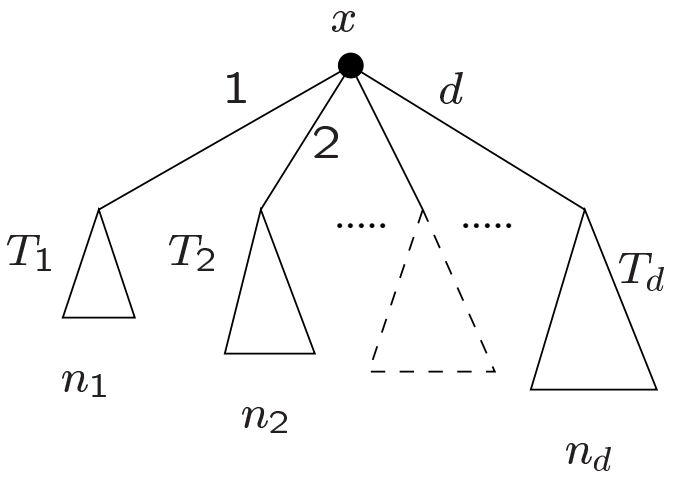


Routing algorithm (local): compute the number of 1's (mod d) before position y.

$\Rightarrow n$ bit space, but $\Omega(n)$ time! (In fact, this problem can be solved with $n + o(n)$ bits and constant time on a $O(\log n)$ bit word-RAM machine.)

Can we do better than n bits?

Lower bound: $c\sqrt{n}$ bits at least, $c \approx 3.7$



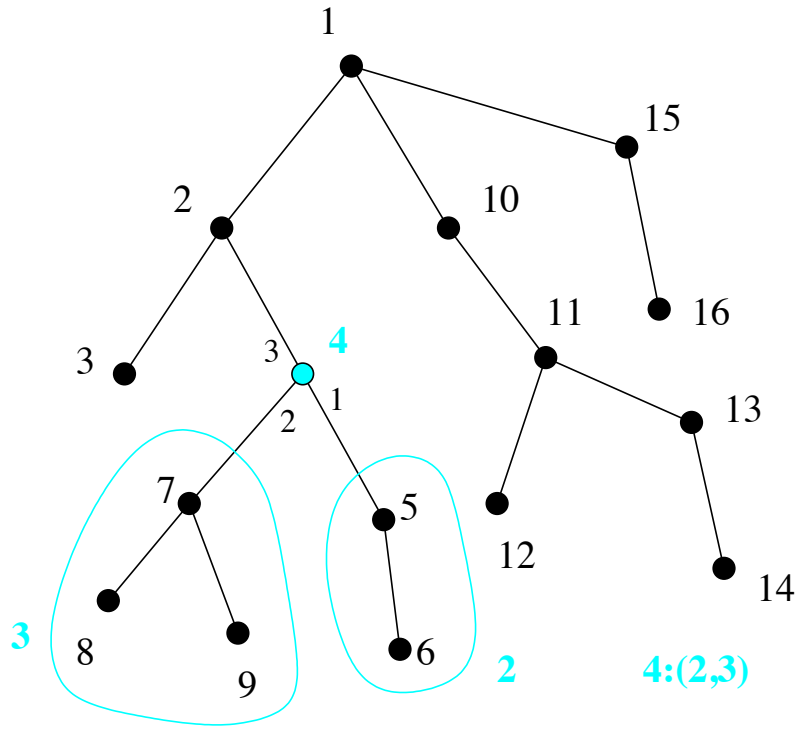
- test in x its local routing table towards all the possible destinations $y \in \{1, \ldots, n\} \setminus \{x\}$
- sort answers by frequency $n_i = |T_i|$, $n_1 \leqslant \cdots \leqslant n_d$

$\sum_{i=1}^{d} n_i = n - 1$ and $1 \leqslant n_1 \leqslant \cdots \leqslant n_d$.

x "computes" $(n_1, \ldots, n_d)$ knowing n and x.

The number of partitions of n is $\sim 2^{c\sqrt{n}}$ $\Rightarrow$ $c\sqrt{n}$ bits for x. $\square$

Can we do better than n bits? (cond't): YES!



- @ = DFS numbering w.r.t. the number of descendents
- port #1 = lightest child
  port #2= 2nd lightest child
  ...

$4 : (2, 3) \Rightarrow$ description with $O(\sqrt{n})$ bits (partition enumeration).

$y \in]4, 4 + 2]$? $y \in]6, 6 + 3]$? otherwise route on port 3.

Problem: Routing time? What happens if addresses $\notin \{1, \ldots, n\}$ and coded on more than $\log_2 n$ bits? #ports numbering?

Results (for n-node trees)

Port permut.?	Address size	Local Table size	Routing time	Preprocessing time
yes	$\log n$	$\geqslant 3.7\sqrt{n}$		
yes	$\log n$	$\leqslant 3.7\sqrt{n}$	$2^{O(\sqrt{n})}$	$2^{O(\sqrt{n})}$
no	$\log n$	$\geqslant n - o(n)$		
no	$\log n$	$\leqslant n + o(n)$	$O(1)$	$O(n^2)$
no	$O\left(\frac{\log^2 n}{\log\log n}\right)$	$O\left(\frac{\log^2 n}{\log\log n}\right)$	$O(1)$	$O(n \log n)$
yes	$\leqslant 5 \log n$	$\leqslant 3 \log n$	$O(1)$	$O(n \log n)$

Principle

(some notations)

r = root of the tree

weight(x) = number of descendents of x

id(x) = identifier for x using a DFS w.r.t. the heaviest children

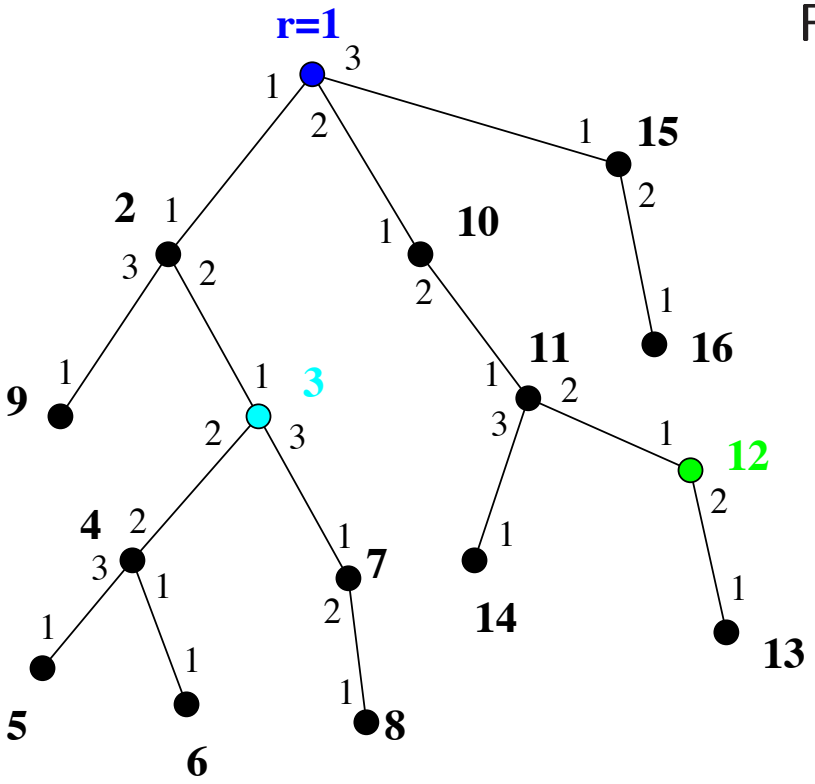
port$(x, \text{child}(x)) = p + 1$ for the p-th heaviest child of x (p=rank)

ℓpath(x) = list of large ports (rank$\geqslant 2$) on the path from r to x

$$
\begin{aligned}
@(x) &= \langle \text{id}(x), \ell\text{path}(x) \rangle \\
\text{table}(x) &= \langle \text{id}(x), \text{weight}(x), \text{weight}(\text{child}_1(x)), |\ell\text{path}(x)| \rangle
\end{aligned}
$$

Routing Algorithm

$$@(x) = \langle \mathrm{id}(x), \ell\mathrm{path}(x) \rangle$$
$$\mathrm{table}(x) = \langle \mathrm{id}(x), \mathrm{weight}(x), \mathrm{weight}(\mathrm{child}_1(x)), |\ell\mathrm{path}(x)| \rangle$$



Routing algorithm from x to y:

- 0 (=stop) if $\mathrm{id}(y) = \mathrm{id}(x)$
- 1 if $\mathrm{id}(y) \notin [\mathrm{id}(x), \mathrm{id}(x) + \mathrm{weight}(x)[$
- 2 if $\mathrm{id}(y) \in [\mathrm{id}(x), \mathrm{id}(x) + \mathrm{weight}(\mathrm{child}_1(x))[$
- $p + 1$ otherwise, p-th item of $\ell\mathrm{path}(y)$

<12,2,1,1> $\ell\mathrm{path}(12) = (2, \not{2}, \not{2})$
<3,6,3,0> $\ell\mathrm{path}(3) = (\not{1}, \not{2})$

Bit Counting

$\mathsf{rank}(x, y)$ = rank of the weight of the child y of x
($\approx \mathsf{port}(x, y) - 1 : 1, 2, ...$)

Observation:

- If y child x, $\mathsf{rank}(x, y) \leqslant \mathsf{weight}(x)/\mathsf{weight}(y)$

$\Rightarrow$ If $\ell\mathsf{path}(x) = (p_1, \ldots, p_k)$, $\prod p_i \leqslant n$ and $k \leqslant \log n$

Coding of ℓpath(x)

Ex: $\ell\text{path}(x) = (3, 9, 2, 5, 2, 4, 6) \longrightarrow (A, B)$

ℓpath(x) =	3	9	2	5	2	4	6
binary writing: A =	11	1001	10	101	10	100	110
field separator: B =	10	1000	10	100	10	100	100

$$|B| = |A| \quad \text{and} \quad |A| \leqslant \sum_{i=1}^{k} \lceil \log p_i \rceil \leqslant \log\left(\prod p_i\right) + k \leqslant 2 \log n$$

$\Rightarrow$ $\ell\text{path}(x) = (A, B)$ on $4 \log n$ bits

$\Rightarrow$ @(x) on $5 \log n$ bits

$\Rightarrow$ table(x) on $3 \log n + O(\log \log n)$ bits

Extracting the p-th item

$\mathsf{select}_B(t)$ = position of the t-th "1" in B.

If B is fixed, $\mathsf{select}_B(\cdot)$ can be implemented with $|B| + o(|B|)$ bit space and constant time (word-RAM model)

[Munro-Raman FOCS'97]

Ex: $\ell\mathsf{path}(x) = (3, 9, 2, \underline{5}, 2, 4, 6)$, $p = 4$.

$A = \overline{11100110101101010100110}$

$B = 10100010\underline{1}00\underline{1}0100100$

$a := \mathsf{select}_B(p)$,

$b := \mathsf{select}_B(p+1)$,

$s := (A \ll (a-1)) \gg (|A| + a - b) =$ "101" $\rightarrow 5$.

Improving the Constants ...

Lemma (Kalmár 1930). Let $Z(n)$ be the number of integer sequences $p_1, p_2, \ldots$ such that $p_i \geqslant 2$ and $\prod p_i \leqslant n$. Then,

$$Z(n) \;\sim\; n^{1.7286\ldots} \quad (\text{as } n \to +\infty)$$

(where 1.7286... is the real root of $\zeta(x) = 2$, $\zeta(x) = \sum_{i \geqslant 1} i^{-x}$).

Thus, @(x) on $2.7286 \log n$ bits and table(x) on $3 \log n$ bits.

Independently, [Thorup-Zwick, SPAA'01]:
@(x) on $(1 + o(1)) \log n$ bits and
table(x) on $(1 + o(1)) \log n$ bits.

Agenda

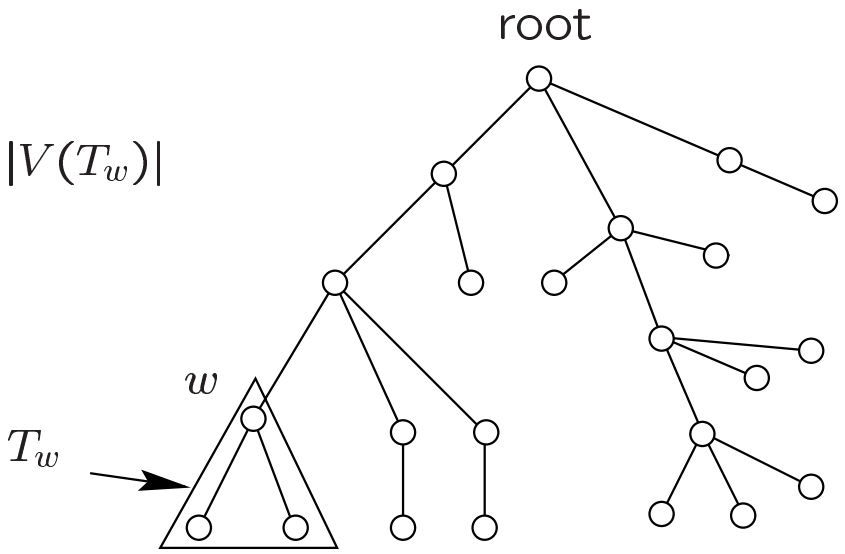
1. Distance Labelling in General Graphs
2. Distance Labelling in Trees
3. Routing in Trees
4. **Nearest Common Ancestor Labelling**
5. Forbidden-Set Labelling
6. Distance in Planar Graphs
7. Distance in Minor-Free Graphs

Theorem. *There is a linear time algorithm that labels the n nodes of a rooted tree T with labels of length $O(\log n)$ bits such that from the labels of nodes $x, y \in T$ alone, one can compute the label of* $\mathrm{nca}(x, y)$ *in* $O(1)$ *time.*

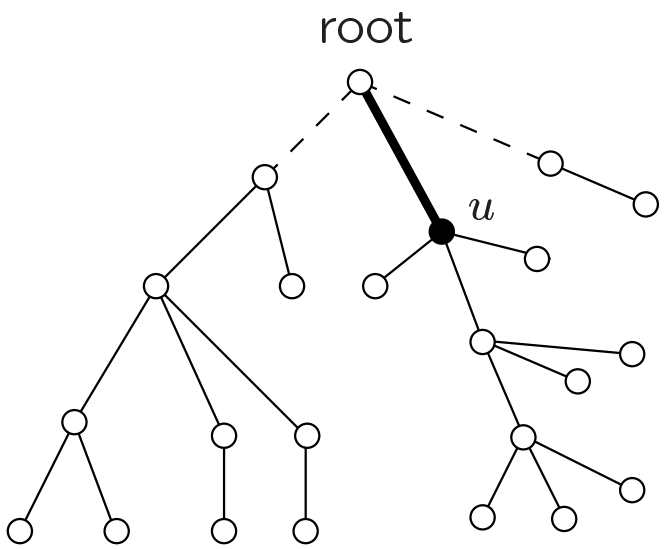
Corollary. *Let $x_1, \ldots, x_n$ be a sequence of n real numbers. We can assign in linear time a label of length $O(\log n)$ bits to each element, such that, given the labels of x_i, x_j, the label of a maximum among $x_i, \ldots, x_j$ can be computed in $O(1)$ time from the labels of x_i and x_j alone.*

NCA Labeling Algorithm

size(w) = $|V(T_w)|$



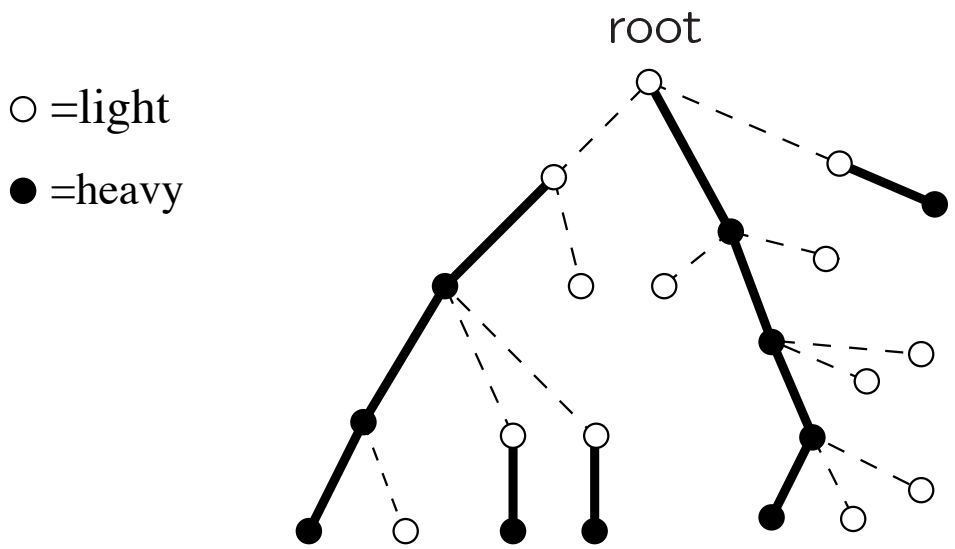
NCA Labeling Algorithm



Partition the nodes and the edges of T:

The heavy child is a child u with size(u) maximum. The other children are light. The root r is light. Heavy/light edges.

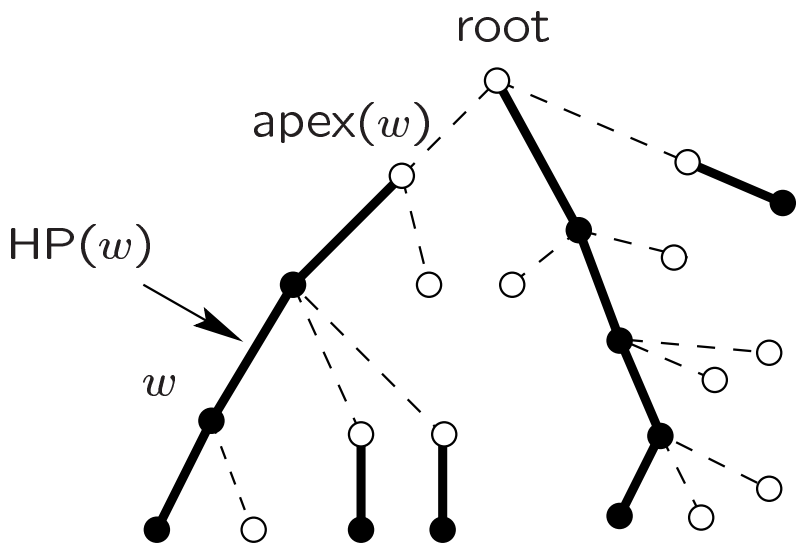
NCA Labeling Algorithm



Partition the nodes and the edges of T:

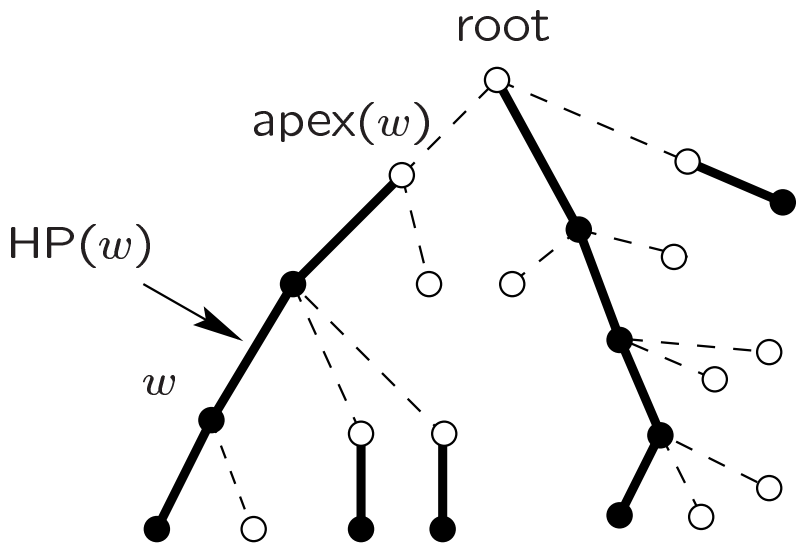
The heavy child is a child u with size(u) maximum. The other children are light. The root r is light. Heavy/light edges.

NCA Labeling Algorithm



HP(w)=Heavy Path containing w.
apex(v)=closest light ancestor of v.

NCA Labeling Algorithm



If v is light then:

$\Rightarrow$ size(v) $< \frac{1}{2}$size(parent(v))

$\Rightarrow$ v has at most $\log_2 n$ light ancestors (or apex)

Labels

v has: a light label: llabel(v)
a heavy label: hlabel(v)

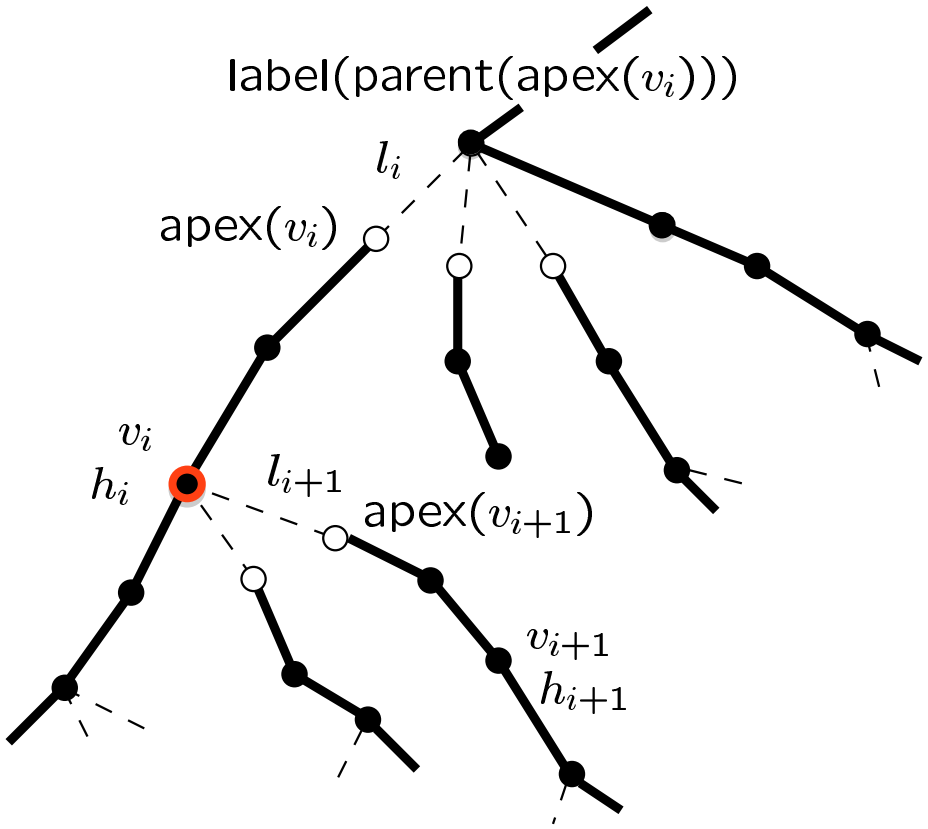
1. llabel(root) := ϵ
2. light label, $v \neq$ root

$$\text{llabel}(v) \notin \{\text{llabel}(z) \mid z \neq v, z \in \text{children}(\text{parent}(v))\}$$

3. heavy label, $v \neq$ root

$$\text{hlabel}(v) <_{\text{lex}} \min_{\text{lex}} \{\text{hlabel}(z) \mid z \neq v, z \in T_v \cap \text{HP}(v)\}$$

(lex: lexicographic order on binary strings)



$$\begin{aligned}\text{label}(v) &:= \text{label}(\text{parent}(\text{apex}(v))) \cdot \text{llabel}(\text{apex}(v)) \cdot \text{hlabel}(v) \\ &= h_0 \cdot l_1 \cdot h_1 \cdot l_2 \cdot h_2 \cdots l_t \cdot h_t \quad (t \leqslant \log_2 n)\end{aligned}$$

(with label(parent(root)) := ϵ)

NCA label computation

$$\text{label}(x) = h_0 \cdot l_1 \cdot h_1 \cdots l_i \cdot h_i$$
$$\text{label}(y) = h'_0 \cdot l'_1 \cdot h'_1 \cdots l'_j \cdot h'_j$$

NCA label computation

$$\text{label}(x) = h_0 \cdot l_1 \cdot h_1 \cdots l_i \cdot h_i$$
$$\text{label}(y) = h'_0 \cdot l'_1 \cdot h'_1 \cdots l'_j \cdot h'_j$$

If label(x), label(y) differ at a light label l_p, then
[case parent(apex(x)) = parent(apex(y))*]*

$$\text{label}(\text{nca}(x, y)) = h_0 \cdot l_1 \cdot h_1 \cdots l_{p-1} \cdot h_{p-1}$$

NCA label computation

$$\text{label}(x) = h_0 \cdot l_1 \cdot h_1 \cdots l_i \cdot h_i$$
$$\text{label}(y) = h'_0 \cdot l'_1 \cdot h'_1 \cdots l'_j \cdot h'_j$$

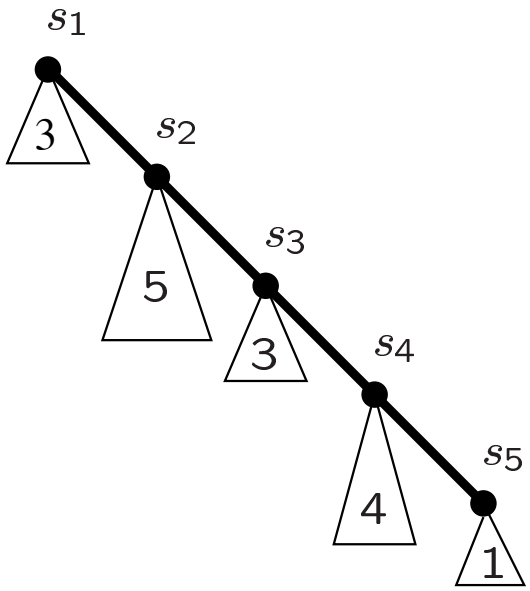
If label(x), label(y) differ at a light label l_p, then
[case parent(apex(x)) = parent(apex(y))*]*

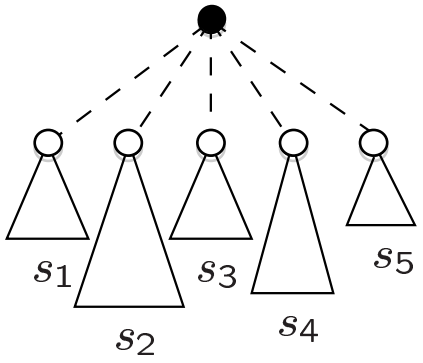
$$\text{label}(\text{nca}(x, y)) = h_0 \cdot l_1 \cdot h_1 \cdots l_{p-1} \cdot h_{p-1}$$

If label(x), label(y) differ at a heavy label h_p, then
[case HP(x) = HP(y)*]*

$$\text{label}(\text{nca}(x, y)) = h_0 \cdot l_1 \cdot h_1 \cdots l_p \cdot \min_{\text{lex}} \left\{h_p, h'_p\right\}$$

Label length





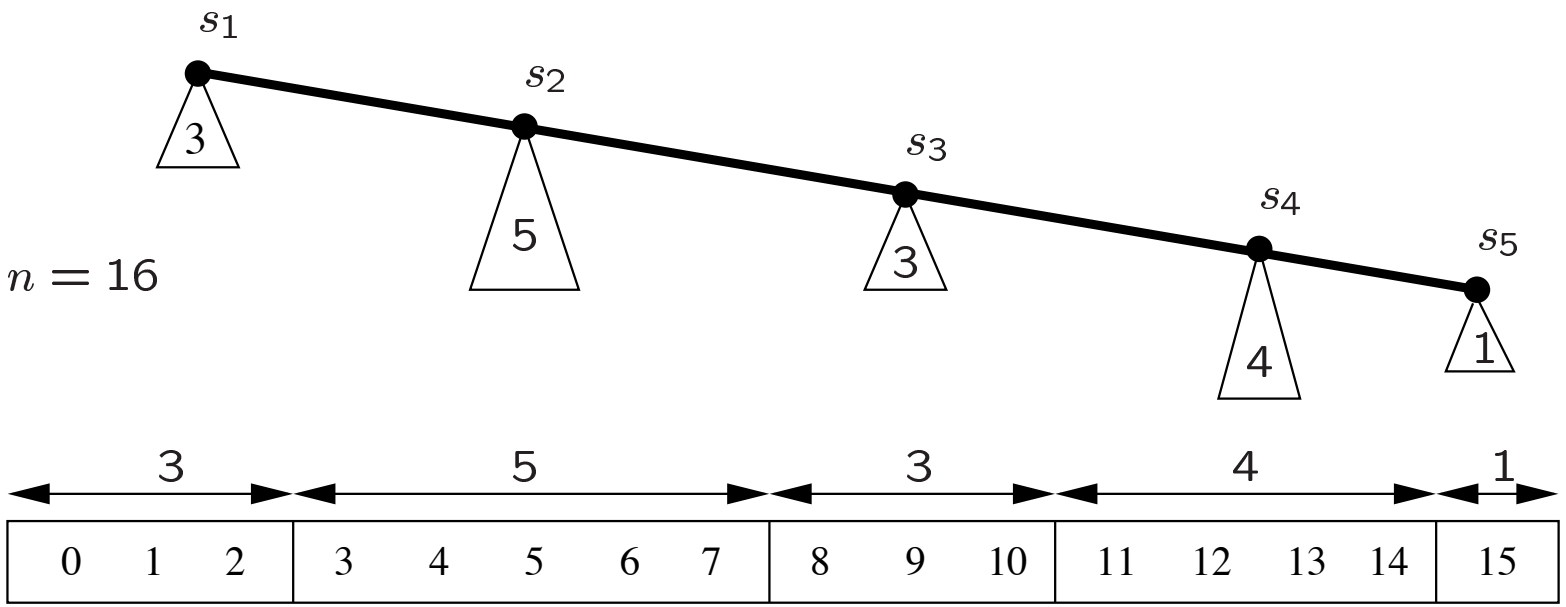
Let $s_1, ..., s_k$ be numbers $\geqslant 1$ with $\sum_i s_i = n$

There exists: $s_i \mapsto \text{code}(s_i)$ a binary string such that

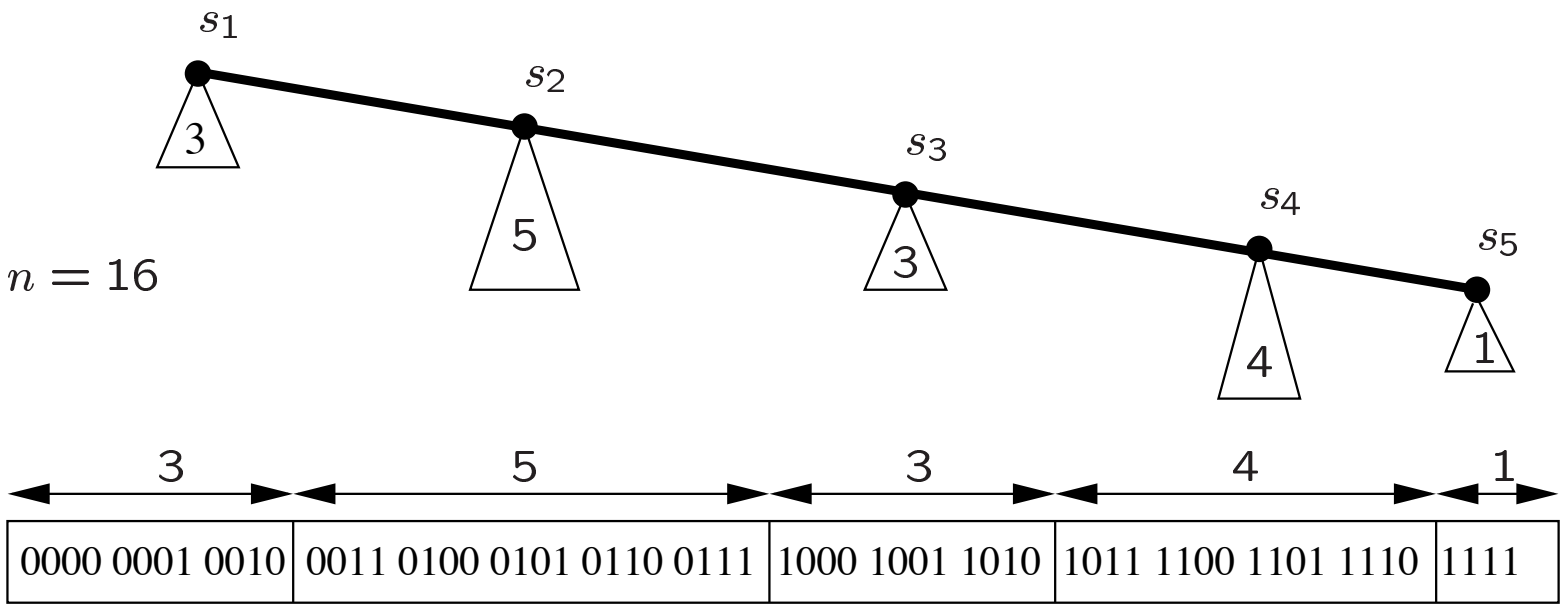
$$\text{code}(s_1) <_{\text{lex}} \cdots <_{\text{lex}} \text{code}(s_k) \quad \text{and}$$

$$1 \leqslant |\text{code}(s_i)| \leqslant \lceil \log_2 n \rceil - \lfloor \log_2 s_i \rfloor \quad \text{bits}$$

Alphabetic code

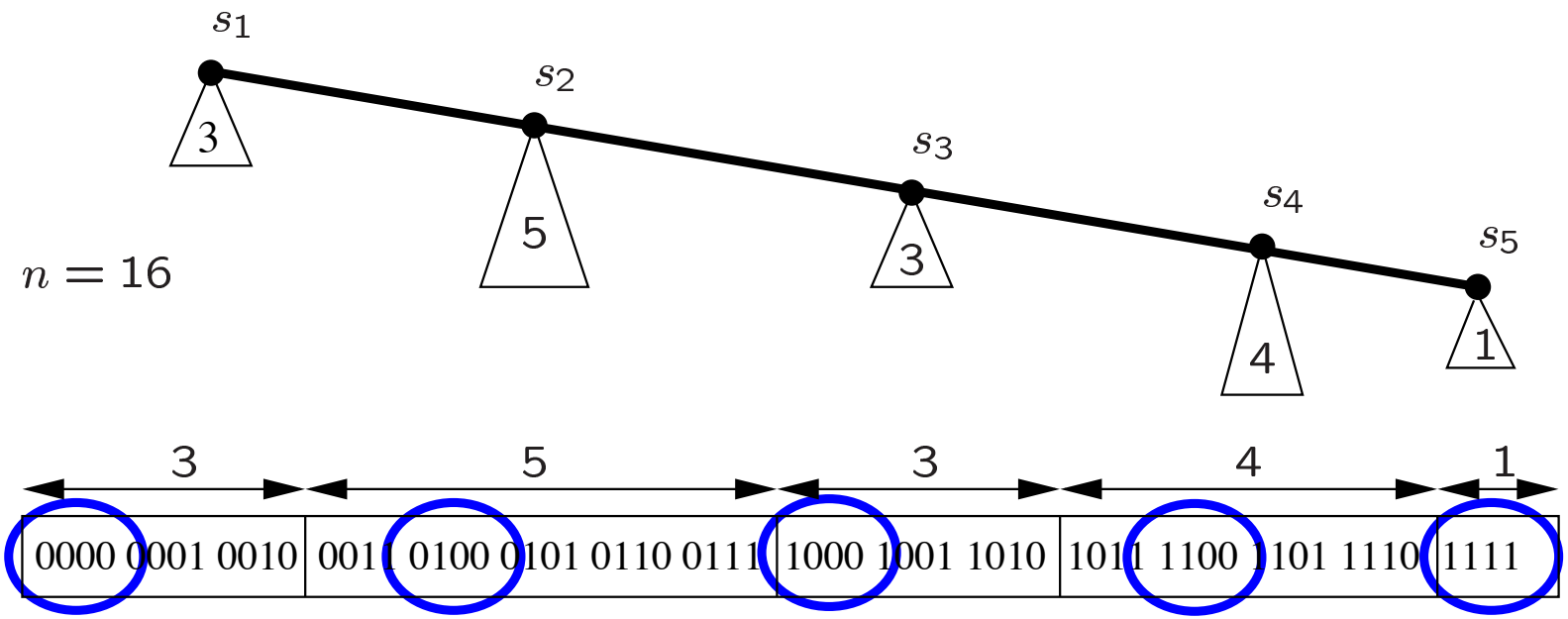


Alphabetic code



There is $z \in [x, x + s_i)$ with at least $\lfloor \log_2 s_i \rfloor$ rightmost zero bits.

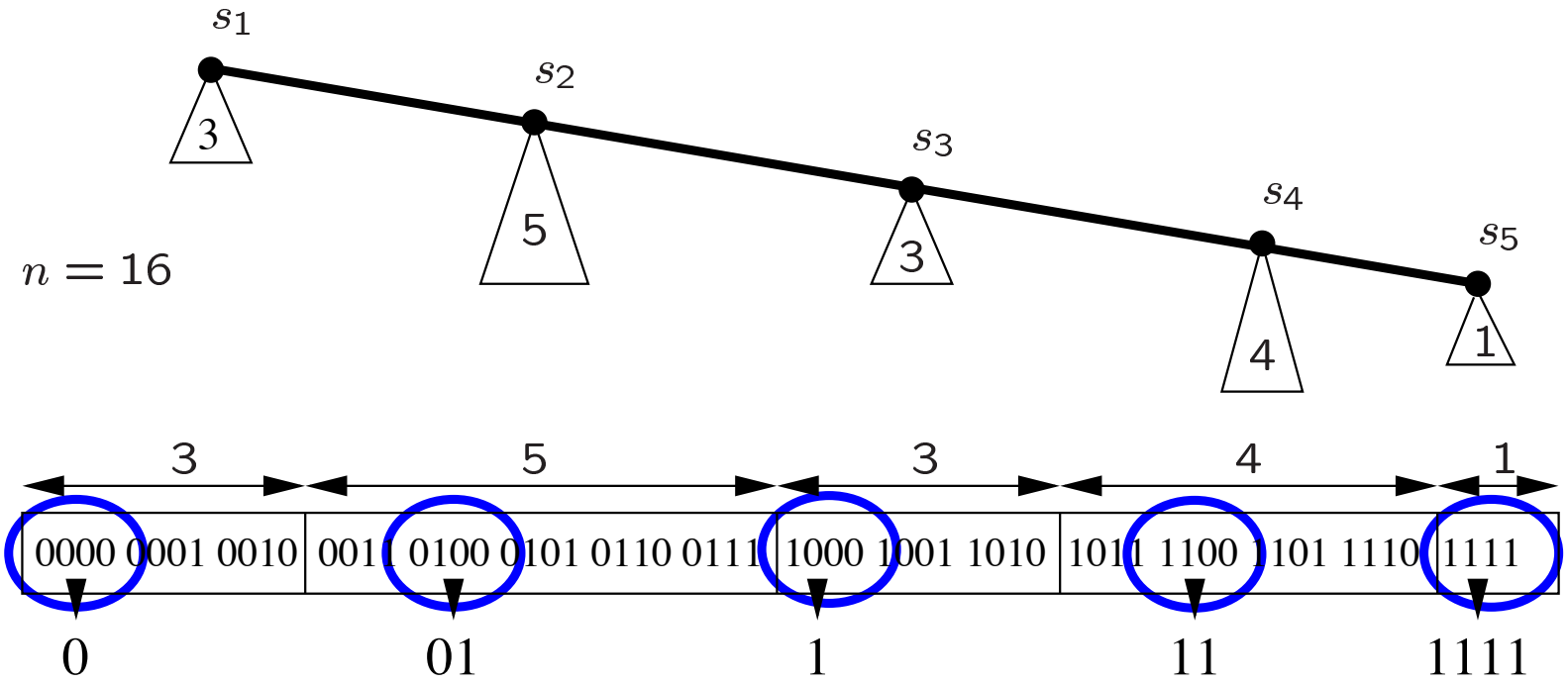
Alphabetic code



There is $z \in [x, x + s_i)$ with at least $\lfloor \log_2 s_i \rfloor$ rightmost zero bits.

$\Rightarrow$ code$(s_i) = z$ with rightmost zero bits removed

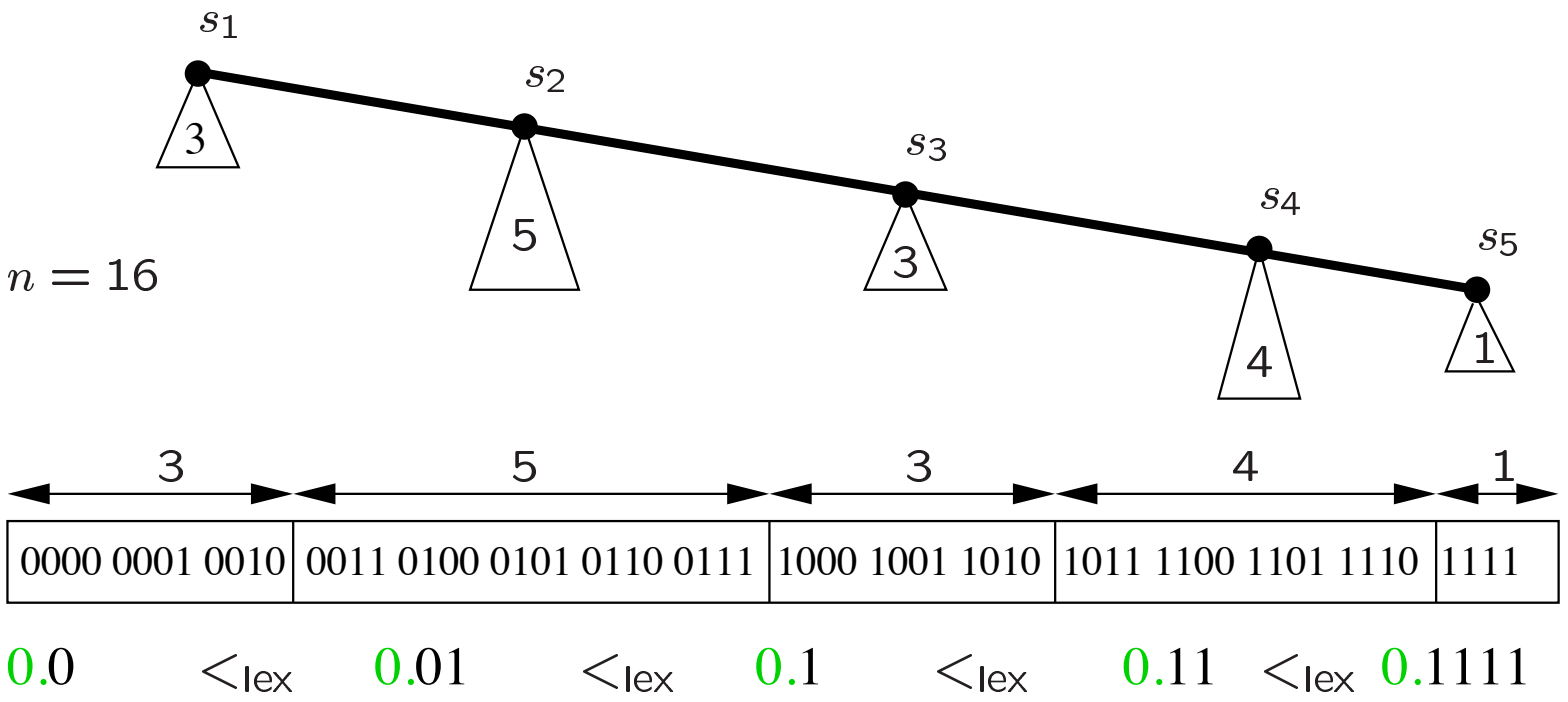
Alphabetic code



There is $z \in [x, x + s_i)$ with at least $\lfloor \log_2 s_i \rfloor$ rightmost zero bits.

$\Rightarrow$ code$(s_i) = z$ with rightmost zero bits removed

Alphabetic code



There is $z \in [x, x + s_i)$ with at least $\lfloor \log_2 s_i \rfloor$ rightmost zero bits.

$\Rightarrow$ code$(s_i) = z$ with rightmost zero bits removed

Length analysis

$\mathsf{lsize}(v) = \mathsf{size}(v) - \mathsf{size}(w)$, for w heavy child of v.

$|\mathsf{llabel}(w)| < \log_2 \mathsf{lsize}(\mathsf{parent}(w)) - \log_2 \mathsf{size}(w) + O(1)$
$|\mathsf{hlabel}(w)| < \log_2(\sum_{v \in \mathsf{HP}(w)} \mathsf{lsize}(v)) - \log_2 \mathsf{lsize}(w) + O(1)$

$\Rightarrow |\mathsf{label}(w)| < \log_2 n - \log_2 \mathsf{lsize}(w) + O(1)$

Length analysis

$\mathsf{lsize}(v) = \mathsf{size}(v) - \mathsf{size}(w)$, for w heavy child of v.

$|\mathsf{llabel}(w)| < \log_2 \mathsf{lsize}(\mathsf{parent}(w)) - \log_2 \mathsf{size}(w) + O(1)$
$|\mathsf{hlabel}(w)| < \log_2(\sum_{v \in \mathsf{HP}(w)} \mathsf{lsize}(v)) - \log_2 \mathsf{lsize}(w) + O(1)$

$\Rightarrow |\mathsf{label}(w)| < \log_2 n - \log_2 \mathsf{lsize}(w) + O(1)$

[By induction]
True if $\mathsf{apex}(w) =$ *root since* $\mathsf{label}(w) = \mathsf{hlabel}(w)$.
Otherwise, by hypothesis we have:

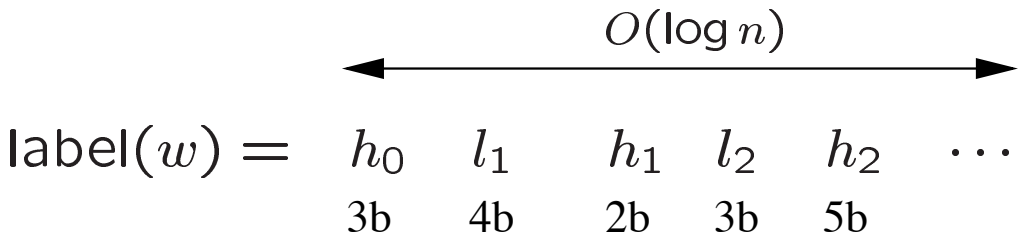
$$|\mathsf{label}(\mathsf{parent}(\mathsf{apex}(w)))| < \log_2 n - \log_2 \mathsf{lsize}(\mathsf{parent}(\mathsf{apex}(w)))$$

But

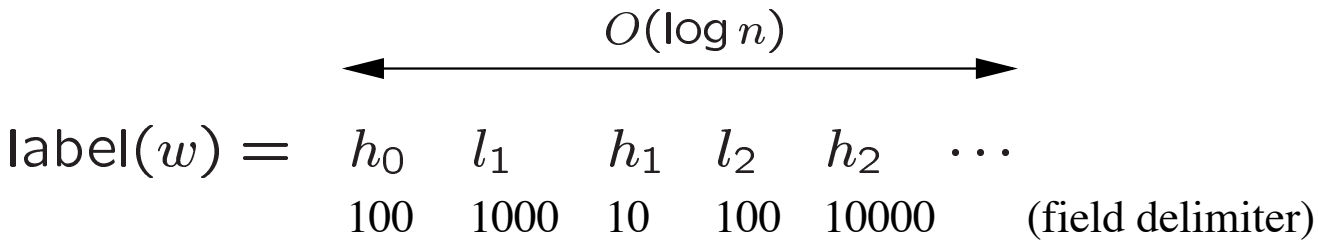
$|\mathsf{llabel}(\mathsf{apex}(w))| < \log_2 \mathsf{lsize}(\mathsf{parent}(\mathsf{apex}(w))) - \log_2 \mathsf{size}(\mathsf{apex}(w))$

And, $|\mathsf{hlabel}(w)| < \log_2 \mathsf{size}(w) - \log_2 \mathsf{lsize}(w)$ *using*
$\sum_{v \in \mathsf{HP}(w)} \mathsf{lsize}(v) < \mathsf{size}(\mathsf{apex}(w))$

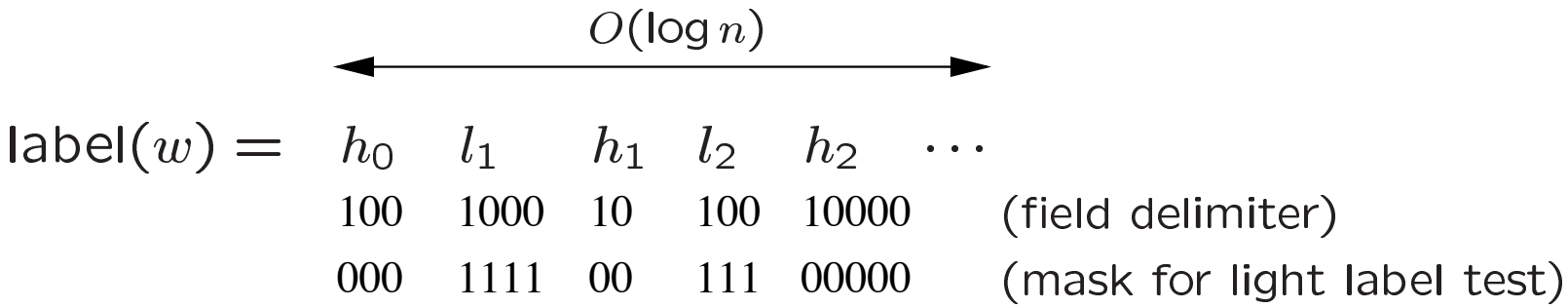
Constant query time using bit manipulations



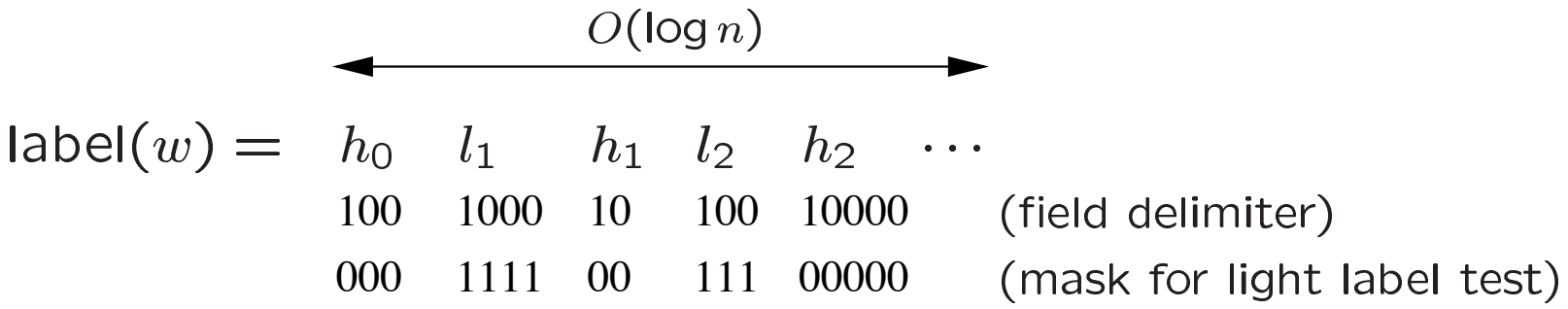
Constant query time using bit manipulations



Constant query time using bit manipulations

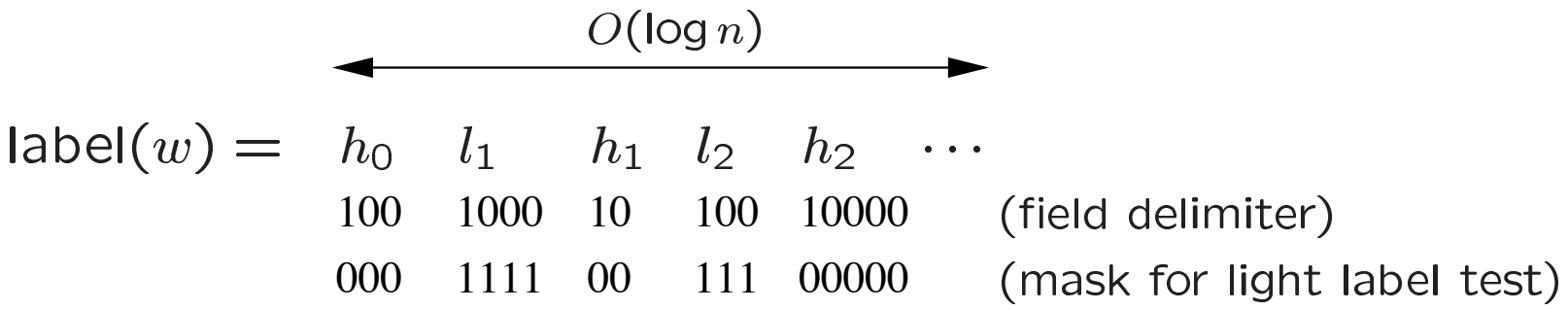


Constant query time using bit manipulations



$\Rightarrow$ label(nca(x, y)) extracted in constant time on a RAM computer with $\Omega(\log n)$ bit words.

Constant query time using bit manipulations



$\Rightarrow$ label(nca(x, y)) extracted in constant time on a RAM computer with $\Omega(\log n)$ bit words.

$\Rightarrow$ Linear time for computing all the labels.

Agenda

1. Distance Labelling in General Graphs
2. Distance Labelling in Trees
3. Routing in Trees
4. Nearest Common Ancestor Labelling
5. **Forbidden-Set Labelling**
6. Distance in Planar Graphs
7. Distance in Minor-Free Graphs

Forbidden-set labelling scheme
(extension of labelling scheme)

- Goal: to treat more elaborated queries

  Given (u,v,w): is there a path from u to v in G\{w}?

Forbidden-set labelling scheme
(extension of labelling scheme)

- Goal: to treat more elaborated queries

  Given (u,v,w): is there a path from u to v in G\{w}?

  [this particular task reduces to classical nca-labelling scheme for the bicomponent/cut-vertex tree: $\Rightarrow$ O(logn) bit labels]

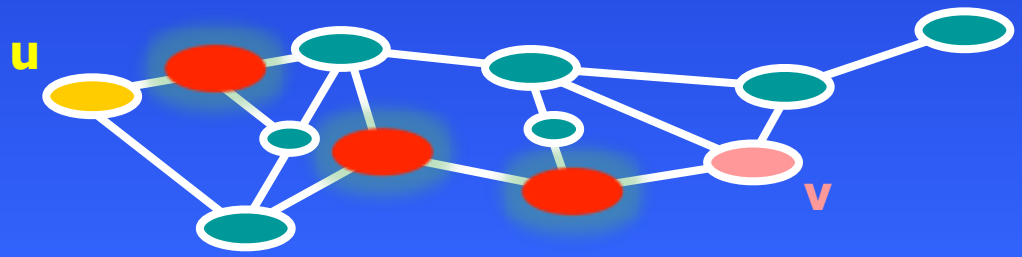
Forbidden-set labelling scheme
(extension of labelling scheme)

- Goal: to treat more elaborated queries

  Given (u,v,w): is there a path from u to v in G\{w}?

  [this particular task reduces to classical nca-labelling scheme for the bicomponent/cut-vertex tree: ⇒ O(logn) bit labels]

- Challenge: Given $(u,v,w_1,\ldots,w_k)$: is there a path from u to v in $G\setminus\{w_1,\ldots,w_k\}$?



Emergency planning for connectivity

[Patrascu,Thorup - FOCS'07]

- Motivation: parallel attack (link/node failure in IP black-bone, earthquake on road networks, malicious attack from worms or viruses,…)

Emergency planning for connectivity

[Patrascu,Thorup - FOCS'07]

- Motivation: parallel attack (link/node failure in IP black-bone, earthquake on road networks, malicious attack from worms or viruses,…)

- CONN(u,v) ⇒ constant time (after pre-processing G)
- CONN(u,v,w) ⇒ constant time (after pre-proc. G)

Emergency planning for connectivity

[Patrascu,Thorup - FOCS'07]

- Motivation: parallel attack (link/node failure in IP black-bone, earthquake on road networks, malicious attack from worms or viruses,…)

- CONN(u,v) ⇒ constant time (after pre-processing G)
- CONN(u,v,w) ⇒ constant time (after pre-proc. G)

- CONN(u,v,w_1,…,w_k) ⇒ O(k) or $\tilde{O}$(k) time? (after pre-proc. G), and constant time? (after pre-proc. w_1…w_k)

Emergency planning for connectivity

[Patrascu,Thorup - FOCS'07]

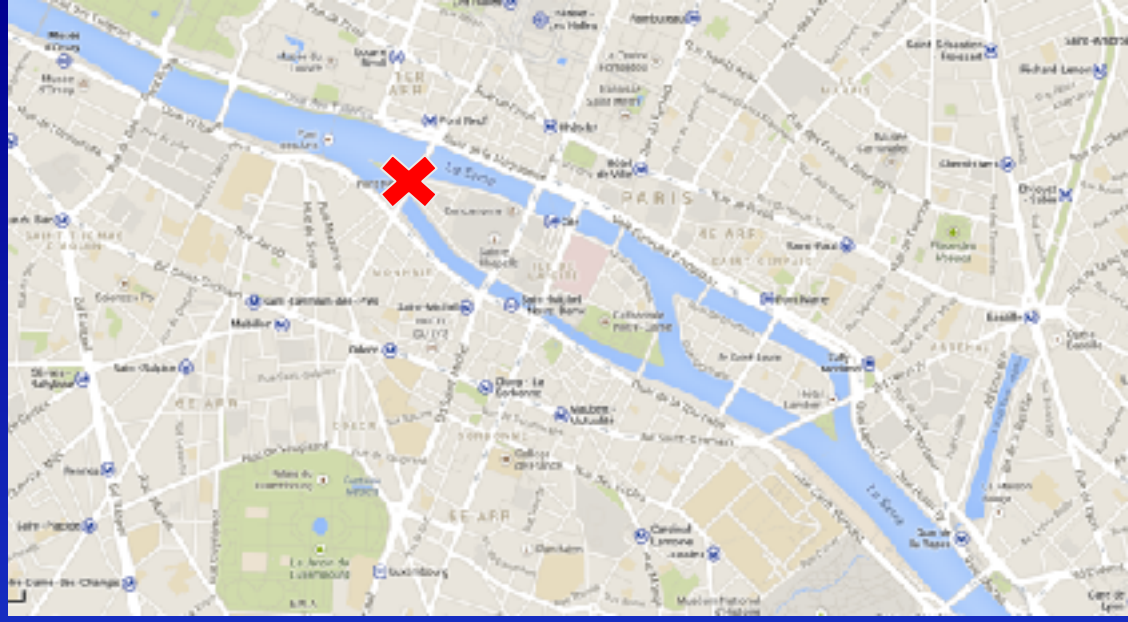
- Motivation: parallel attack (link/node failure in IP black-bone, earthquake on road networks, malicious attack from worms or viruses,…)

- CONN(u,v) ⇒ constant time (after pre-processing G)
- CONN(u,v,w) ⇒ constant time (after pre-proc. G)

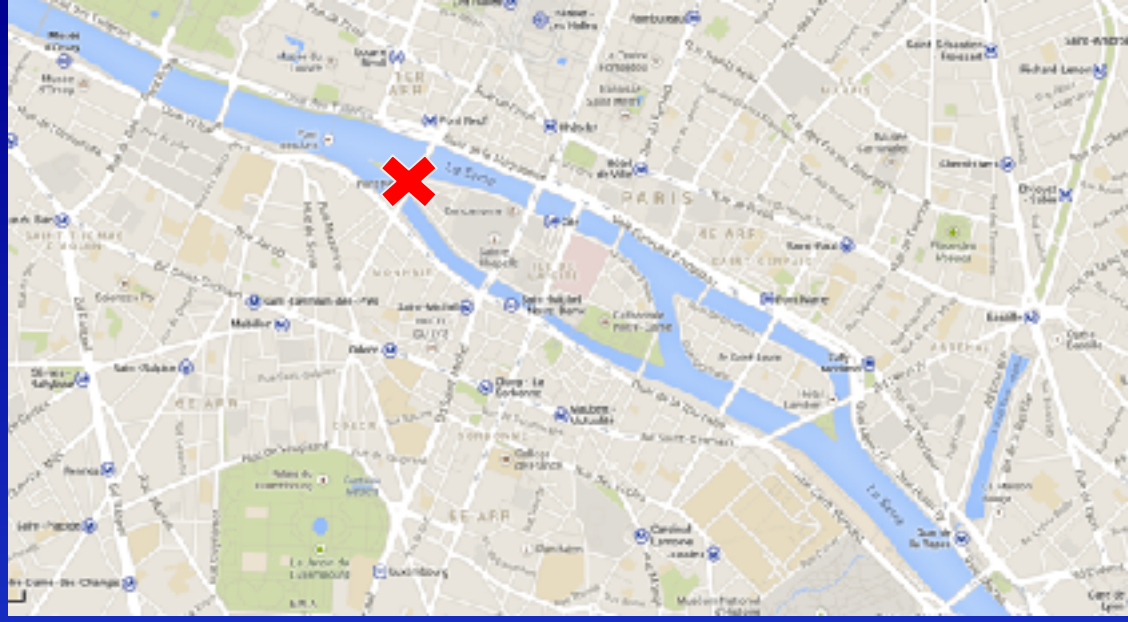
- CONN(u,v,w_1,…,w_k) ⇒ O(k) or $\tilde{O}$(k) time? (after pre-proc. G), and constant time? (after pre-proc. w_1…w_k)

- Note: O(n+m) time is too much. Need a query time depending only on the #nodes involved in the query.

Assume there is a failure x (node or edge) due to: flooding, earthquake, damage, attack …



Assume there is a failure x (node or edge) due to: flooding, earthquake, damage, attack …



How to find efficiently the connected component of any node u in G\{x}?

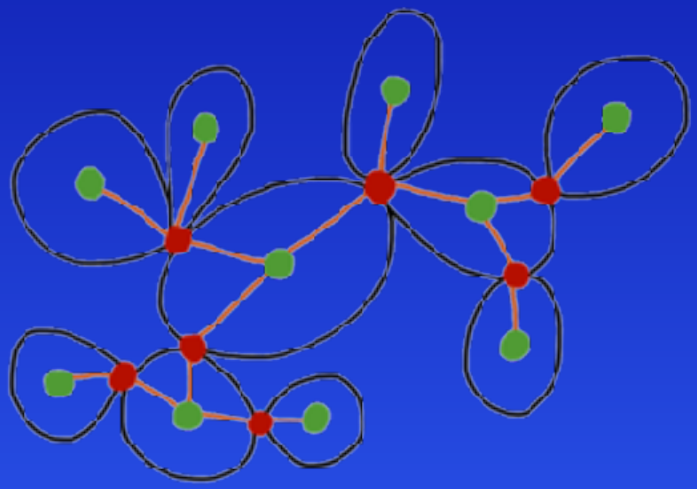
➟ Update all the component labels with a linear time traversal of G\{x}, and then answer in O(1) for each query node u.

Main issue: G is ***extremely*** large, and even linear time is too much in case of emergency! We would like the answer ***immediately***.

If we pre-process G ***accordingly***, can we then quickly answer queries "is there a path from u to v in G\{x}"?

Yes we can!

Pre-process G in a more clever way. Identify cut-vertices, the component-tree and design an efficient NCA data structure (all take linear time).



u,v are not connected in G\{x} iff x is a cut-vertex on the path from c(u) to c(v) in the component-tree.

Forbidden-set labelling scheme

[Courcelle,Twigg - STACS'07]

Let P be a graph property defined on pairs of vertices, and let F be a graph family.

A P –forbidden-set labeling scheme for F is a pair $\langle L,f \rangle$ s.t. $\forall\, G \in F$, $\forall u,v \in G$, $\forall\, X \subseteq G$:

- $L(u,G)$ is a binary string
- $f(L(u,G),L(v,G),L(X,G)) = P\,(u,v,X,G)$

where $L(X,G) := \{L(w,G) : w \in X\}$

FS connectivity labelling

[Courcelle,G.,Kanté,Twigg – TGGT'08]

[Borradaile,Pettie,Wulff-Nilsen – SWAT'12]

Connectivity in planar graphs: O(logn) bit labels

[O(loglogn) query time after O(klogk) time for query pre-processing]

FS connectivity labelling

[Courcelle,G.,Kanté,Twigg – TGGT'08]

[Borradaile,Pettie,Wulff-Nilsen – SWAT'12]

Connectivity in planar graphs: O(logn) bit labels

[O(loglogn) query time after O(klogk) time for query pre-processing]

Meta-Theorem: [Courcelle,Twigg – STACS'07]

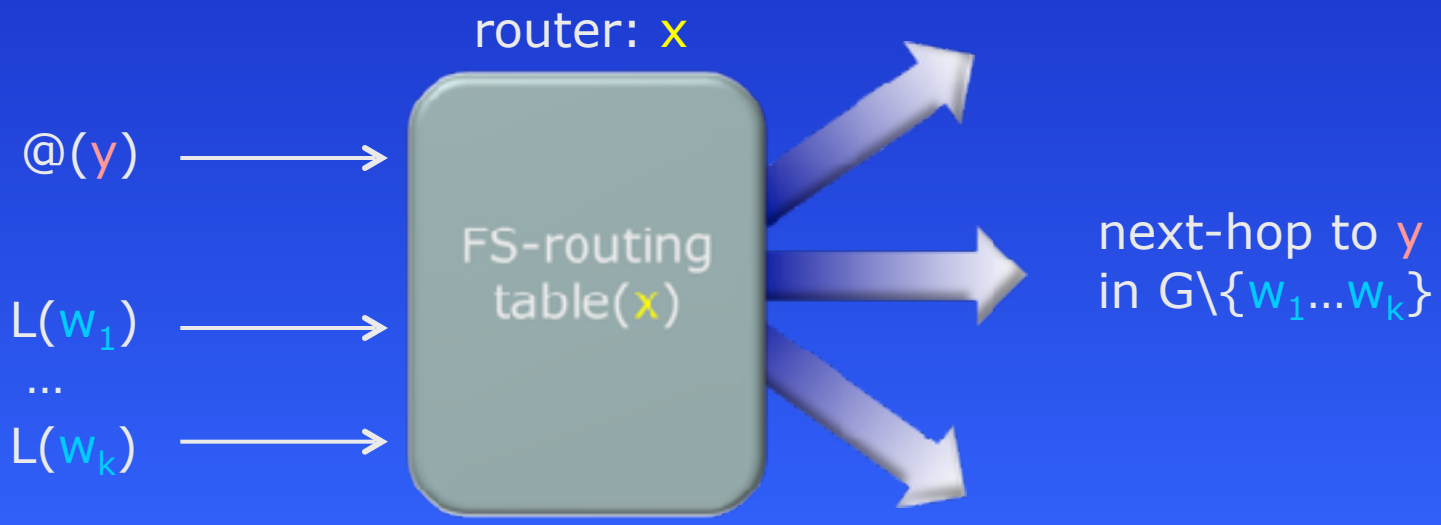
If G has "clique-width" at most cw (generalization of tree-width) and if predicate P is expressed in MSO-logic (like distances, connectivity, …), then labels of $O(cw^2 \log^2 n)$-bit suffice.

Notes: same (optimal) bounds for distances in trees for the static case, but do not include planar …

Routing with forbidden-sets

Design a routing scheme for G s.t. for every subset X of "forbidden" nodes (crashes, malicious, …) routing tables can be updated efficiently provided X.

⇒ This capture routing policies



Some results for FS routing

[Courcelle,Twigg – STACS'07]

Clique-width cw: $O(cw^2 \log^2 n)$ bit labels and routing tables for shortest path routing.

[Abraham,Chechik,G.,Peleg – PODC'10]

Doubling dimension-α: $O(1+\varepsilon^{-1})^{2\alpha} \log^2 n$ bit labels and routing tables for stretch $1+\varepsilon$ routing (wrt. shortest path)

[Abraham,Chechik,G. – STOC'12]

Planar: $O(\varepsilon^{-1} \log^3 n)$ bit routing tables and $O(\varepsilon^{-2} \log^5 n)$ bit labels for stretch $1+\varepsilon$ routing

Focus on Connectivity
(in planar graphs)

(1) [Courcelle,G.,Kanté,Twigg – TGGT'08]
(2) [Borradaile,Pettie,Wulff-Nilsen – SWAT'12]

Pre-processing time: O(n)

Query pre-processing time: O(|X|log|X|)

(2) Space: O(n) & Q. Time: O(loglogn)

(1) Space: O(logn) bit labels & Q. time: O(√logn)

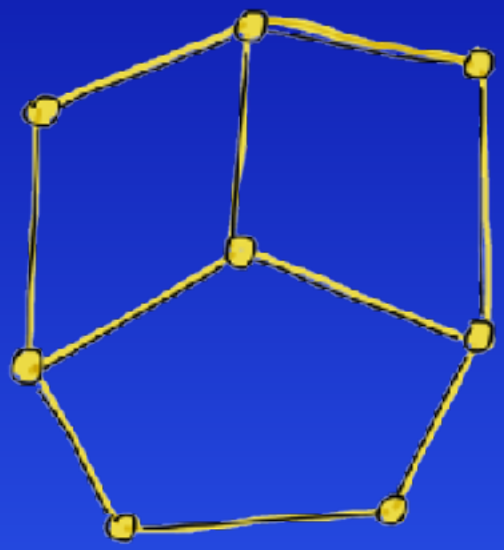
(2) can be generalized to H-minor free graphs

Main Idea (1)

[Here X subset of edges only ... much more tricky otherwise]

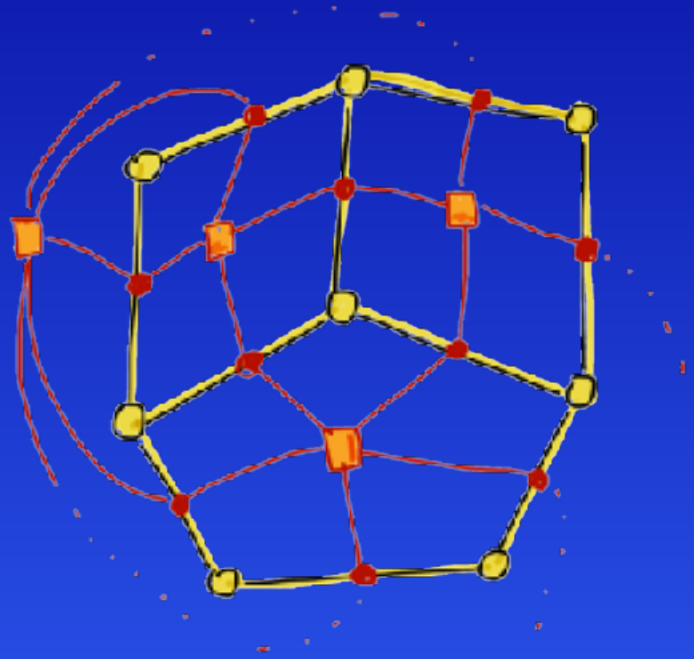
Main Idea (1)

[Here X subset of edges only ... much more tricky otherwise]



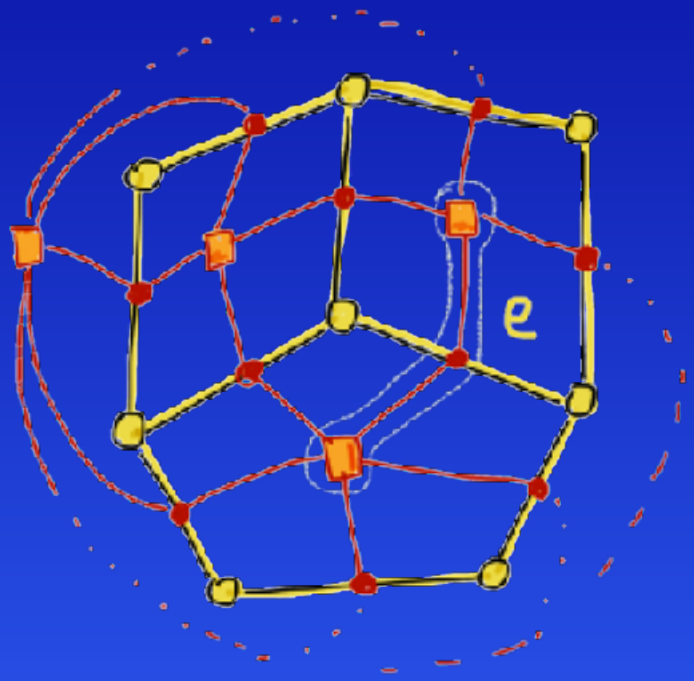
Main Idea (1)

[Here X subset of edges only ... much more tricky otherwise]



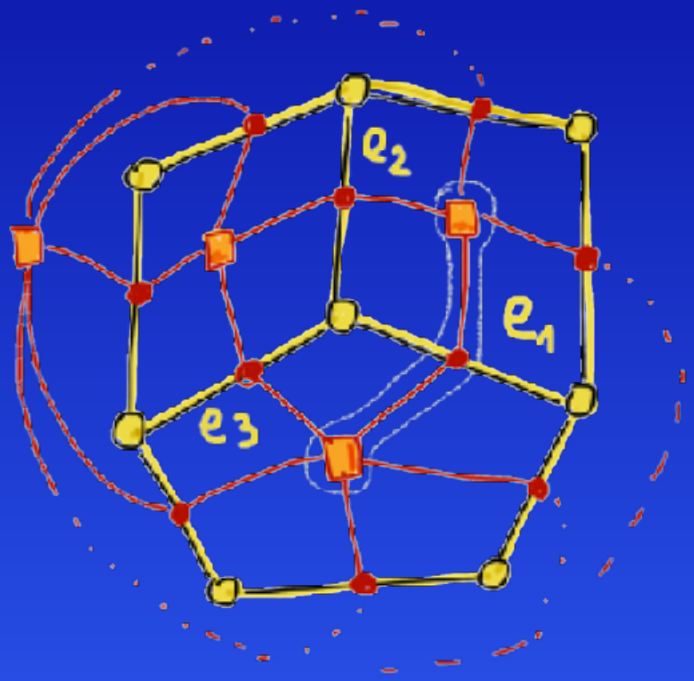
Main Idea (1)

[Here X subset of edges only ... much more tricky otherwise]



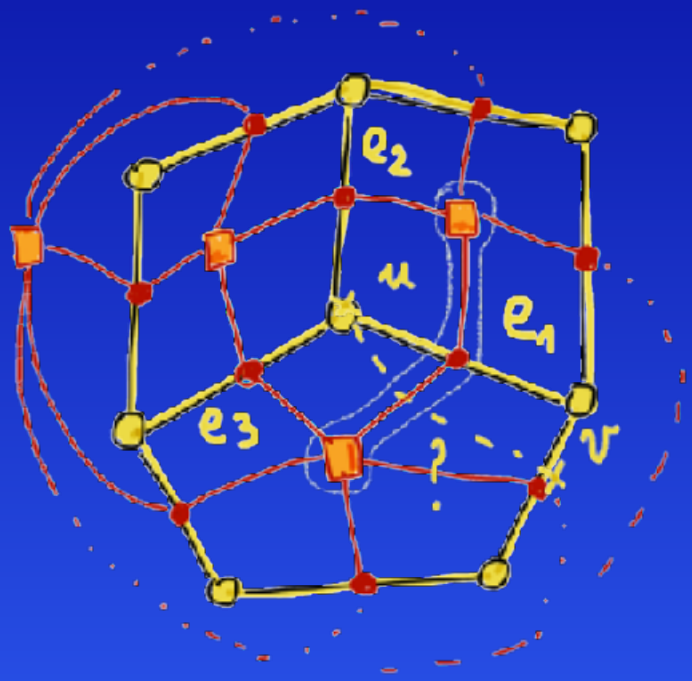
Main Idea (1)

[Here X subset of edges only ... much more tricky otherwise]



Main Idea (1)

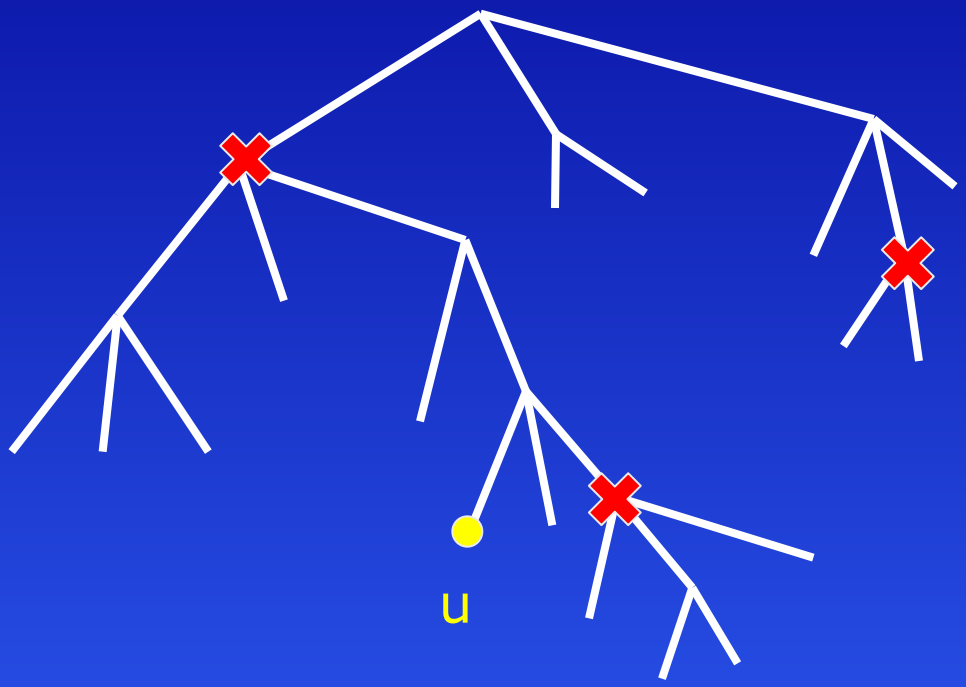
[Here X subset of edges only ... much more tricky otherwise]



Query = PLANAR POINT LOCATION in O(√logn) time (polynomial coordinates)

Note: space does NOT depend on |X|.

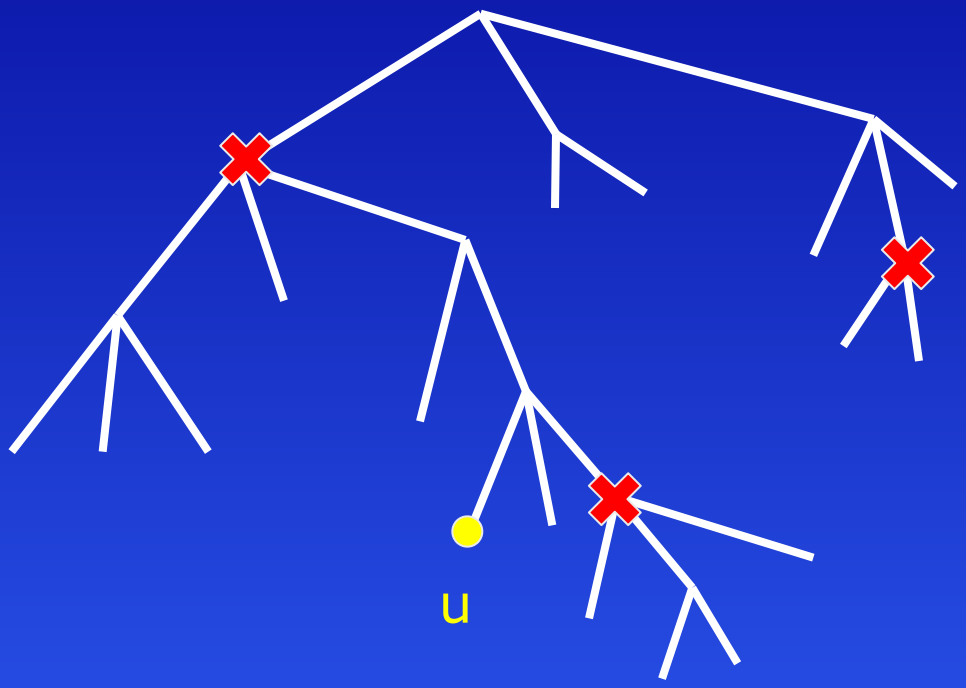
A Simple Solution for Trees



$\text{Find}_{G\setminus X}(u)$?

(find some identifier of the component of u in G\X)

A Simple Solution for Trees

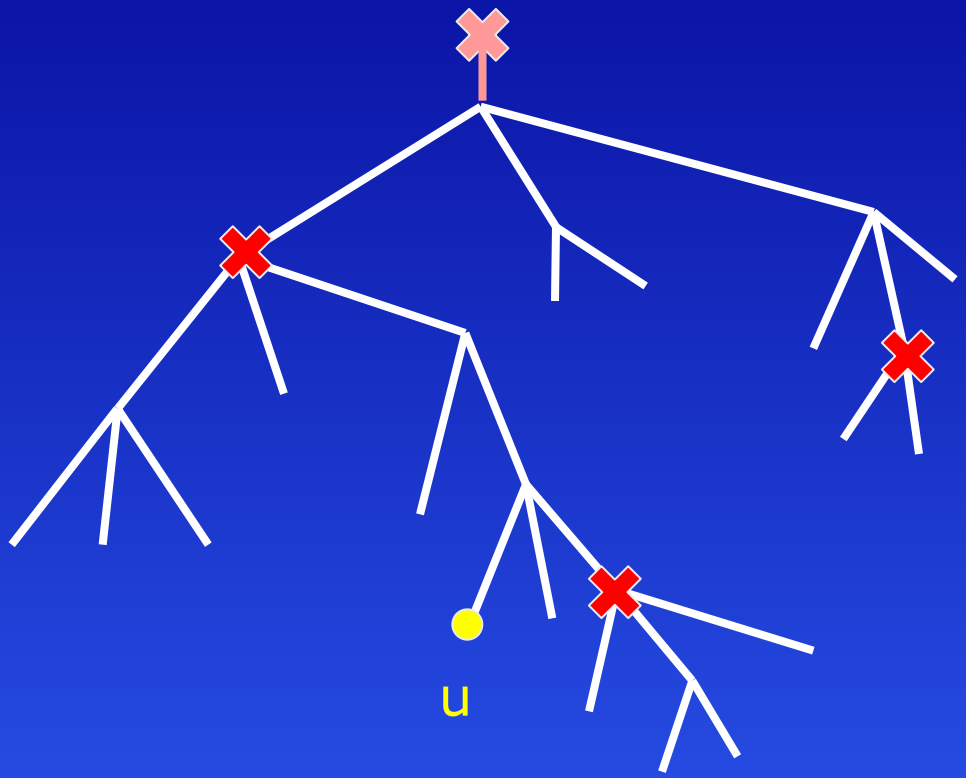


Find$_{G\setminus X}$(u)?

(find some identifier of the component of u in G\X)

1. Find the closest failure x ancestor of u

A Simple Solution for Trees

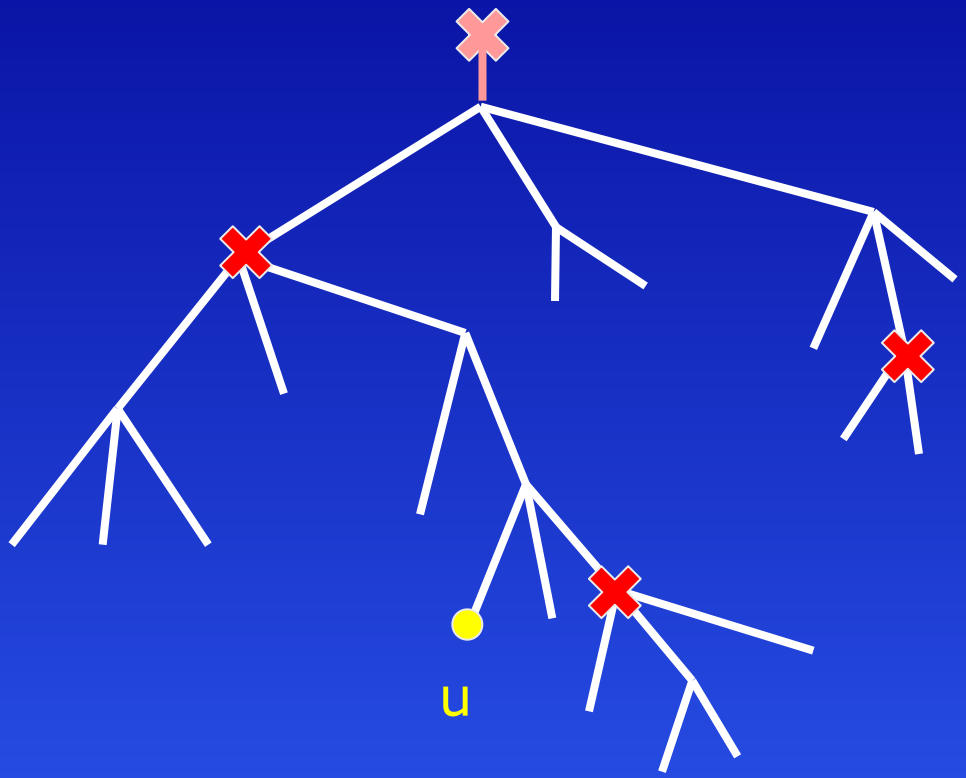


Find$_{G\setminus X}$(u)?

(find some identifier of the component of u in G\X)

1. Find the closest failure x ancestor of u

A Simple Solution for Trees



$\text{Find}_{G\setminus X}(u)$?

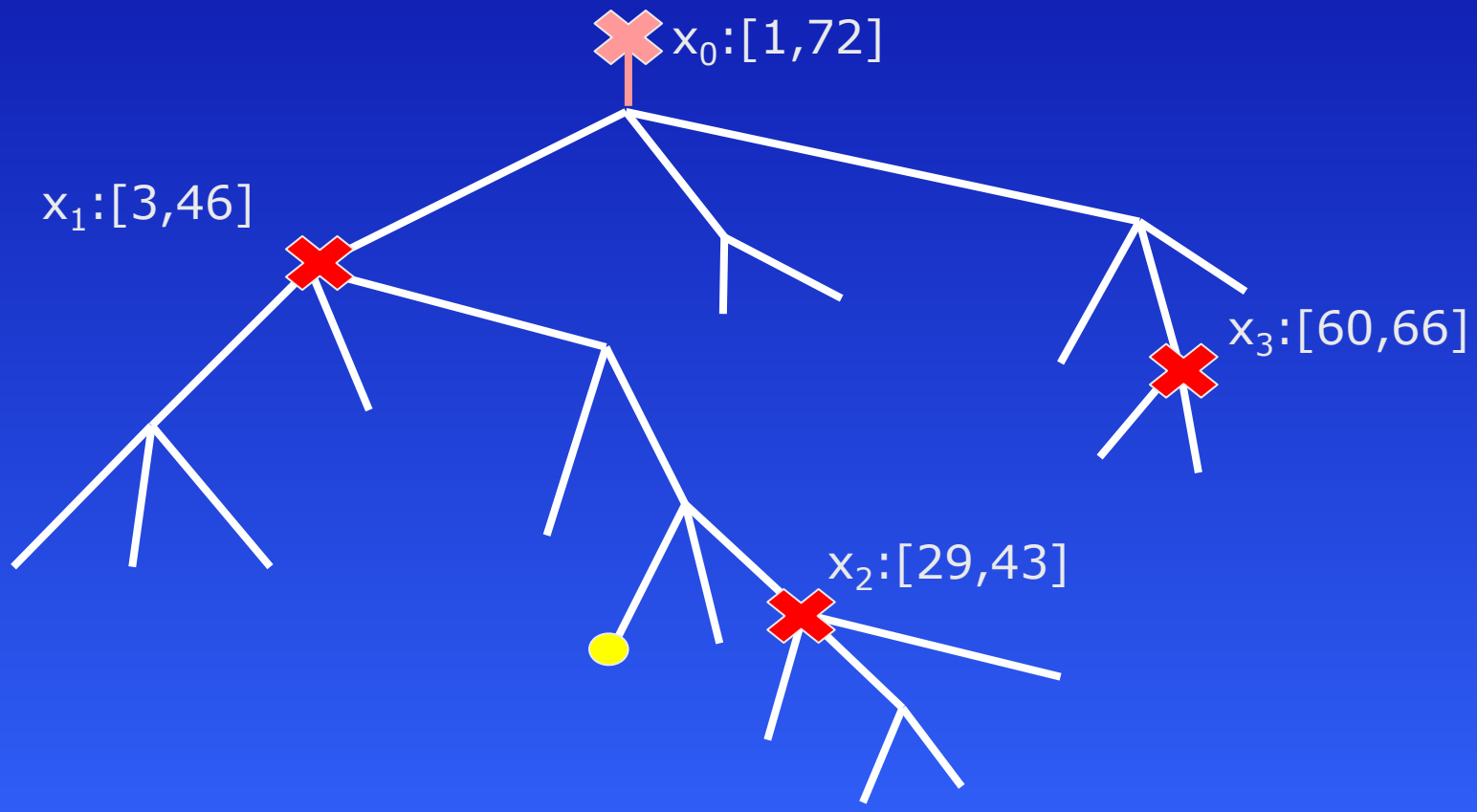
(find some identifier of the component of u in G\X)

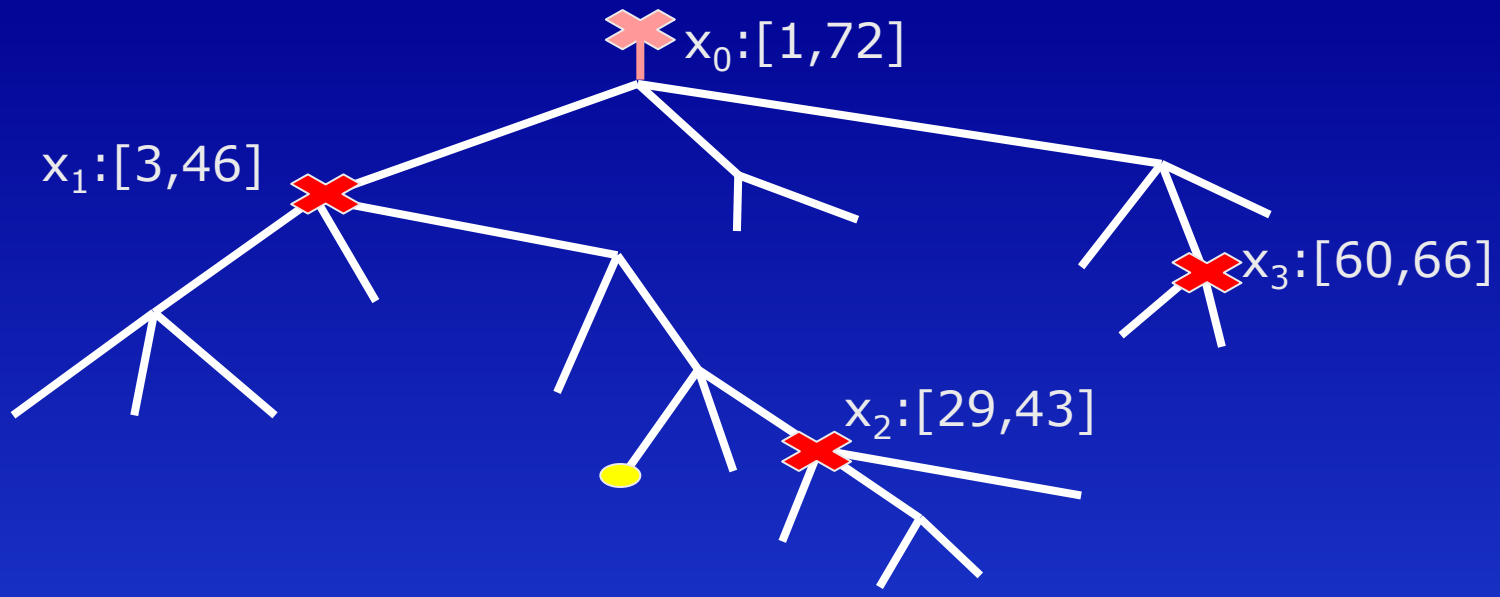
1. Find the closest failure x ancestor of u
2. Next-hop when routing from x to u in the tree

Label(z):=<[a(z),b(z)],@(z)>

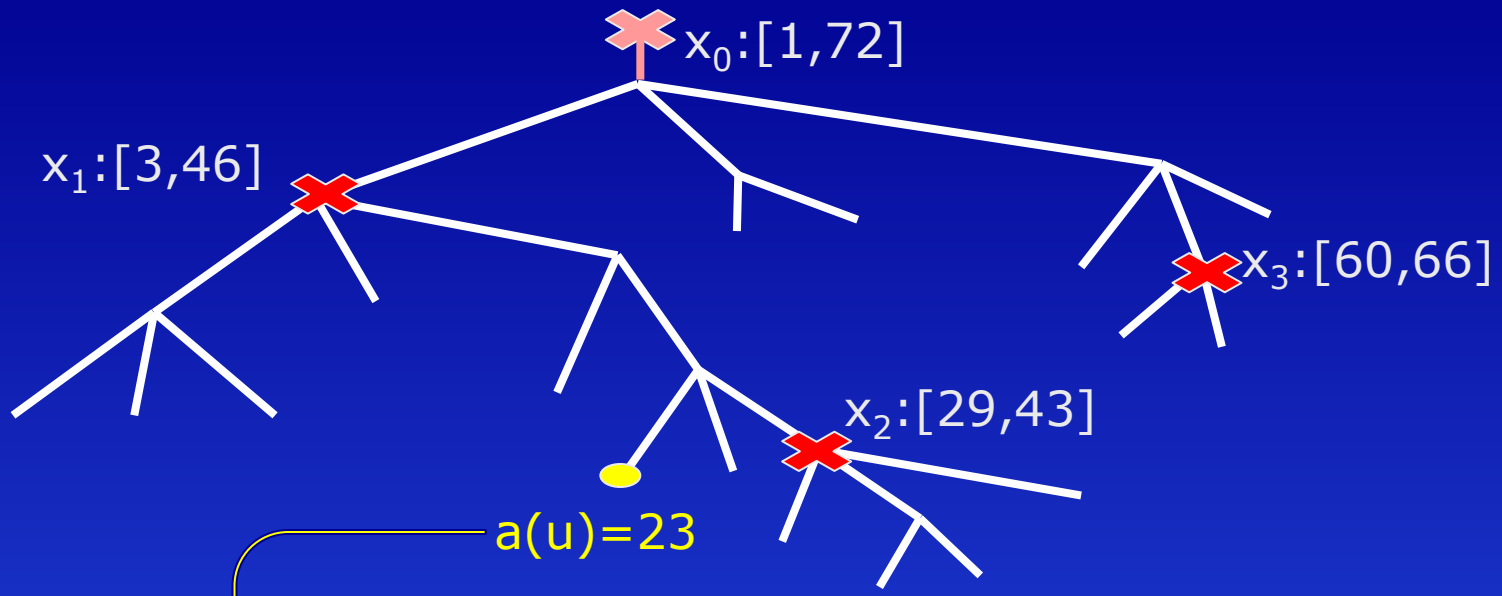
[a(z),b(z)]=first/last visit time in Euler tour

@(z)=routing label for routing to/from z





S=[1	[3	[29	43]	46]	[60	66]	72]
A= x_0	x_1	x_2	x_1	x_0	x_3	x_0	-



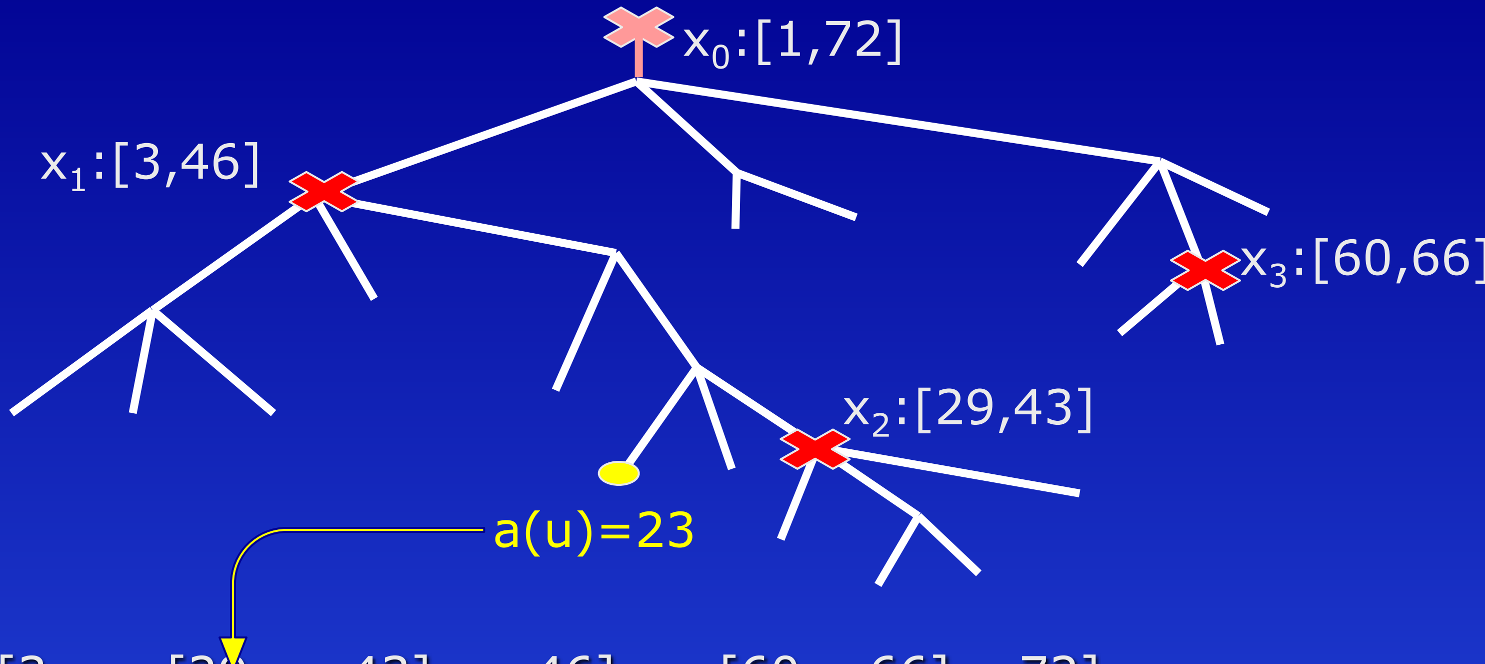
S=[1 [3 [29 43] 46] [60 66] 72]

A= x_0 x_1 x_2 x_1 x_0 x_3 x_0 -

$Find_{G\setminus X}(u):=(x,route(@(x),@(u)))$

x=A[p] (closest ancestor failure)

$p=PRED_S(a(u))=max\{s\in S:s\leq a(u)\}$ (predecessor)



S=[1	[3	[29	43]	46]	[60	66]	72]
A= x_0	x_1	x_2	x_1	x_0	x_3	x_0	-

- Find$_{G\setminus X}$(u)=(x, out(@(x),@(u)))
- Space: O(logn) bit labels

Query pre-processing: O(SORT(|X|,n))

x=A[p] (closest ancestor failure)

Query time: O(min{log|X|,loglogn})

p=PRED$_S$(a(u))=max{s∈S:s≤a(u)} (predecessor)

Query Time Lower Bound

Why $\Omega(\log\log n)$ is required? (for large $|X|$)

Given X and $\text{Find}_{\text{Path}\setminus X}$ we construct an associative table $\text{Tab}[\text{Find}_{\text{Path}\setminus X}(x_i+1)]=x_i$ in time $O(|X|\log|X|)$.

- $\text{PRED}_X(u)=\text{Tab}[\text{Find}_{\text{Path}\setminus X}(u)]$
- $\text{Query-time}(\text{PRED}_X) \le \text{Query-time}(\text{Find}_{\text{Path}\setminus X})+O(1)$

Query Time Lower Bound

Why $\Omega(\log\log n)$ is required? (for large $|X|$)

Given X and $\text{Find}_{\text{Path}\setminus X}$ we construct an associative table $\text{Tab}[\text{Find}_{\text{Path}\setminus X}(x_i+1)]=x_i$ in time $O(|X|\log|X|)$.

- $\text{PRED}_X(u)=\text{Tab}[\text{Find}_{\text{Path}\setminus X}(u)]$
- $\text{Query-time}(\text{PRED}_X) \le \text{Query-time}(\text{Find}_{\text{Path}\setminus X})+O(1)$

[Patrascu,Thorup – STOC'06]

Any data structure with space $\tilde{O}(|X|)$ and supporting PRED_X queries requires query time $\Omega(\log\log n)$ provided $|X|\in[n^{\varepsilon},n^{1-\varepsilon}]$.

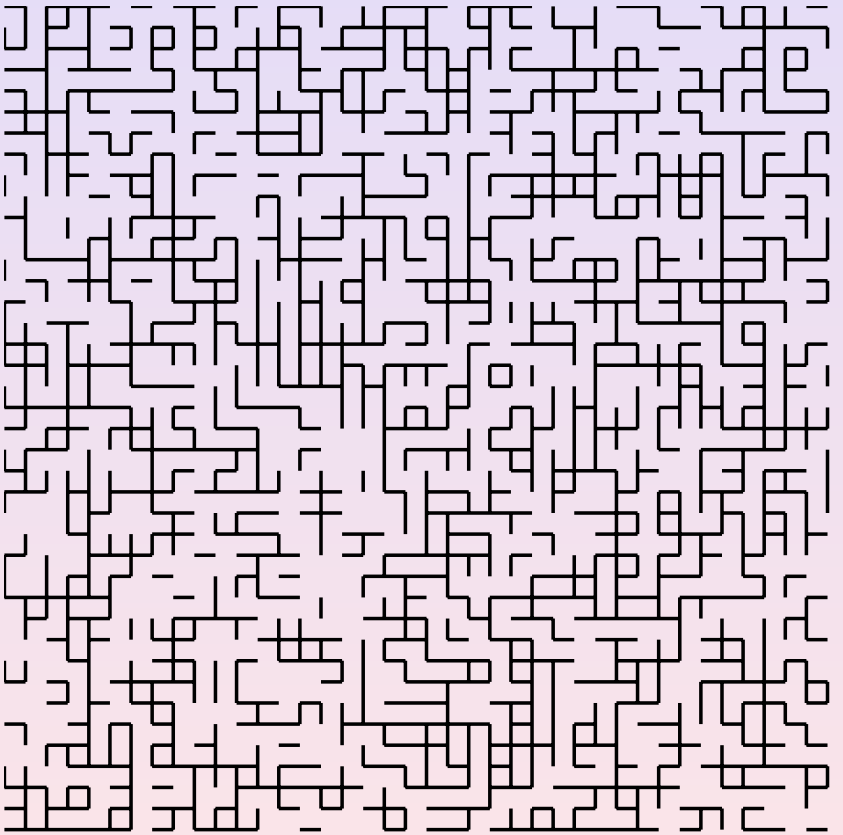
Agenda

1. Distance Labelling in General Graphs
2. Distance Labelling in Trees
3. Routing in Trees
4. Nearest Common Ancestor Labelling
5. Forbidden-Set Labelling
6. **Distance in Planar Graphs**
7. Distance in Minor-Free Graphs

Distance labeling in planar networks

Shortest path metrics of planar graphs are difficult to capture

- Planars are $\neq$ Euclidian networks (TSP, ℓ_p embedding, ...)
- Planars have no tree structure, treewidth can be $\Omega(\sqrt{n})$



Distance labeling in planar networks

Shortest path metrics of planar graphs are difficult to capture

- Planars are $\neq$ Euclidian networks (TSP, ℓ_p embedding, ...)
- Planars have no tree structure, treewidth can be $\Omega(\sqrt{n})$

Some history

stretch	label size (bits)	reference	
1	$n^{1/3} ... n^{1/2} \log \Delta$	G., Peleg et al .	[SODA '01]

Distance labeling in planar networks

Shortest path metrics of planar graphs are difficult to capture

- Planars are $\neq$ Euclidian networks (TSP, ℓ_p embedding, ...)
- Planars have no tree structure, treewidth can be $\Omega(\sqrt{n})$

Some history

stretch	label size (bits)	reference	
1	$n^{1/3}...n^{1/2}\log\Delta$	G., Peleg et al .	[SODA '01]
3	$n^{1/3}\log\Delta$	G., Peleg et al.	[ESA '01]

Distance labeling in planar networks

Shortest path metrics of planar graphs are difficult to capture

- Planars are $\neq$ Euclidian networks (TSP, ℓ_p embedding, ...)
- Planars have no tree structure, treewidth can be $\Omega(\sqrt{n})$

Some history

stretch	label size (bits)	reference	
1	$n^{1/3} ... n^{1/2} \log \Delta$	G., Peleg et al .	[SODA '01]
3	$n^{1/3} \log \Delta$	G., Peleg et al.	[ESA '01]
3	$\log n \log \Delta$	Gupta et al.	[SICOMP '05]

Distance labeling in planar networks

Shortest path metrics of planar graphs are difficult to capture

- Planars are $\neq$ Euclidian networks (TSP, ℓ_p embedding, ...)
- Planars have no tree structure, treewidth can be $\Omega(\sqrt{n})$

Some history

stretch	label size (bits)	reference	
1	$n^{1/3}...n^{1/2}\log\Delta$	G., Peleg et al .	[SODA '01]
3	$n^{1/3}\log\Delta$	G., Peleg et al.	[ESA '01]
3	$\log n\log\Delta$	Gupta et al.	[SICOMP '05]
$1+\varepsilon$	$\varepsilon^{-1}\log n\log\Delta$	Thorup	[JACM '04]

Shortest-path separator

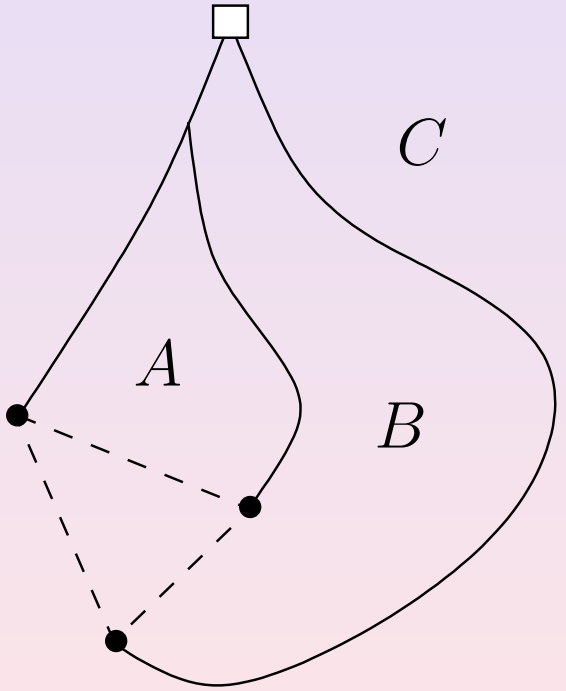
Lemma

Every n-node planar graph G has a shortest-path tree T with at most 3 leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

Shortest-path separator

Lemma

Every n-node planar graph G has a shortest-path tree T with at most 3 leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

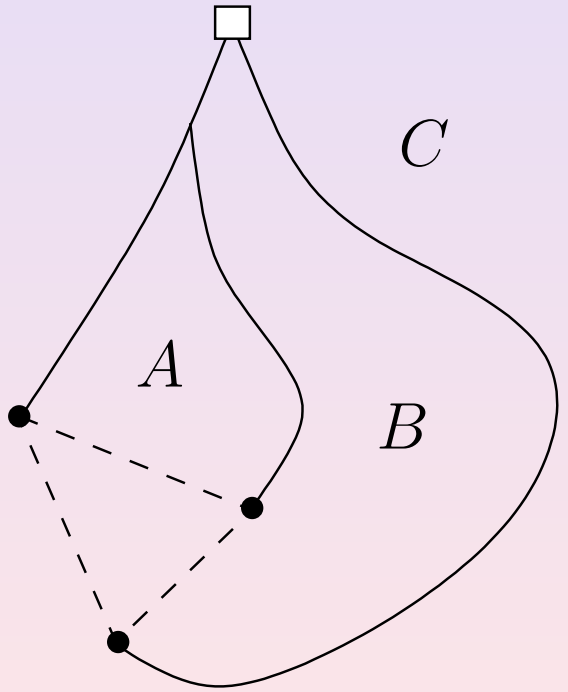


- $w(R) =$#nodes in region R

Shortest-path separator

Lemma

Every n-node planar graph G has a shortest-path tree T with at most 3 leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

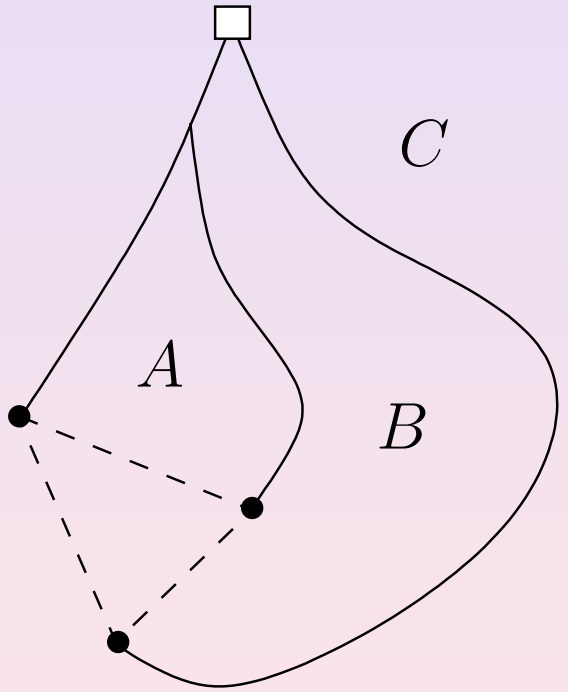


- $w(R) =$#nodes in region R
- $w(A), w(B), w(C) \leqslant n/2$: done

Shortest-path separator

Lemma

Every n-node planar graph G has a shortest-path tree T with at most 3 leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

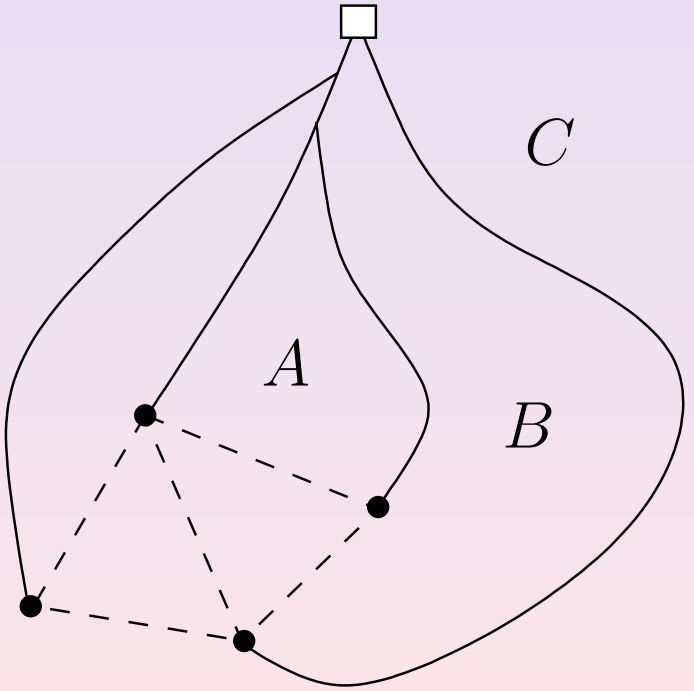


- $w(R) =$#nodes in region R
- $w(A), w(B), w(C) \leqslant n/2$: done
- or $w(C) > n/2$

Shortest-path separator

Lemma

Every n-node planar graph G has a shortest-path tree T with at most 3 leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

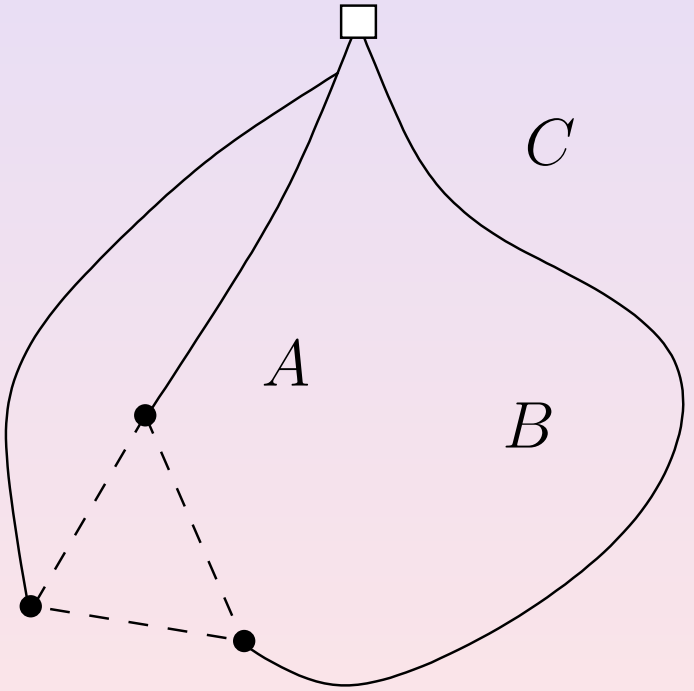


- $w(R) =$#nodes in region R
- $w(A), w(B), w(C) \leqslant n/2$: done
- or $w(C) > n/2$
- $w(A) + w(B) \leqslant n/2$

Shortest-path separator

Lemma

Every n-node planar graph G has a shortest-path tree T with at most 3 leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

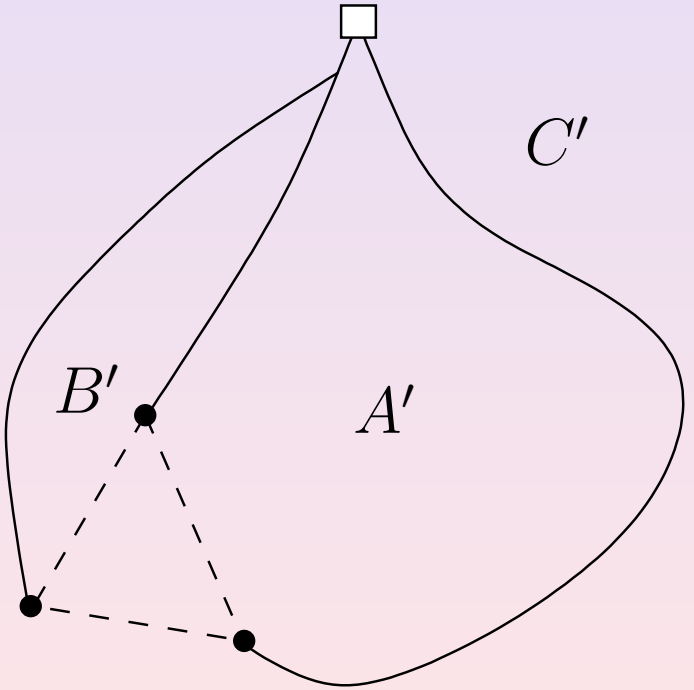


- $w(R) =$#nodes in region R
- $w(A), w(B), w(C) \leqslant n/2$: done
- or $w(C) > n/2$
- $w(A) + w(B) \leqslant n/2$
- merge & repeat

Shortest-path separator

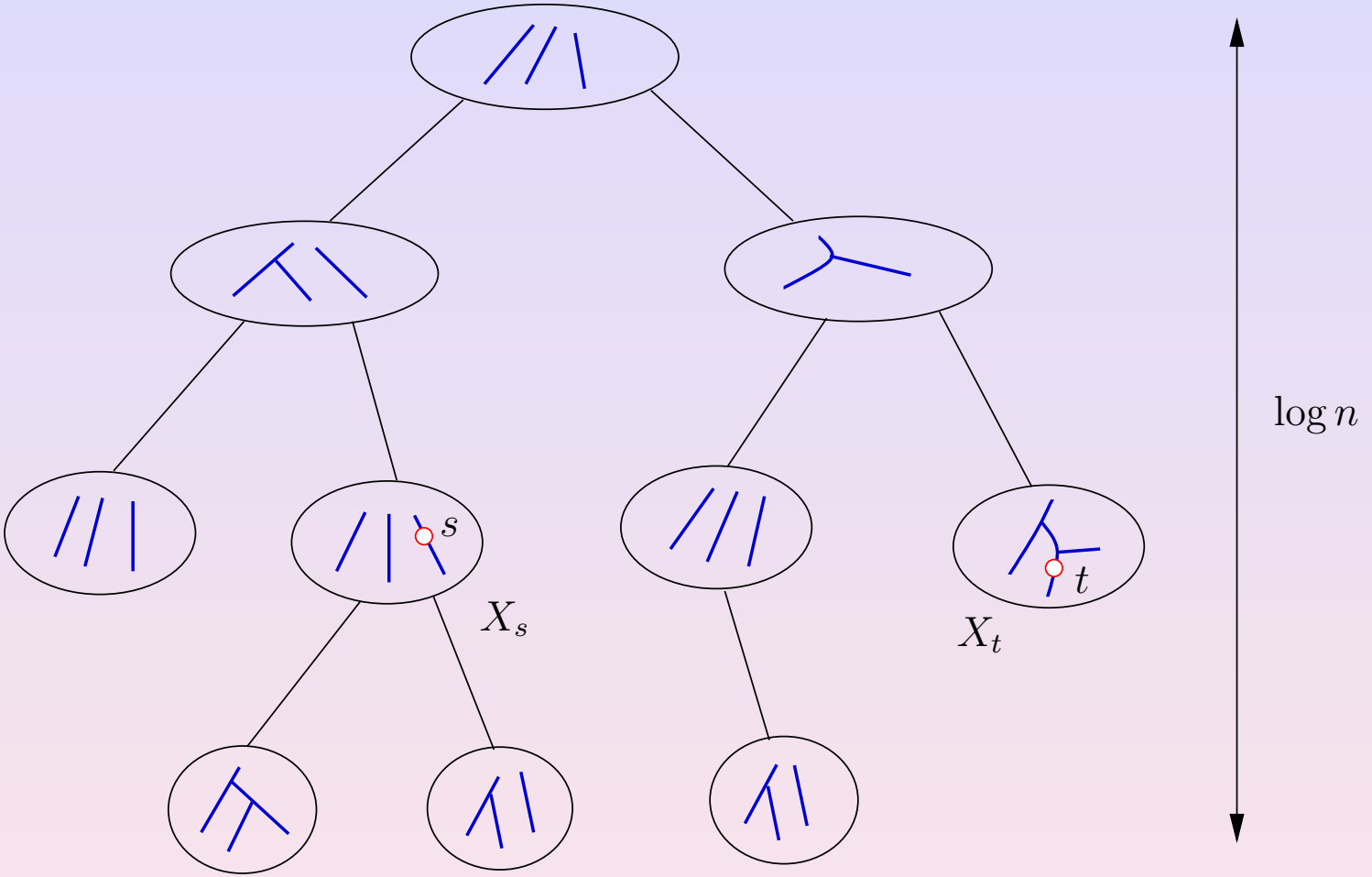
Lemma

Every n-node planar graph G has a shortest-path tree T with at most 3 leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.



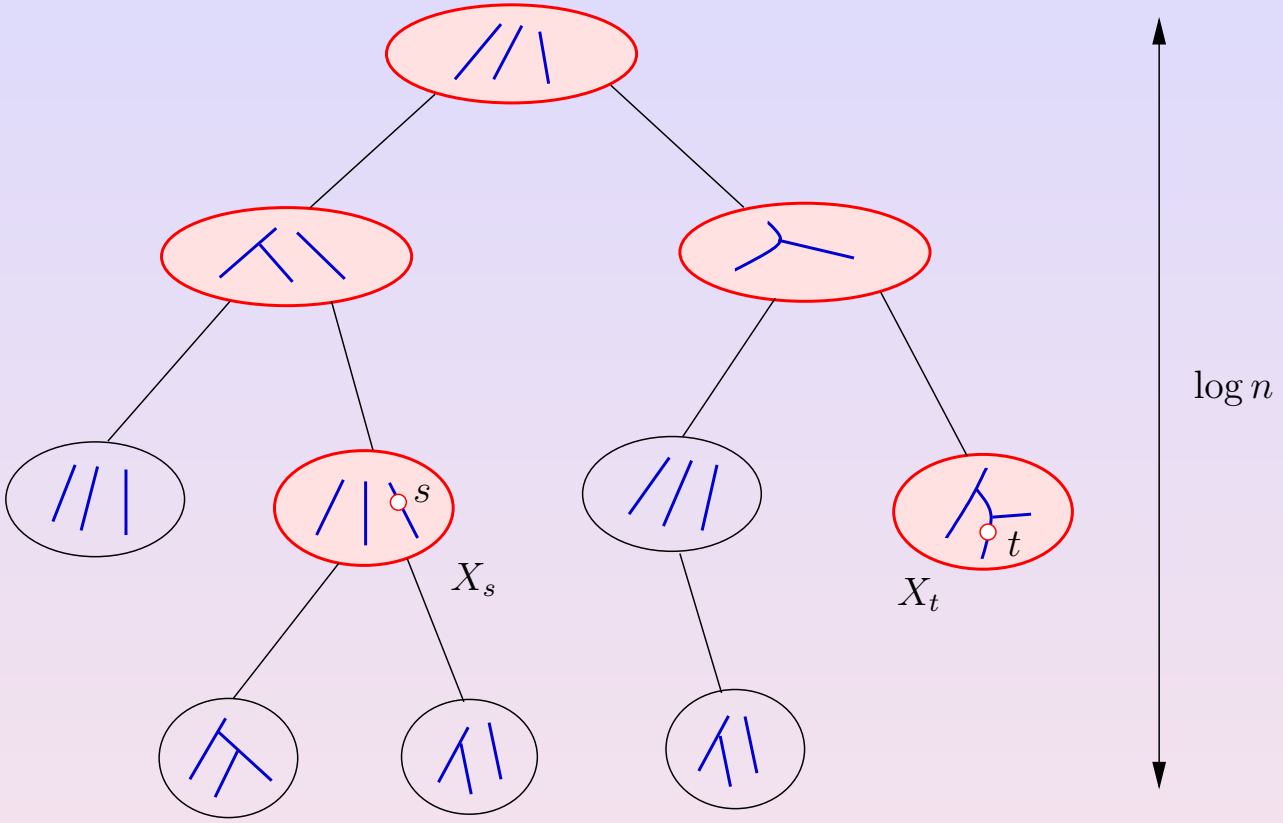
- $w(R) =$#nodes in region R
- $w(A), w(B), w(C) \leqslant n/2$: done
- or $w(C) > n/2$
- $w(A) + w(B) \leqslant n/2$
- merge & repeat
- $w(C') < w(C)$

Shortest-path tree-decomposition



$X_u =$ highest separator where node u belongs to

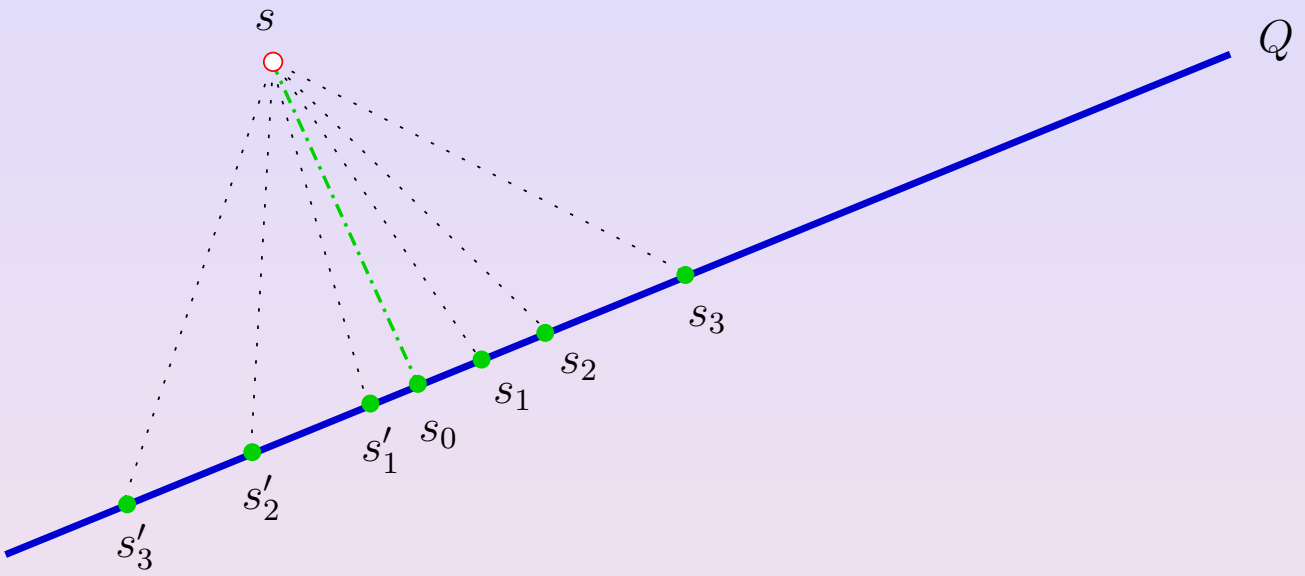
Shortest-path tree-decomposition



Let P be *any* path $s \to t$.

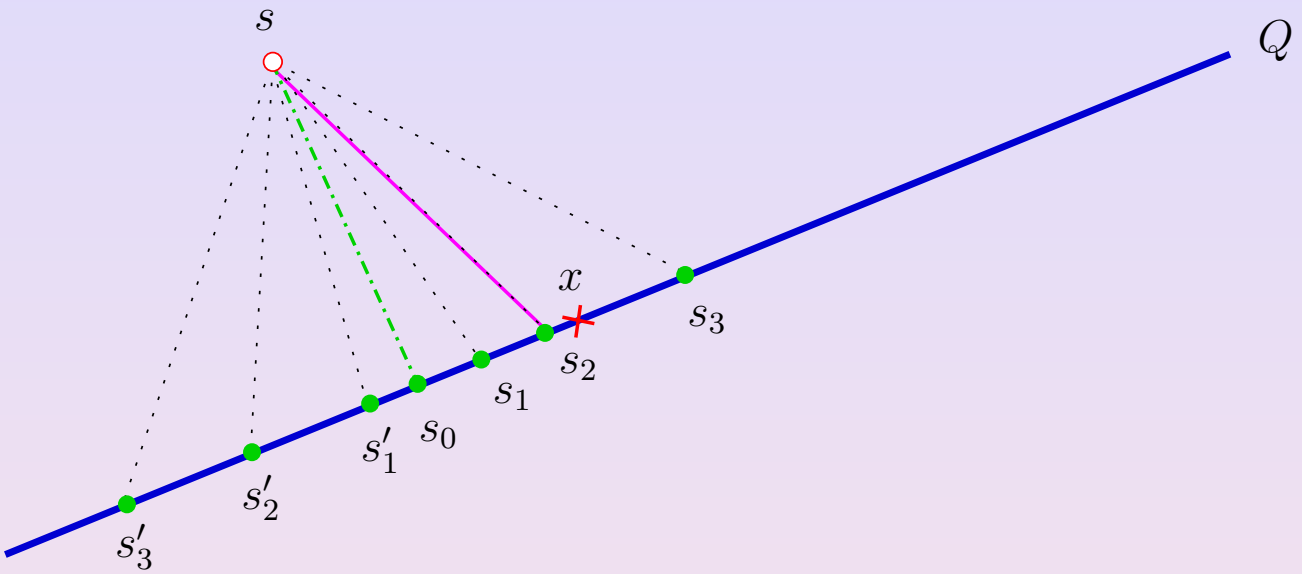
Property 1. $\exists Q \in \mathrm{ancestor(X_s)} \cap \mathrm{ancestor(X_t)}$, $Q \cap P \neq \varnothing$.

Landmark and ε-cover



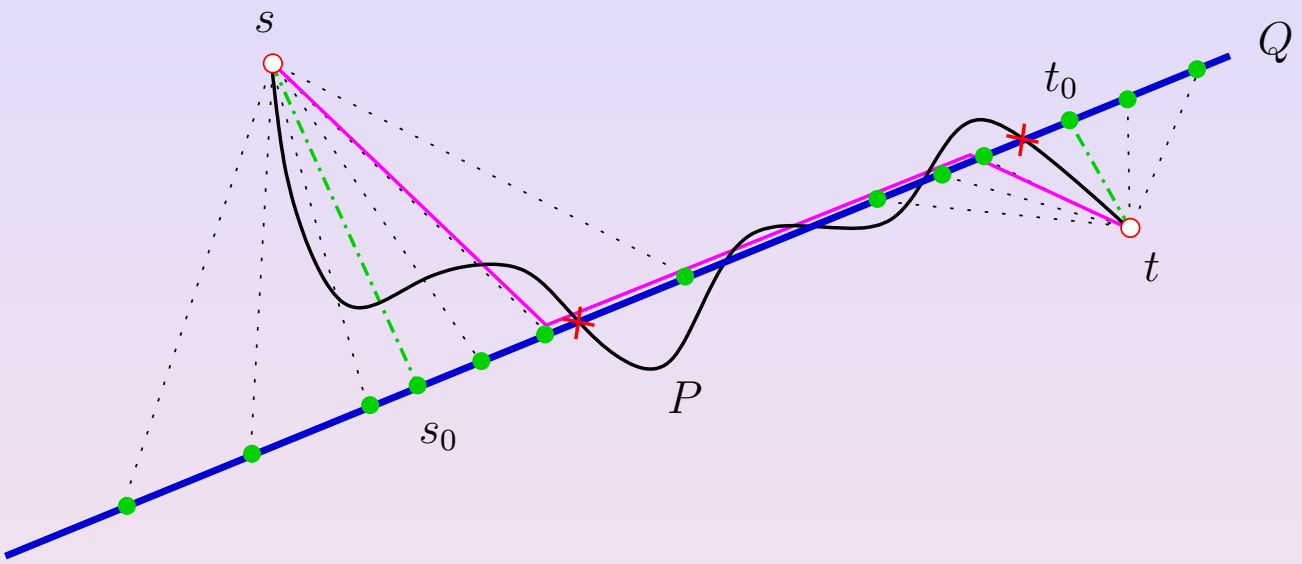
Property 2. For every s and Q, at most $1 + 4/\varepsilon$ landmarks suffices to ε-cover every $x \in Q$, i.e.,
$d_G(s, s_i) + d_Q(s_i, x) \leqslant (1 + \varepsilon) \cdot d_G(s, x)$ for some landmark s_i.

Landmark and ε-cover



Property 2. For every s and Q, at most $1 + 4/\varepsilon$ landmarks suffices to ε-cover every $x \in Q$, i.e.,
$d_G(s, s_i) + d_Q(s_i, x) \leqslant (1 + \varepsilon) \cdot d_G(s, x)$ for some landmark s_i.

Landmark and ε-cover



Property 3. If Q intersects P, then there are landmarks s_i, t_j st. $d_G(s, s_i) + d_Q(s_i, t_j) + d_G(t_j, t) \leqslant (1 + \varepsilon) \cdot d_G(s, t)$.

Conclusion

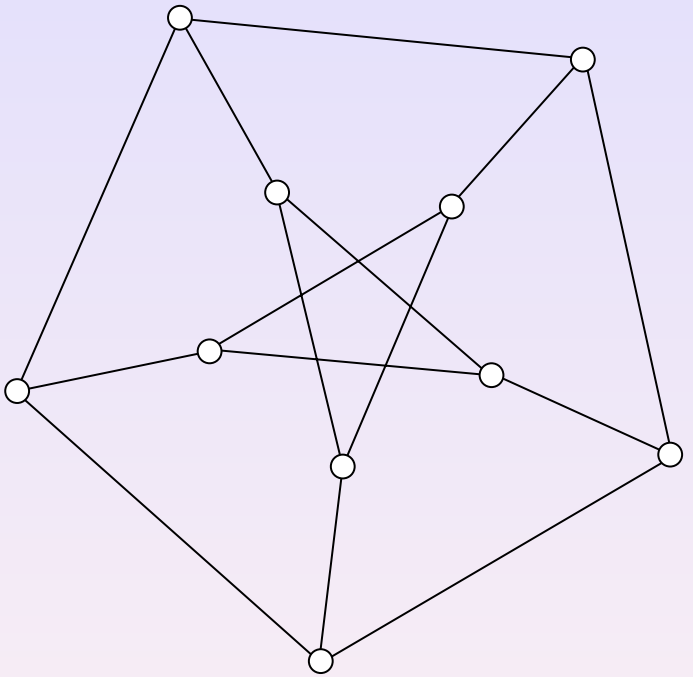
Properties 1, 2 & 3 show that each node only needs to store at most $(1 + 4/\varepsilon) \times 3 \log n$ distances informations, that is $O(\varepsilon^{-1} \log n \log \Delta)$ bits per node label, to $(1 + \varepsilon)$-approximate any s, t-distance in the graph. $\square$

Agenda

1. Distance Labelling in General Graphs
2. Distance Labelling in Trees
3. Routing in Trees
4. Nearest Common Ancestor Labelling
5. Forbidden-Set Labelling
6. Distance in Planar Graphs
7. **Distance in Minor-Free Graphs (what is a minor?)**

What is a minor?

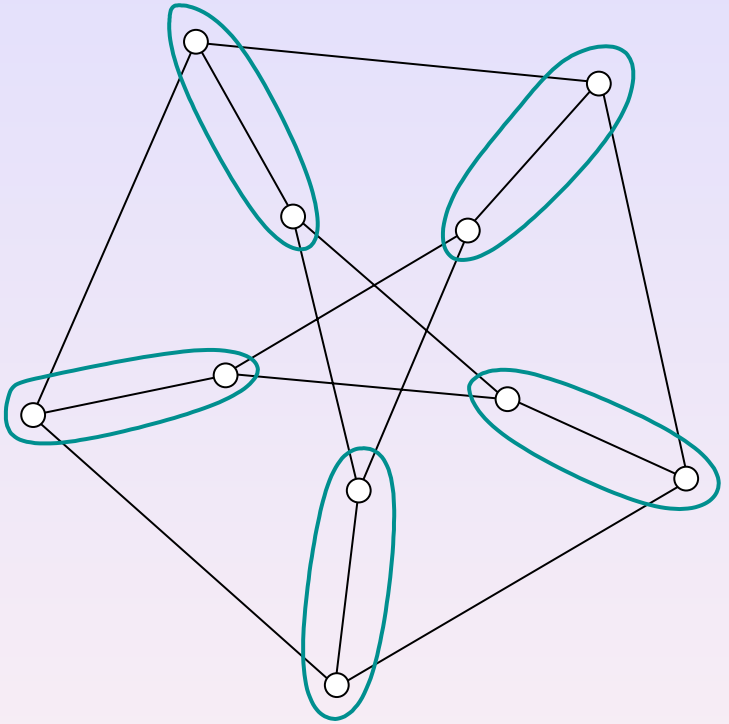
A **minor** of G is a subgraph of a graph obtained from G by edge contraction.



A H-minor free graph is a graph without minor H.

What is a minor?

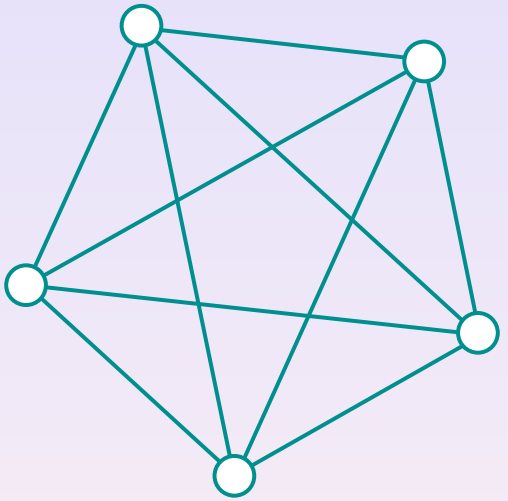
A **minor** of G is a subgraph of a graph obtained from G by edge contraction.



A H-minor free graph is a graph without minor H.

What is a minor?

A **minor** of G is a subgraph of a graph obtained from G by edge contraction.

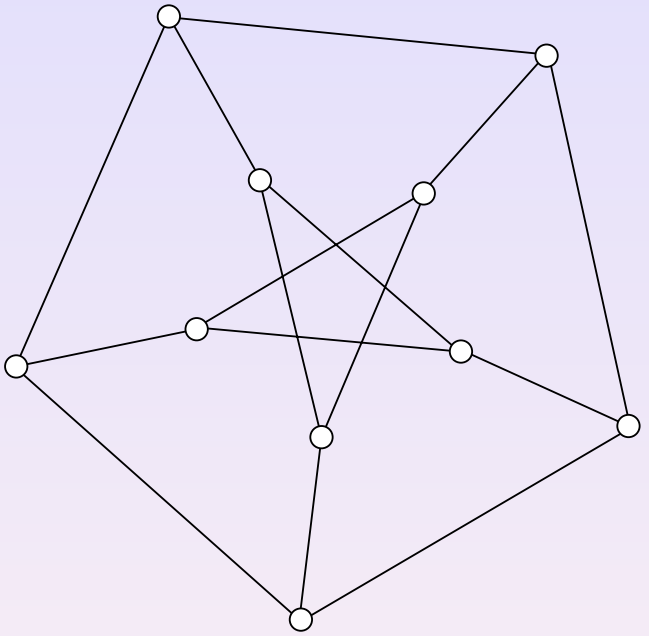


A H-minor free graph is a graph without minor H.

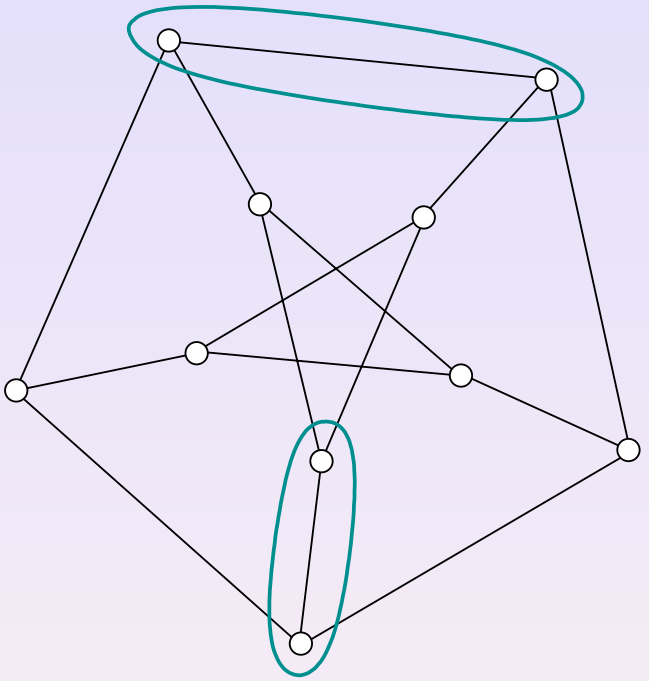
Some H-minor free graph families

- Trees are K_3-minor free
- Outerplanar graphs are $K_{2,3}$-minor free
- Series-Parallel graphs are K_4-minor free
- Planar are K_5-minor free (and also $K_{3,3}$-minor free)
- Genus-g graphs are $K_{5+\lfloor 2\sqrt{3g} \rfloor}$-minor free
- Treewidth-t graphs are K_{t+2}-minor free
- The graphs of any minor closed families $\mathcal{F}$ are H-minor free for some $H = H(\mathcal{F})$.

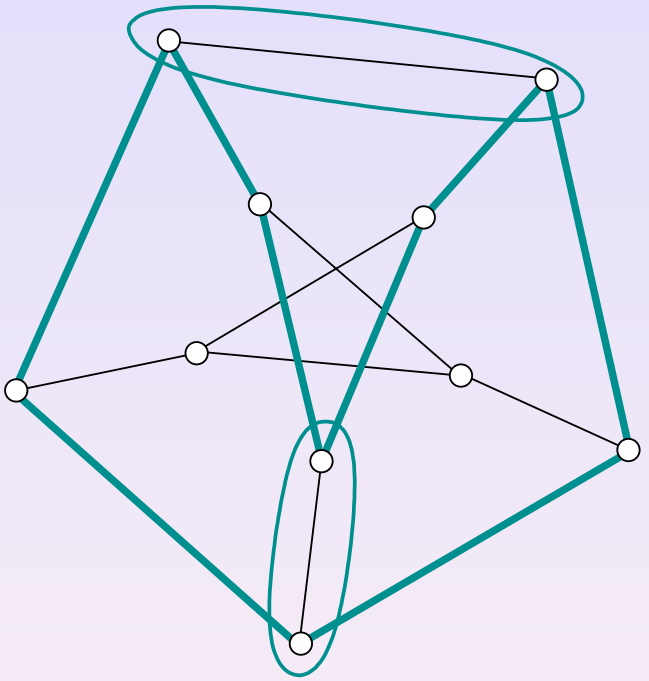
Excercise: is there a $K_{2,4}$-minor?



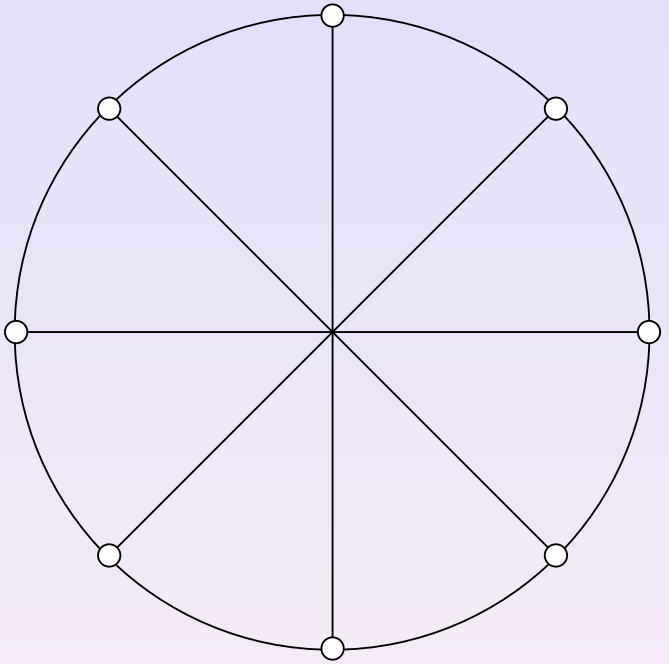
Excercise: is there a $K_{2,4}$-minor?



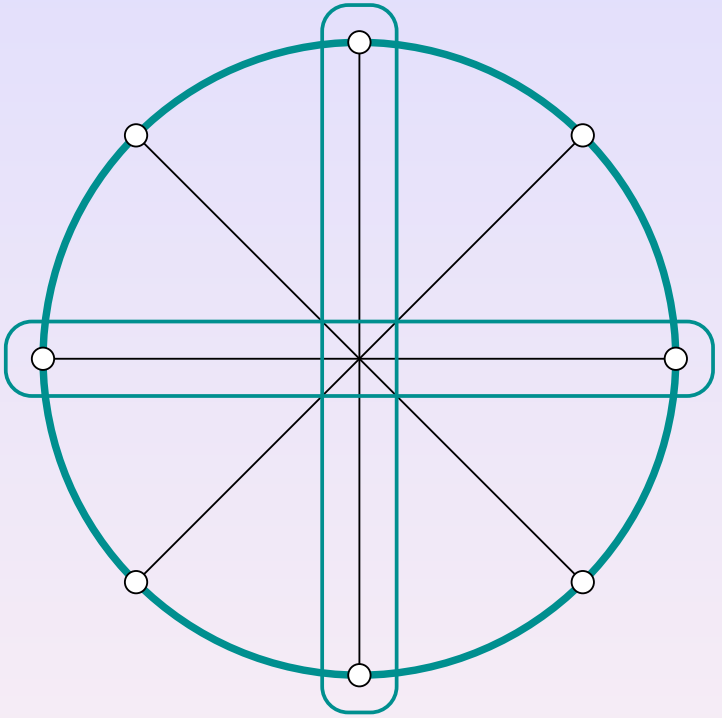
Excercise: is there a $K_{2,4}$-minor?



Excercise: is there a $K_{2,4}$-minor?



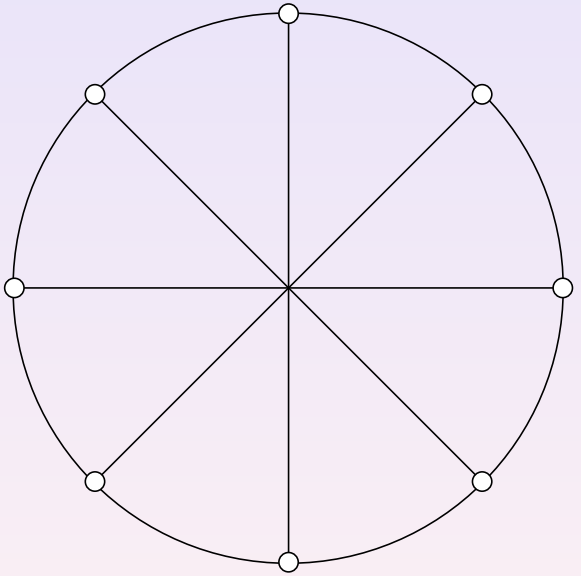
Excercise: is there a $K_{2,4}$-minor?



K_5-minor free graphs

Theorem (Wagner - 1937)

Every K_5-minor free graph has a tree-decomposition whose bags intersect in at most 3 vertices, and induced a planar graph or a V_8.



Corollary: 4-coloring of K_5-minor free graphs $\Leftrightarrow$ 4CC.

H-minor free graphs

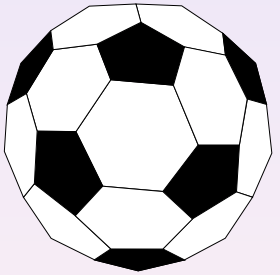
Theorem (Robertson & Seymour - Graph Minor 16)

Every H-minor free graph has a tree-decomposition whose bags intersect in $\leqslant k$ vertices, and induced graphs that either have $\leqslant k$ vertices, or are k-almost embeddable on a surface Σ on which H has no embedding.

H-minor free graphs

Theorem (Robertson & Seymour - Graph Minor 16)

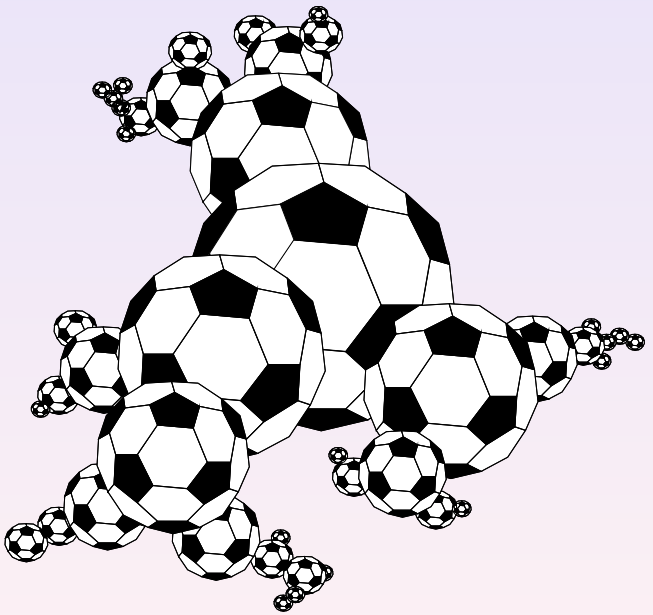
Every H-minor free graph has a tree-decomposition whose bags intersect in $\leqslant k$ vertices, and induced graphs that either have $\leqslant k$ vertices, or are k-almost embeddable on a surface Σ on which H has no embedding.



H-minor free graphs

Theorem (Robertson & Seymour - Graph Minor 16)

Every H-minor free graph has a tree-decomposition whose bags intersect in $\leqslant k$ vertices, and induced graphs that either have $\leqslant k$ vertices, or are k-almost embeddable on a surface Σ on which H has no embedding.



Wagner's Theorem : $H = K_5$, $k = 3$, $\Sigma = \mathbb{S}_0$.

Problem: the constant!

The structure given by [RS] Theorem is not fine enough for practical use. No bounds on $k = k(H)$ is known!

Problem: the constant!

The structure given by [RS] Theorem is not fine enough for practical use. No bounds on $k = k(H)$ is known!

Bounds if H is planar [RST '94]: the tree-decomposition has width $k = k(H)$, and thus has a $(k+1)$-coloring.

Problem: $k \leqslant 20^{2(|V(H)|+4|E(H)|)^5}$ ...

Problem: the constant!

The structure given by [RS] Theorem is not fine enough for practical use. No bounds on $k = k(H)$ is known!

Bounds if H is planar [RST '94]: the tree-decomposition has width $k = k(H)$, and thus has a $(k+1)$-coloring.

Problem: $k \leqslant 20^{2(|V(H)|+4|E(H)|)^5}$ ...

So excluding $H = K_4$ leads to treewidth of at most 400^{28^5}. In fact this is 2, such graphs are series-parallel. They have a 3-coloring.

K_6-minor free: conjectures

Conjecture (Hadwiger - 1943)

Every K_{r+1}-minor free graph has a r-coloring.

Proved for $r \in \{1, \ldots, 5\}$.

K_6-minor free: conjectures

Conjecture (Hadwiger - 1943)

Every K_{r+1}-minor free graph has a r-coloring.

Proved for $r \in \{1, \ldots, 5\}$.

[Robertson et al. - 1993] ($r = 5$)

5-coloring of K_6-minor free graphs $\Leftrightarrow$ 4CC

[Every minimal counter-example is
a planar plus one vertex (83 pages)]

However, the structure of K_6-minor free graph is still unknown. Ken-ichi Kawarabayashi explains in SODA '07 why the problem is important and difficult.

K_6-minor free: conjectures

Conjecture (Jørgensen - 2001)

Every K_6-minor free graph has a arboricity at most 3.

K_6-minor free: conjectures

Conjecture (Jørgensen - 2001)

Every K_6-minor free graph has a arboricity at most 3.

Conjecture (Jørgensen - 1994)

Every 6-connected K_6-minor free graph has a vertex u such that $G \setminus \{u\}$ is planar.

DeVos, Hegde, Kawarabayashi, Norine, Thomas, and Wollan have announced that [J94] is true if the graph has many vertices ...

Problem: replace "6" by "r" in [J94].

Agenda

1. Distance Labelling in General Graphs
2. Distance Labelling in Trees
3. Routing in Trees
4. Nearest Common Ancestor Labelling
5. Forbidden-Set Labelling
6. Distance in Planar Graphs
7. **Distance in Minor-Free Graphs (what is a minor?)**

Our approach: path separator technique

Definition (Main)

A weighted graph G with n nodes is *k-path separable* if there exists a subgraph S, called k-path separator, such that:

Our approach: path separator technique

Definition (Main)

A weighted graph G with n nodes is *k-path separable* if there exists a subgraph S, called k-path separator, such that:

1. $S = P_0 \cup P_1 \cup \cdots$, where each subgraph P_i is the union of k_i shortest paths in $G \setminus \bigcup_{j<i} P_j$;

Our approach: path separator technique

Definition (Main)

A weighted graph G with n nodes is *k-path separable* if there exists a subgraph S, called k-path separator, such that:

1. $S = P_0 \cup P_1 \cup \cdots$, where each subgraph P_i is the union of k_i shortest paths in $G \setminus \bigcup_{j<i} P_j$;
2. $\sum_i k_i \leqslant k$; and

Our approach: path separator technique

Definition (Main)

A weighted graph G with n nodes is *k-path separable* if there exists a subgraph S, called k-path separator, such that:

1. $S = P_0 \cup P_1 \cup \cdots$, where each subgraph P_i is the union of k_i shortest paths in $G \setminus \bigcup_{j<i} P_j$;
2. $\sum_i k_i \leqslant k$; and
3. each connected component of $G \setminus S$ is k-path separable and has at most $n/2$ nodes.

Our approach: path separator technique

Definition (Main)

A weighted graph G with n nodes is *k-path separable* if there exists a subgraph S, called k-path separator, such that:

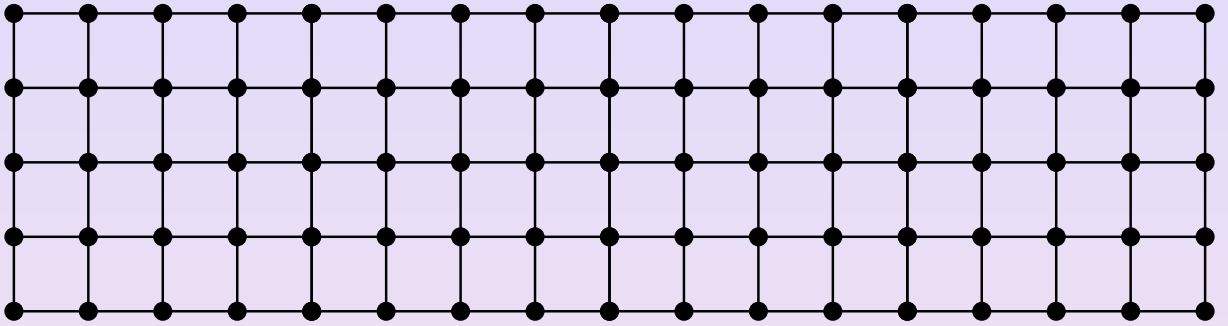
1. $S = P_0 \cup P_1 \cup \cdots$, where each subgraph P_i is the union of k_i shortest paths in $G \setminus \bigcup_{j<i} P_j$;
2. $\sum_i k_i \leqslant k$; and
3. each connected component of $G \setminus S$ is k-path separable and has at most $n/2$ nodes.

If Q is a path forming P_i, then:

- Q is not necessarily of bounded size
- Q is not necessarily a shortest path in G

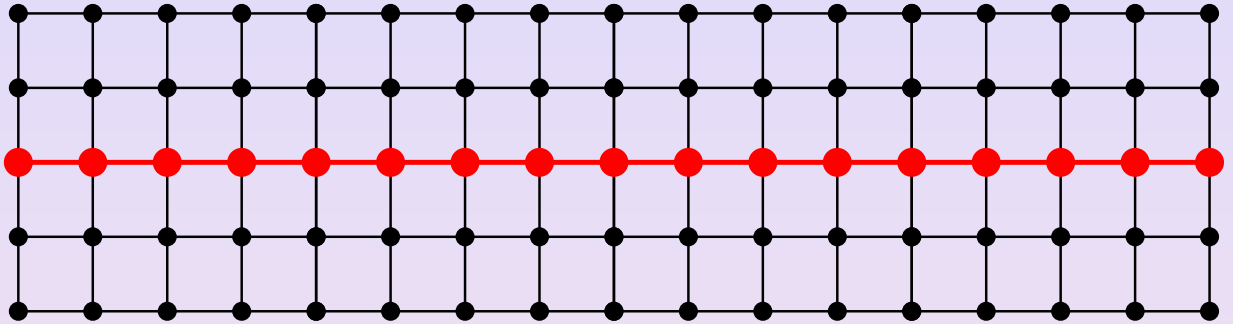
Some basic examples

- Unweighted meshes are 1-path separable



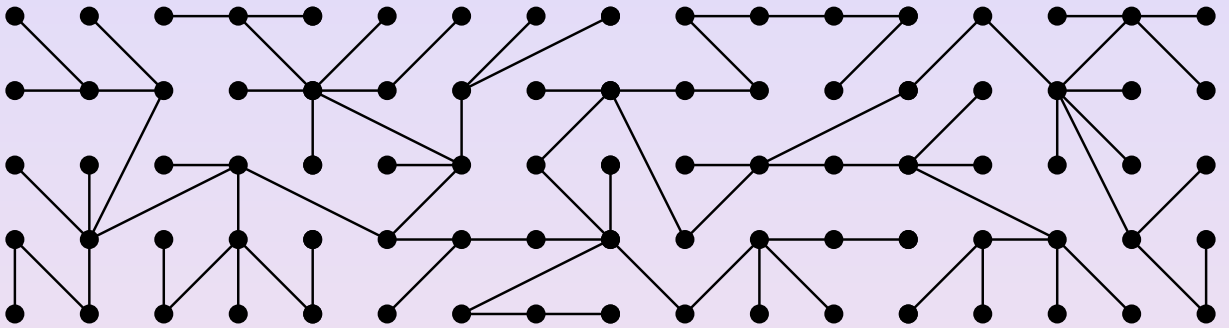
Some basic examples

- Unweighted meshes are 1-path separable



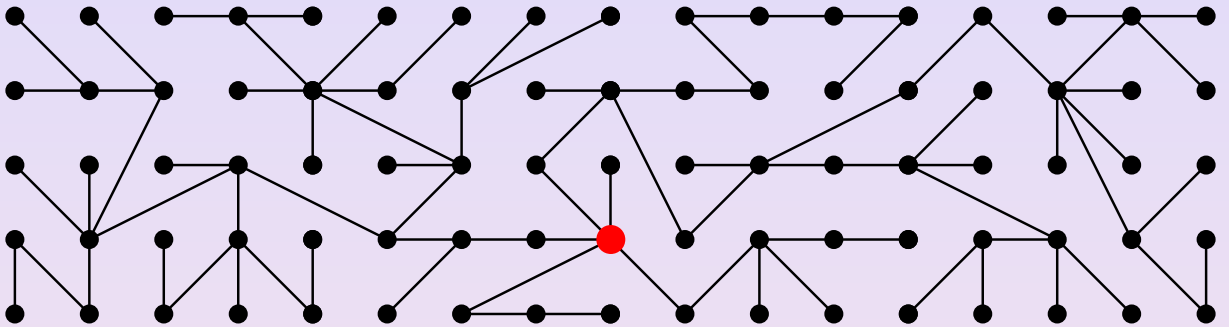
Some basic examples

- Unweighted meshes are 1-path separable
- Trees are 1-path separable



Some basic examples

- Unweighted meshes are 1-path separable
- Trees are 1-path separable



Some basic examples

- Unweighted meshes are 1-path separable
- Trees are 1-path separable

Lemma (Thorup [JACM '04])

Every n-node planar graph G has a shortest-path tree T with at most 3 leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

$\Rightarrow$ planars are 3-path separable

Conjecture (Thorup)

Every n-node H-minor-free graph has a shortest-path tree T with at most $\ell = \ell(H)$ leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

Conjecture (Thorup)

Every n-node H-minor-free graph has a shortest-path tree T with at most $\ell = \ell(H)$ leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

True for $H = K_2, K_3, K_4, K_5$, also true if H is planar

Conjecture (Thorup)

Every n-node H-minor-free graph has a shortest-path tree T with at most $\ell = \ell(H)$ leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

True for $H = K_2, K_3, K_4, K_5$, also true if H is planar

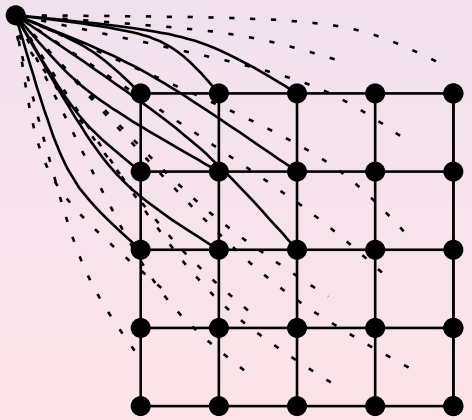
Wrong for K_6! There are K_6-minor-free graphs for which a **sequence** of **unions** of shortest paths is required!

Conjecture (Thorup)

Every n-node H-minor-free graph has a shortest-path tree T with at most $\ell = \ell(H)$ leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

True for $H = K_2, K_3, K_4, K_5$, also true if H is planar

Wrong for K_6! There are K_6-minor-free graphs for which a **sequence** of **unions** of shortest paths is required!



- genus $\Omega(n)$
- tree-width $\Omega(\sqrt{n})$
- no K_6 minor
- $\Omega(\sqrt{n})$ shortest paths to halve
- ... but is 2-path separable

Conjecture (Thorup)

Every n-node H-minor-free graph has a shortest-path tree T with at most $\ell = \ell(H)$ leaves such that each component of $G \setminus T$ has $\leqslant n/2$ nodes.

True for $H = K_2, K_3, K_4, K_5$, also true if H is planar

Wrong for K_6! There are K_6-minor-free graphs for which a **sequence** of **unions** of shortest paths is required!

Theorem (Main)

Every H-minor-free graph is k-path separable for $k = k(H)$.

A k-path separator can be find in $n^{O(k)}$ time

Consequences of the Main Theorem

Theorem (Object Location)

Let G be a weighted k-path separable graph of aspect ratio Δ

1. *stretch-$(1+\varepsilon)$ distance labeling with $O(k\varepsilon^{-1}\log n \log(\varepsilon^{-1}\log\Delta))$-bit labels*
2. *stretch-$(1+\varepsilon)$ labeled routing scheme with $O(k\varepsilon^{-1}\log^3 n/\log\log n)$-bit headers and routing tables*
3. *One can augment G with 1 directed edge per node such that greedy routing performs in $O(k^2\log^2 n\log^2\Delta)$ expected number of hops*
4. *And others: reachability, distance oracles in digraphs, ...*

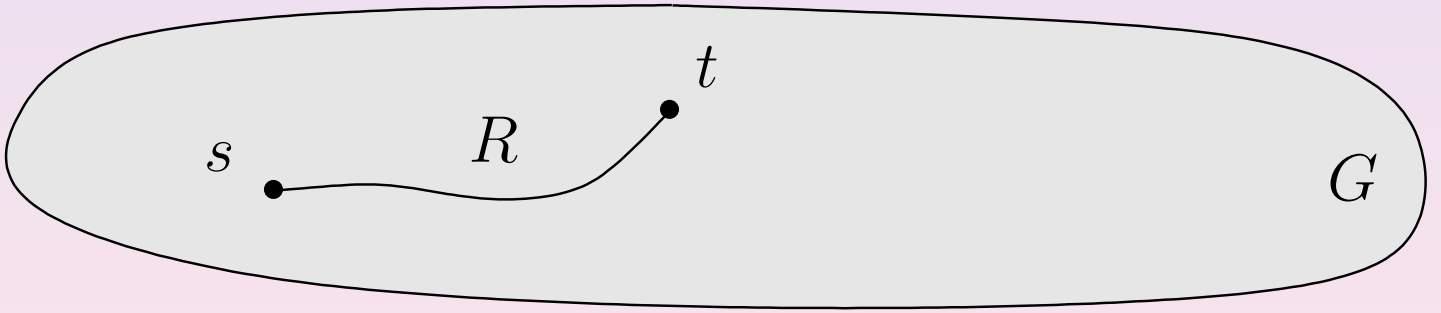
Proving the Object Location Theorem

(≈ extension of Thorup's data-structures)

$\forall$ s, t-shortest path R in G there exist:

- a subgraph G' in the separator decomposition of G;
- a k-path separator S' of G'; and
- a path Q that composes S' such that

Q and R intersect and **both** are shortest paths in G'.



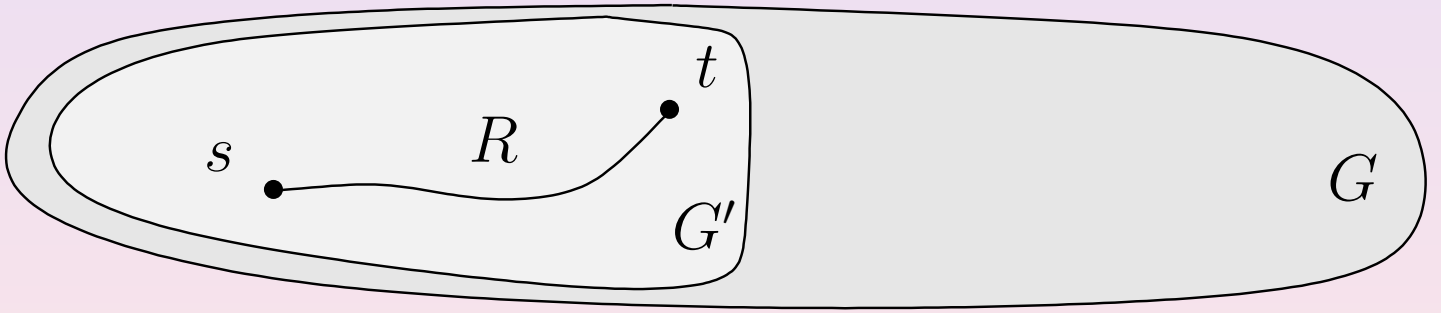
Proving the Object Location Theorem

(≈ extension of Thorup's data-structures)

$\forall$ s, t-shortest path R in G there exist:

- a subgraph G' in the separator decomposition of G;
- a k-path separator S' of G'; and
- a path Q that composes S' such that

Q and R intersect and **both** are shortest paths in G'.

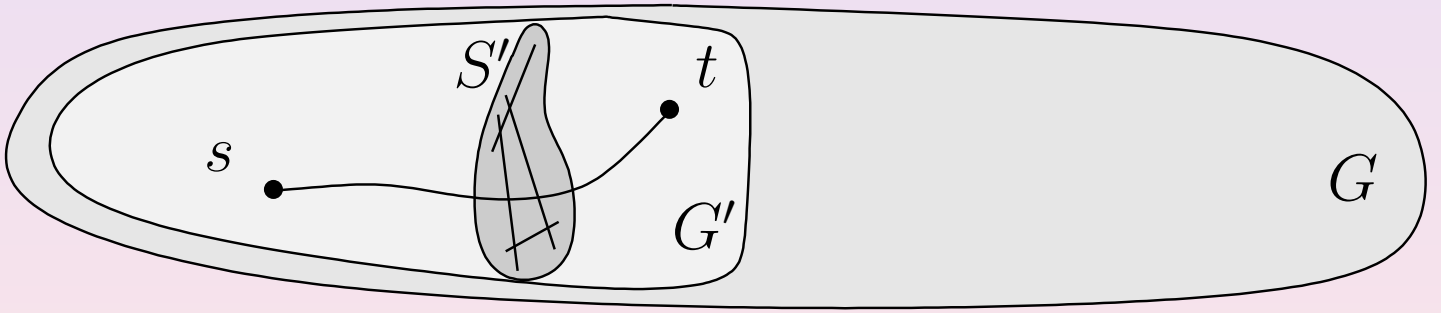


Proving the Object Location Theorem
(≈ extension of Thorup's data-structures)

$\forall$ s, t-shortest path R in G there exist:

- a subgraph G' in the separator decomposition of G;
- a k-path separator S' of G'; and
- a path Q that composes S' such that

Q and R intersect and **both** are shortest paths in G'.

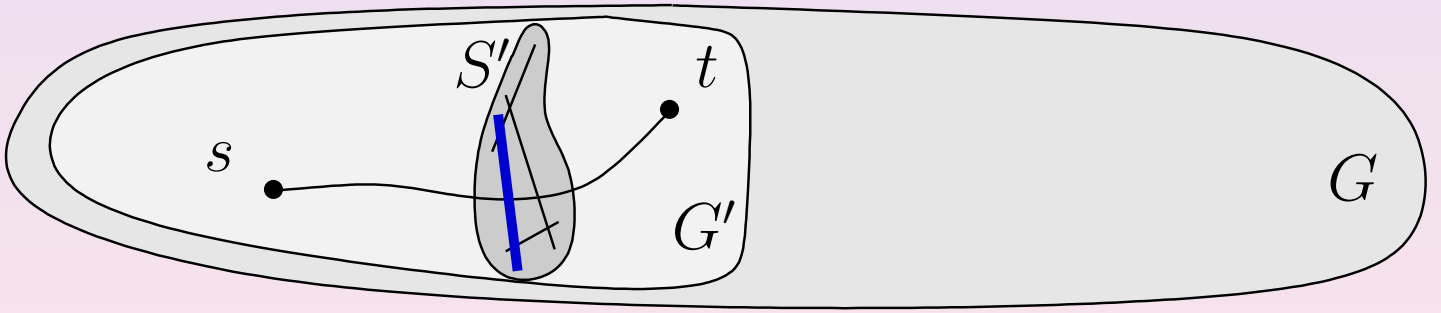


Proving the Object Location Theorem
(≈ extension of Thorup's data-structures)

$\forall$ s, t-shortest path R in G there exist:

- a subgraph G' in the separator decomposition of G;
- a k-path separator S' of G'; and
- a path Q that composes S' such that

Q and R intersect and **both** are shortest paths in G'.

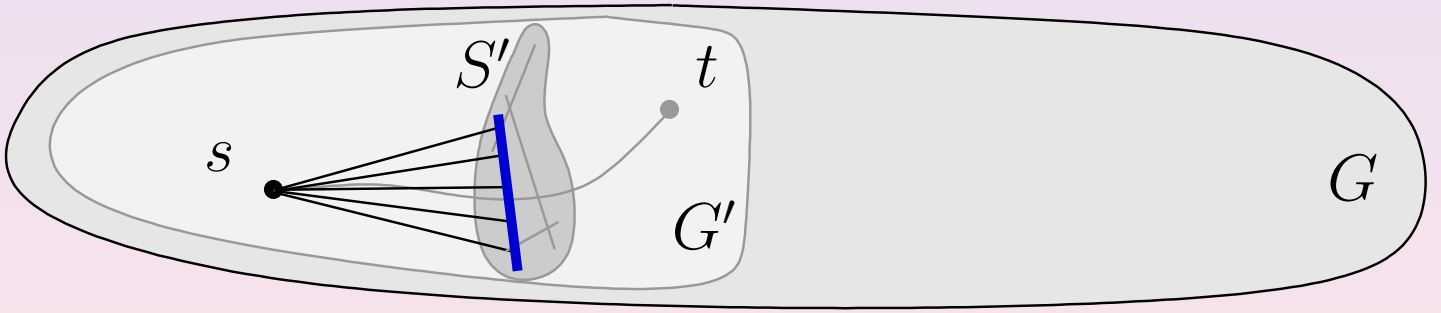


Proving the Object Location Theorem
(≈ extension of Thorup's data-structures)

$\forall$ s, t-shortest path R in G there exist:

- a subgraph G' in the separator decomposition of G;
- a k-path separator S' of G'; and
- a path Q that composes S' such that

Q and R intersect and **both** are shortest paths in G'.



Node s can select, **independently** of t, few "landmarks" on Q so that one of these landmarks is close to $R \cap Q$

Proving the Main Theorem

Theorem (Main)

Every H-minor-free graph is k-path separable for $k = k(H)$.

Proving the Main Theorem

Theorem (Main)

Every H-minor-free graph is k-path separable for $k = k(H)$.

The full proof is technical (needs long preliminaries), based on the recent decomposition theorem of Robertson & Seymour:

Proving the Main Theorem

Theorem (Main)

Every H-minor-free graph is k-path separable for $k = k(H)$.

The full proof is technical (needs long preliminaries), based on the recent decomposition theorem of Robertson & Seymour:

Roughly speaking,

Theorem (Graph Minor-16, 2003)

Every graph excluding a fixed minor has a tree-decomposition in subgraphs that are h-almost embeddable on a surface of bounded Euler genus.

Proving the Main Theorem

Theorem (Main)

Every H-minor-free graph is k-path separable for $k = k(H)$.

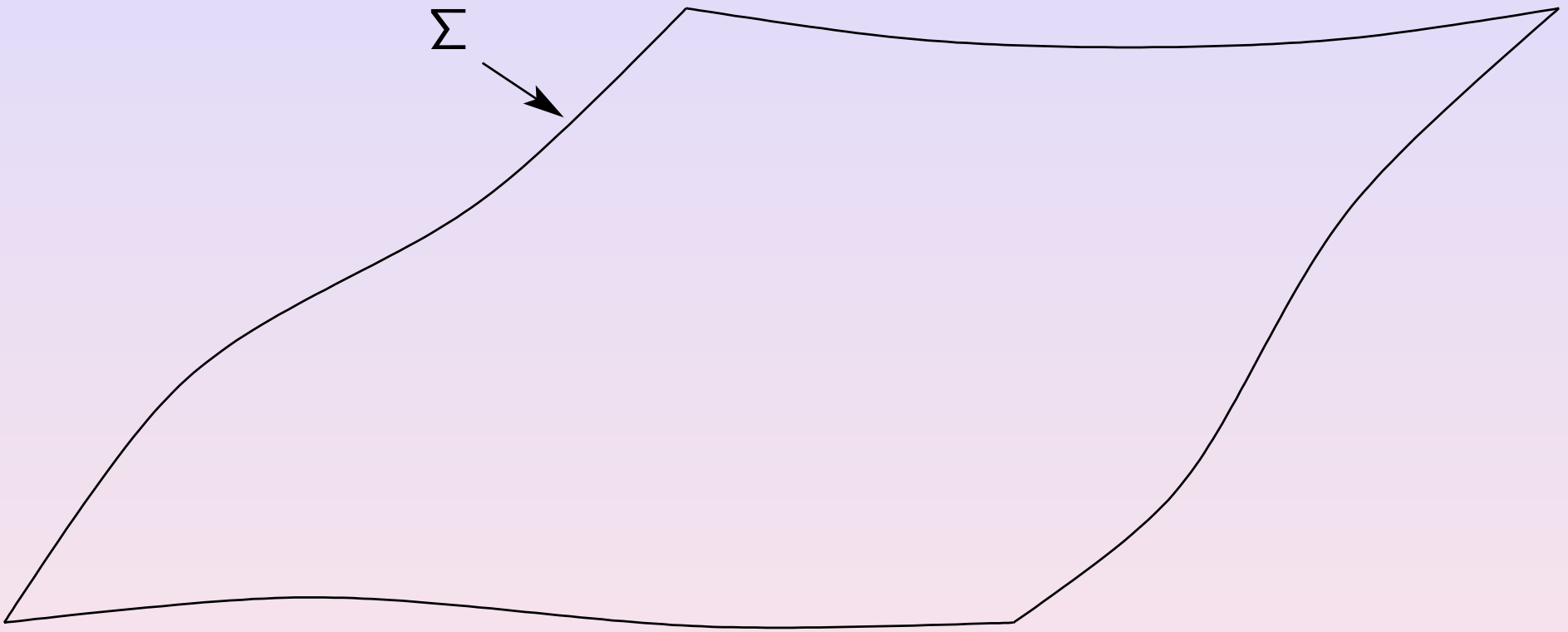
The full proof is technical (needs long preliminaries), based on the recent decomposition theorem of Robertson & Seymour:

Actually,

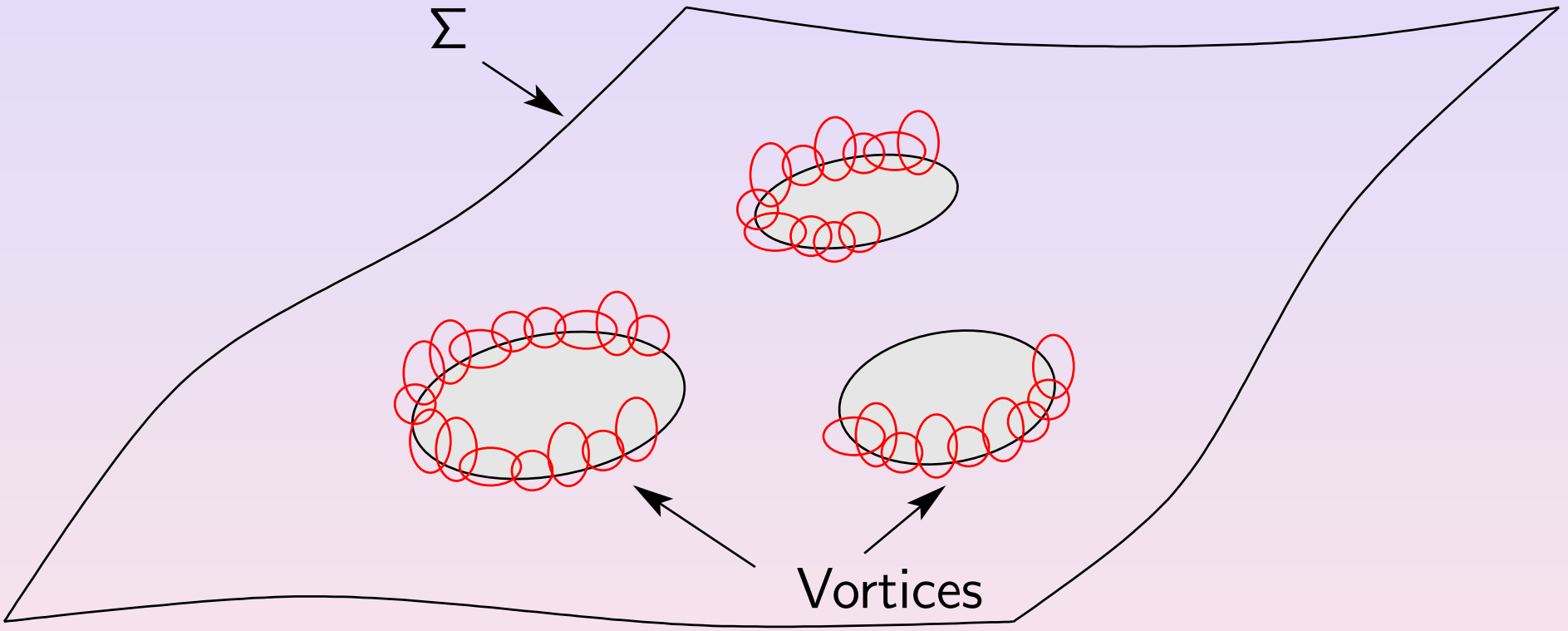
Theorem (Graph Minor-16, 2003)

Every graph excluding a minor H has a tree-decomposition whose the "torso" of its bags are h-almost embeddable on a surface on which H cannot be embedded.

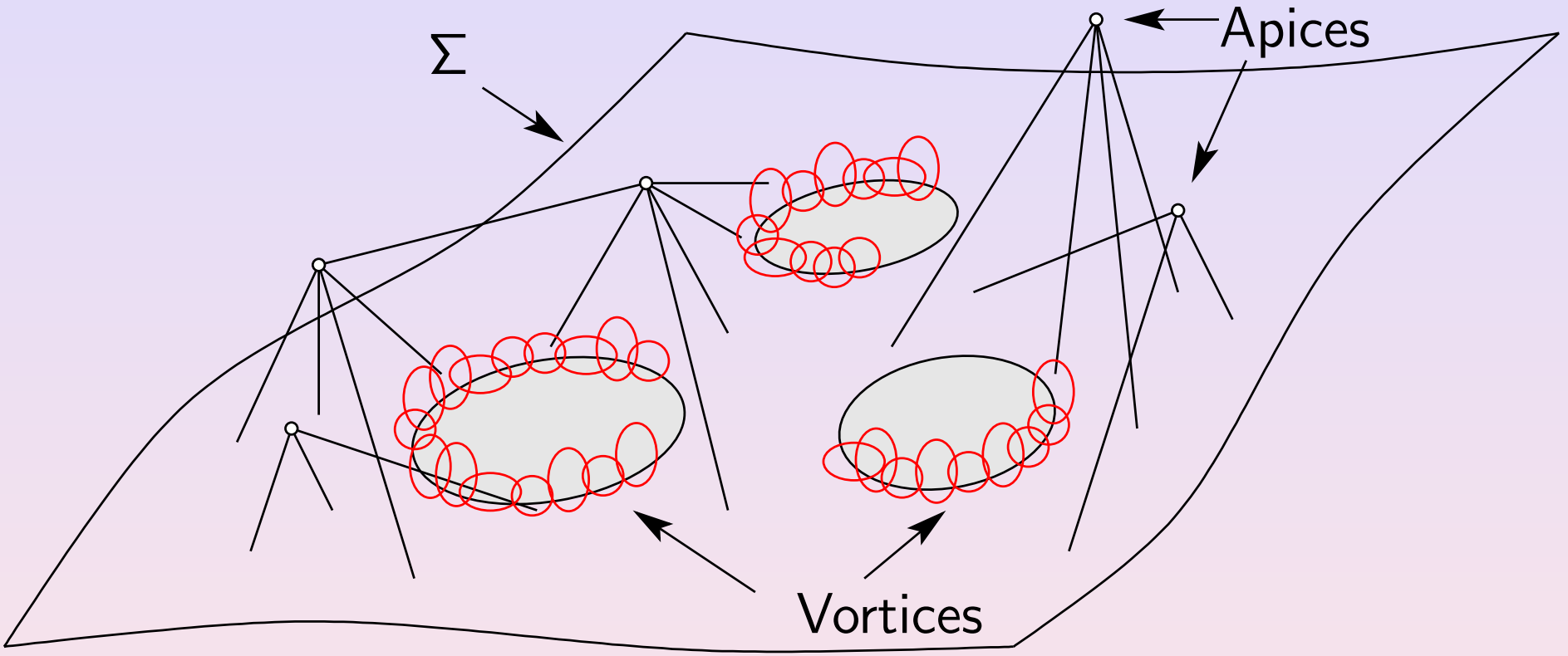
h-almost embeddable graphs



h-almost embeddable graphs

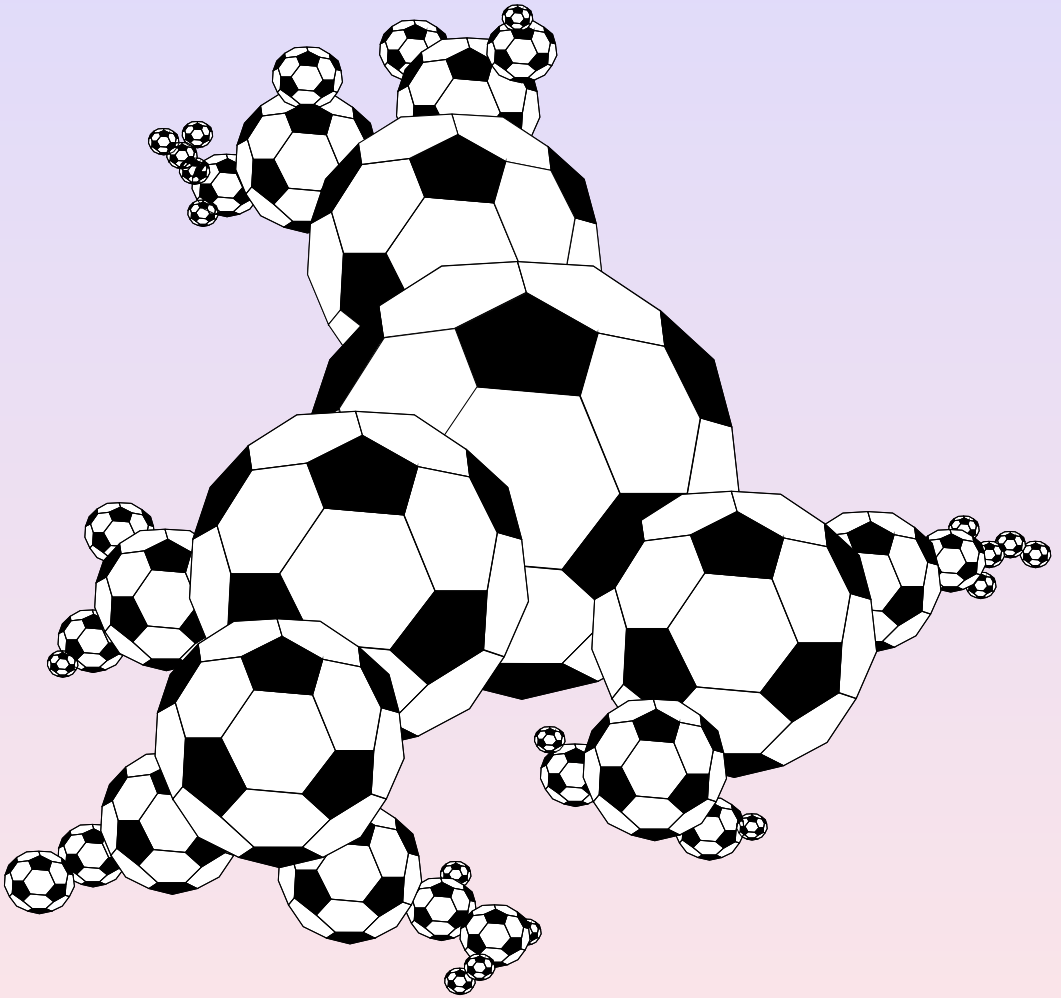


h-almost embeddable graphs

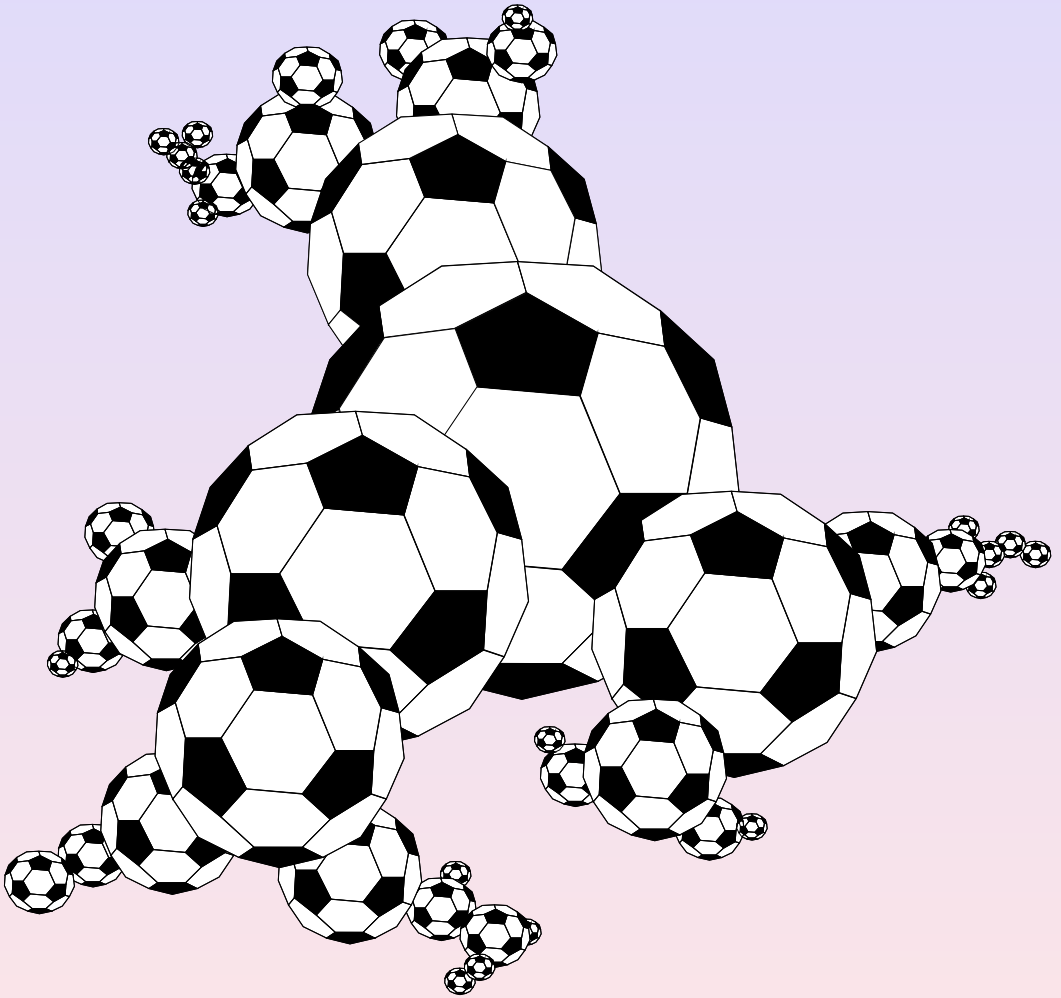


A tree of h-almost embeddable graphs

A tree of h-almost embeddable graphs



Finding a k-path separator with this?



Finding a k-path separator with this?

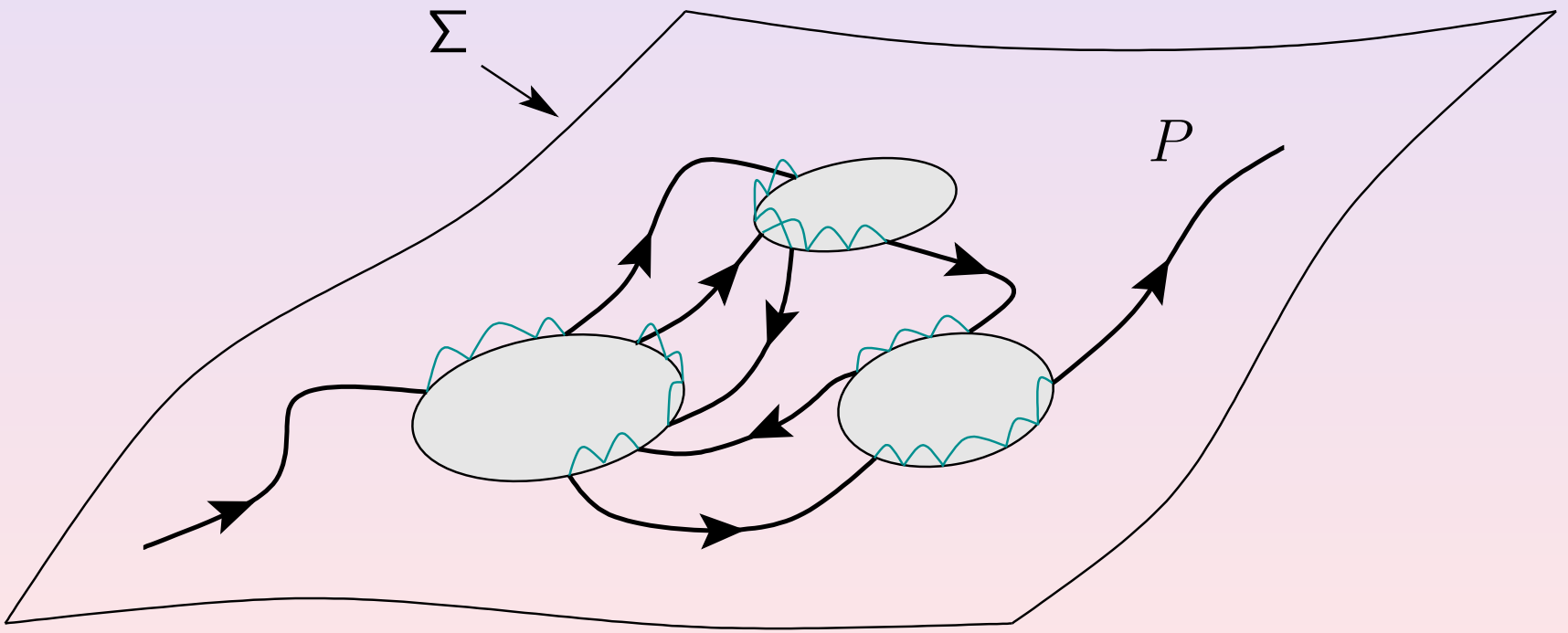
Some remarks:

- shortest paths go everywhere
- Σ can be non-orientable
- Jordan curve Theorem does not work (vortices!)

Finding a k-path separator with this?

Some remarks:

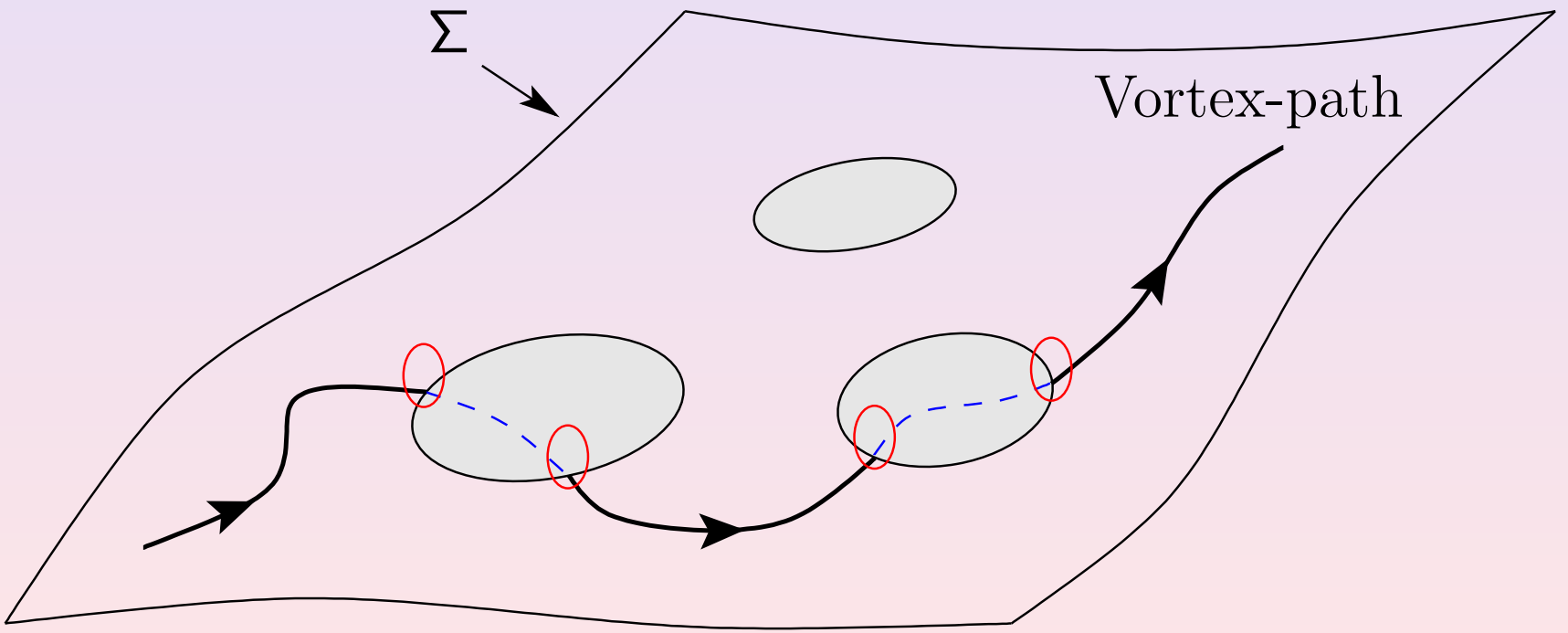
- shortest paths go everywhere
- Σ can be non-orientable
- Jordan curve Theorem does not work (vortices!)

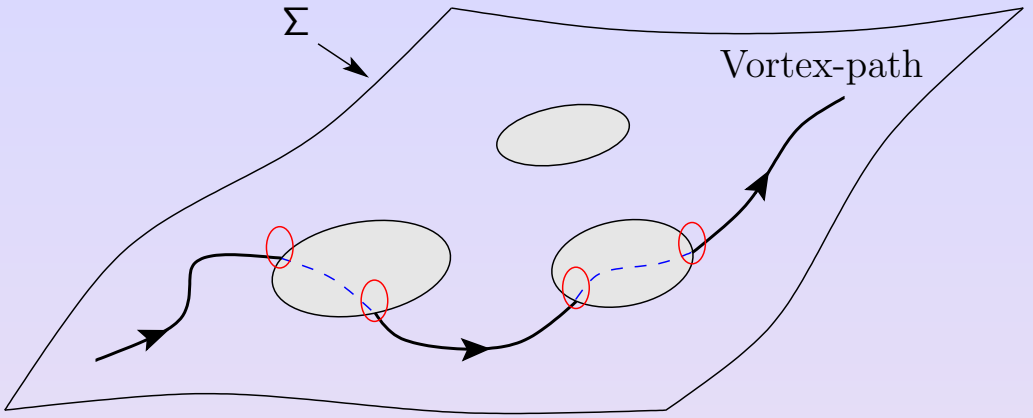


Finding a k-path separator with this?

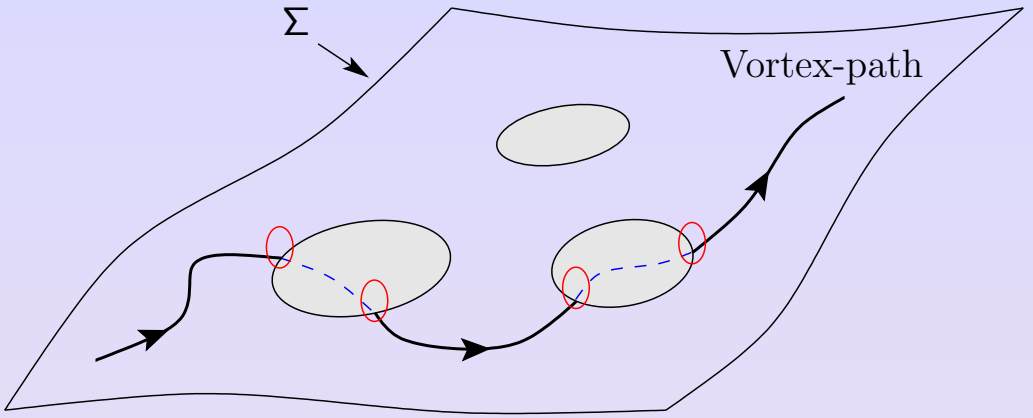
Some remarks:

- shortest paths go everywhere
- Σ can be non-orientable
- Jordan curve Theorem does not work (vortices!)





Note: a vortex-path can be covered by a constant number of shortest paths if segments are shortest paths



Note: a vortex-path can be covered by a constant number of shortest paths if segments are shortest paths

Lemma

If the center subgraph is "nearly-planar" (= no apices and $\Sigma = \mathbb{R}^2$), there are three vortex-paths whose segments are shortest paths, and whose deletions leave components of size $\leqslant n/2$.

Q.E.D.