Distance Labeling

Understanding the Source of Hardness

Adrian Kosowski

Inria Paris

adrian.kosowski@inria.fr

Keywords

I will talk about computing graph distances, using:

- Distance oracles
- Distance labeling
- Hub labeling

to obtain: exact distance values (or additive approximation of distance)

on: general graphs, sparse graphs, and planar instances.

Keywords

I will talk about computing graph distances, using:

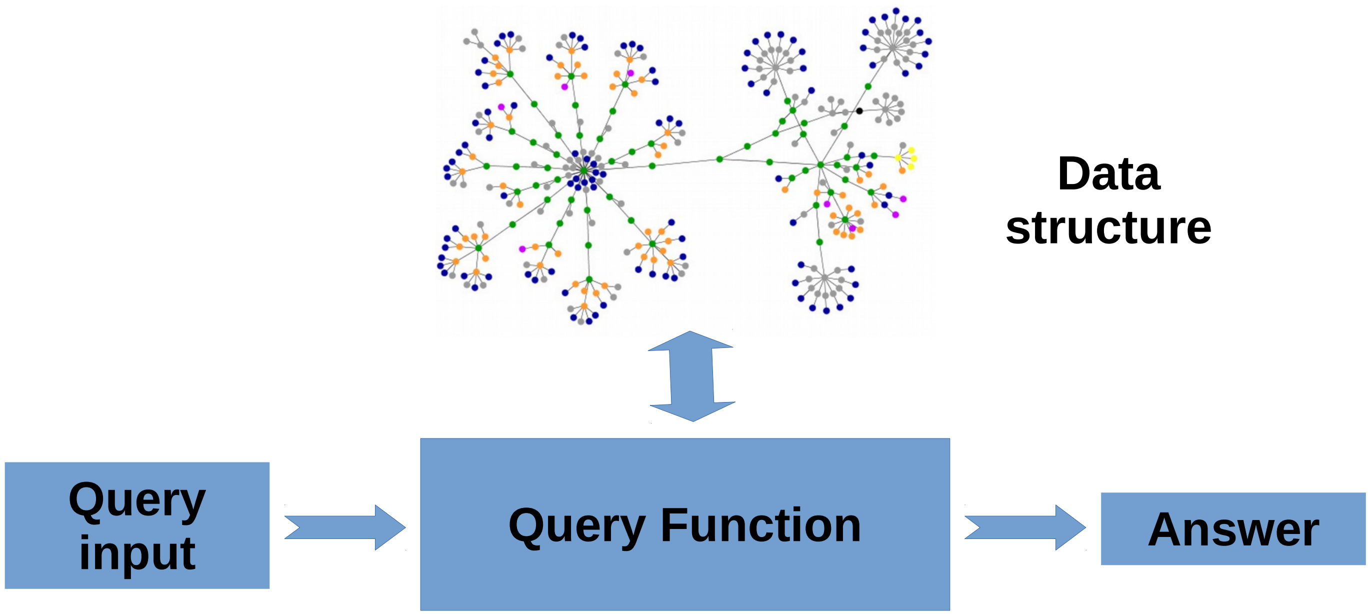
- Distance oracles
- Distance labeling
- Hub labeling

to obtain: exact distance values (or additive approximation of distance)

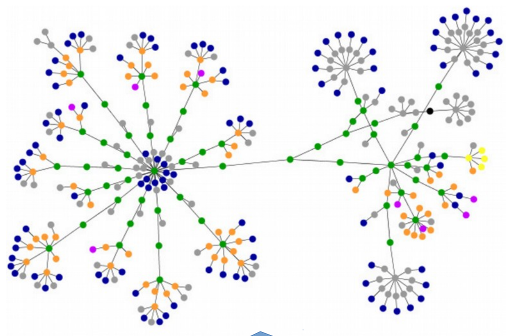
on: general graphs, sparse graphs, and planar instances.

What I will NOT talk about:
compact routing
adjacency oracles
graph embedding
additive spanners
(multiplicative) approximation distance oracles
distance preservers
sketching
oracles for dynamic graphs

Querying for information (static, centralized setting)



Distance Oracle (static, centralized setting)



Data structure

Graph G

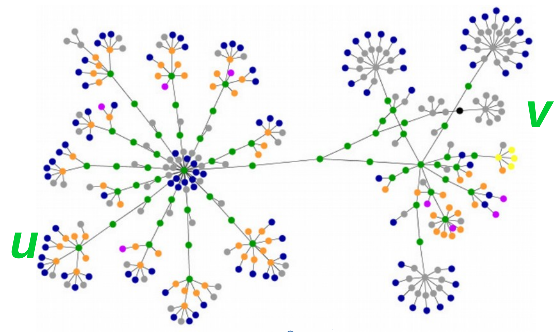
Query input

Query Function

Answer

***GetDistance* (* , *)**

Distance Oracle (static, centralized setting)



Data structure

Graph G

Query input

(u, v)

Query Function

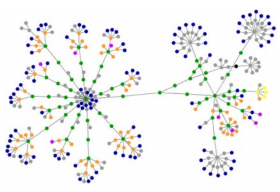
GetDistance (* , *)

Answer

"16"

Designing a distance oracle

Encode once



Data structure
G ∈ $\mathcal{G}$

Encoder

Compact data structure
***G*'** ∈ $\mathcal{G}$'

Use many times

Compact data structure *G*'

Query input
(*u*, *v*)

Decoder

Answer
$dist_G$ (*u*, *v*)

Designing a distance oracle (formally)

Given:

- a promise on graph class $\mathcal{G}$

Design:

- an encoder function *encode* : $\mathcal{G} \rightarrow \mathcal{G}' \subseteq \{0,1\}^*$
- a decoder function *decode* : $\mathcal{G}' \times V^2 \rightarrow \mathbf{N}_+$

Such that:

***decode* (*encode*(G), (*u*,*v*)) = dist$_G$ (u,v)**, for all $G \in \mathcal{G}$, for all $(u,v) \in V^2$

dist$_G$ represents distance in the usual graph metric;
we consider both edge-weighted and unweighted graphs

Objectives in distance oracle design

Desirable features:

- Compactness – small (bit-)size of the encoded data structure
- Fast implementation for decoder (e.g., constant-time)
- Fast implementation for encoder (e.g., linear-time)
- **Labeling scheme:** distributed representation of the distance oracle; handling distributed queries

+ Variants: approximate answers,
handling dynamic graphs,
handling other graph metrics, …

State-of-the-art: unweighted graphs

Distance oracles

0.5 n bits/node
$O(n^2)$ query time

(adjacency matrix)

$O(n \log n)$ bits/node
$O(1)$ query time

(store all distances)

State-of-the-art: unweighted graphs

Distance oracles

0.5 n bits/node
$O(n^2)$ query time

(adjacency matrix)

O(n log n) bits/node
$O(1)$ query time

(store all distances)

($0.5 \log_2 3$) n bits/node
$O(1)$ query time

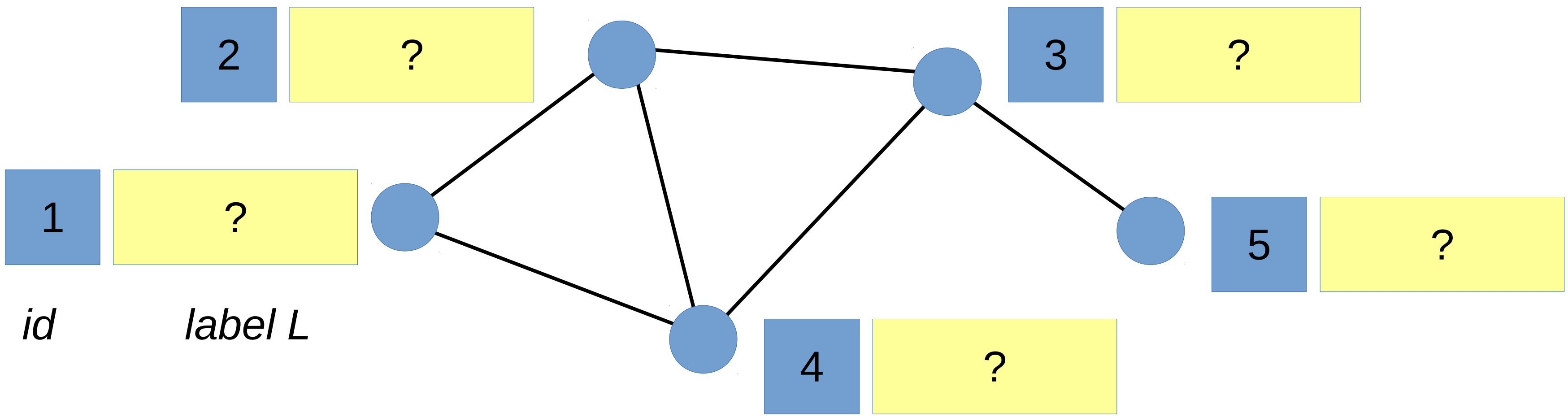
[Nitto and Venturini, CPM 2008]

Distance labeling scheme

A distance oracle distributed over the nodes of the graph.

Initial structure: a n-node graph $G = (V,E,id)$, $id : V \longrightarrow \{1, 2, \ldots, n\}$

Encoder computes vertex labels $L(v) \in \{0,1\}^*$ for $v \in V$

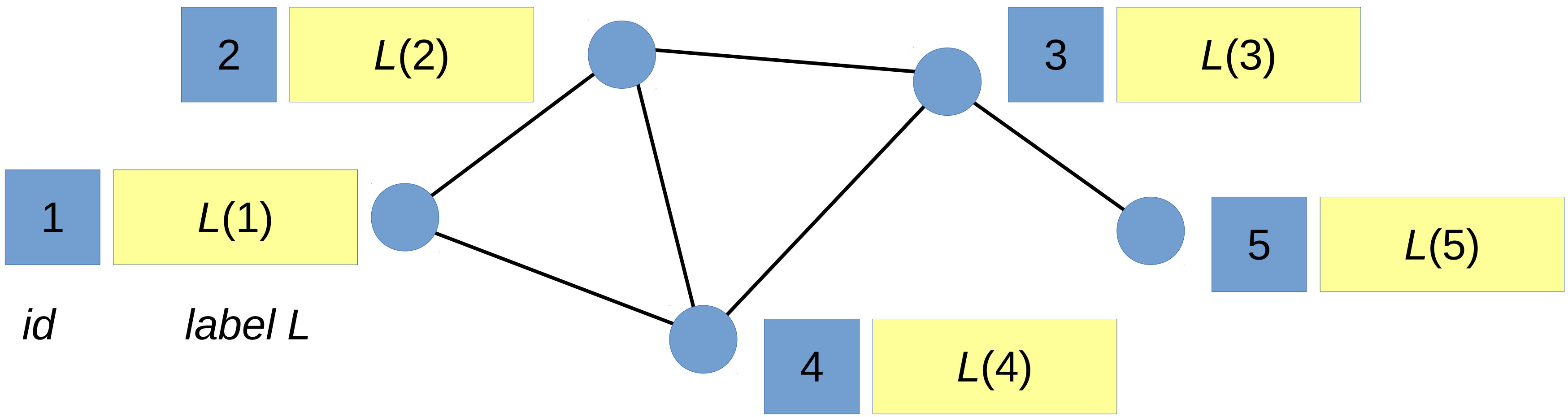


Distance labeling scheme

A distance oracle distributed over the nodes of the graph.

Initial structure: a *n*-node graph $G = (V,E,id)$, $id : V \longrightarrow \{1, 2, \ldots, n\}$

Encoder computes vertex labels $L(v) \in \{0,1\}^*$ for $v \in V$

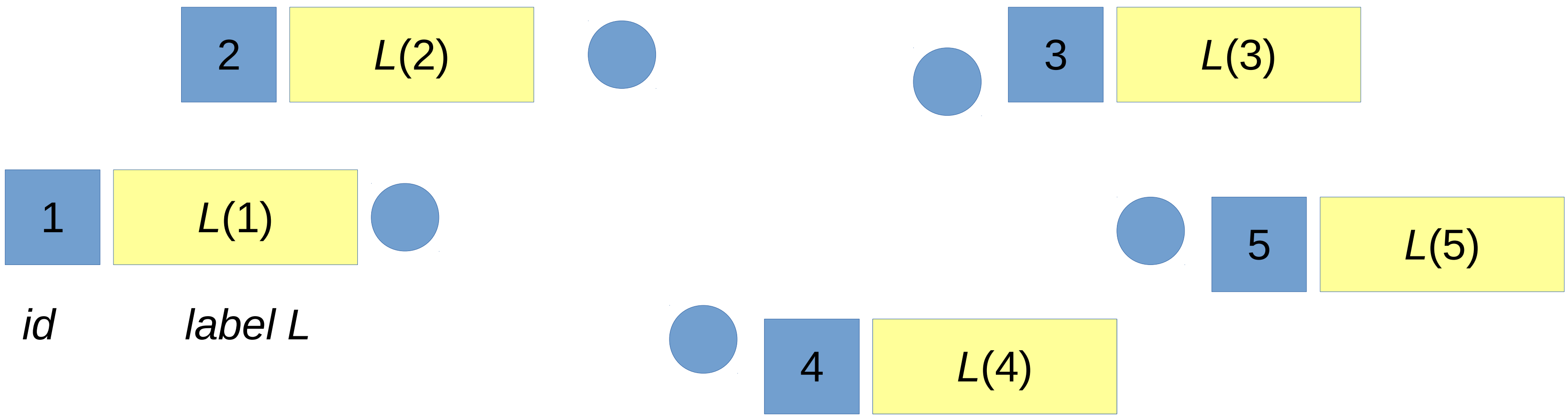


Labeling schemes

The decoder must be able to process a distance query based only on the labels of the involved nodes.

Query: $(L(u), L(v))$

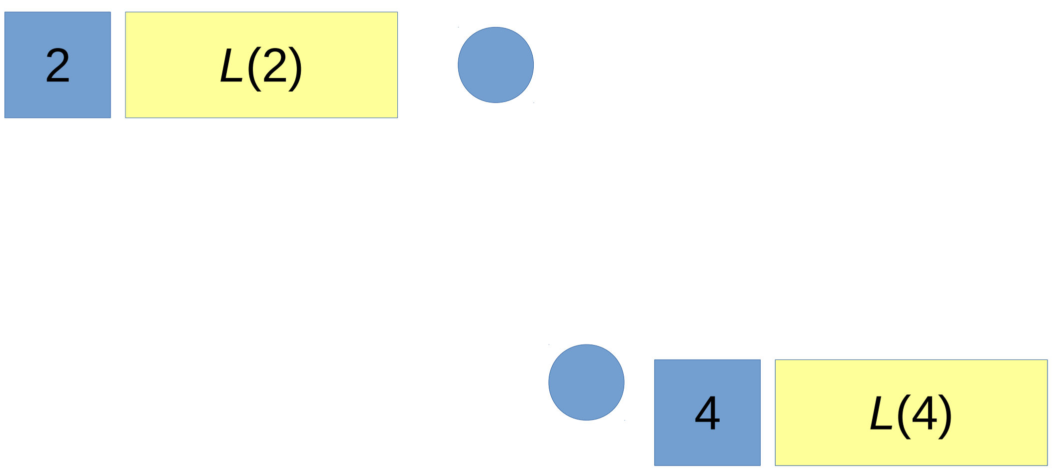
Decoder: $decode\,(L(u), L(v)) = \mathrm{dist}_G\,(\mathrm{L}(u), \mathrm{L}(v))$



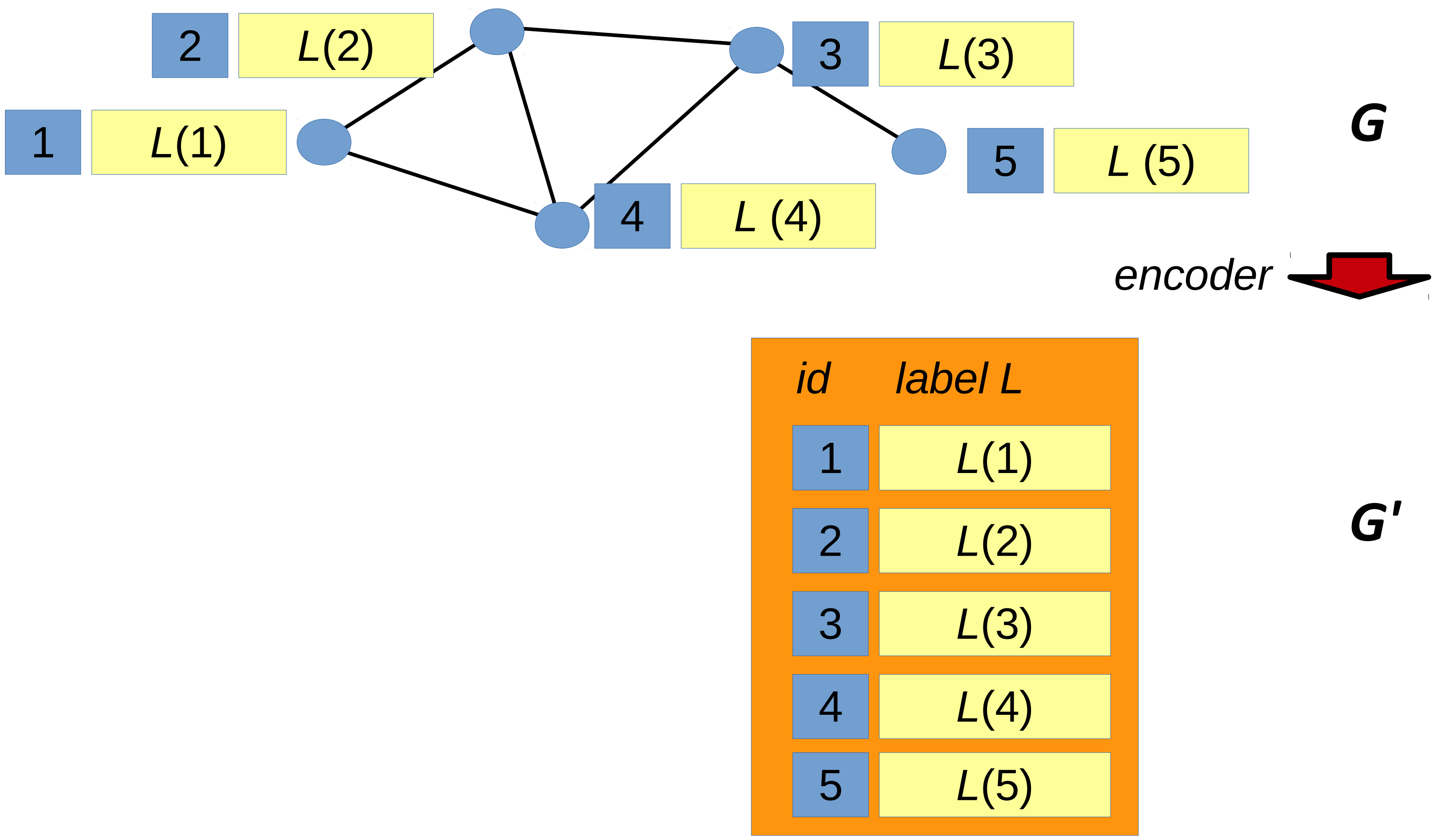
Labeling schemes

The decoder must be able to process a distance query based only on the labels of the involved nodes.

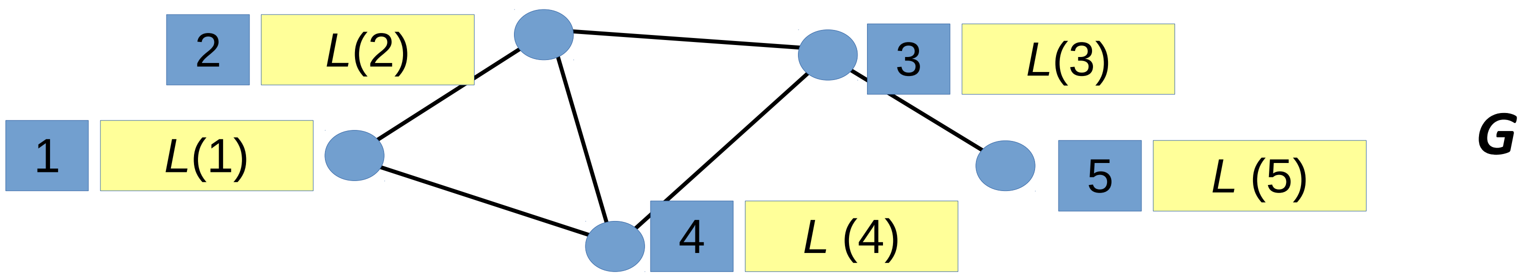
Example: $\mathit{decode}\,(L(2), L(4)) = \mathrm{dist}_G\,(\mathrm{L}(2), \mathrm{L}(4))$



Centralized view of a labeling scheme



Centralized view of a labeling scheme



Query: ”distance between nodes 2 and 4?”

input: (2,4)

id	*label L*
1	*L*(1)
2	*L*(2)
3	*L*(3)
4	*L*(4)
5	*L*(5)

Algebraic distance labeling

General idea

- Node labels are vectors in a space with some dot product : $L(v_x) = (x_1, x_2, \ldots, x_s)$, for $v_x \in V$.
- Apply the following decoding:

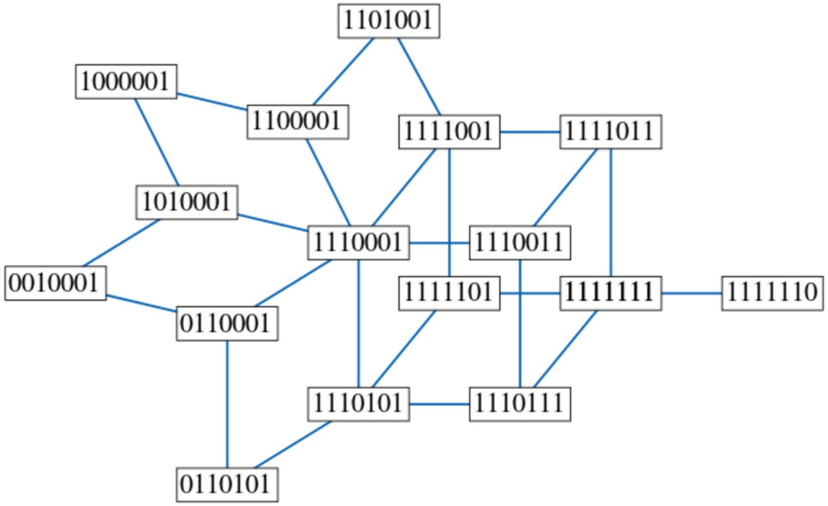
$$\mathrm{dist}(v_x,v_y) = L(v_x) \bullet L(v_y) = \bigoplus^{s}_{i=1} x_i \bullet y_i.$$

- Label size for v follows from the number of non-zero entries of vector $L(v)$.

First approach: cube embedding

Distance-preserving embedding of the graph in a cube

- Use bit vectors of a given length to represent node labels *L*
- Choose labels so that: Hamming-Distance (L(u),L(v)) = dist_G (u,v).

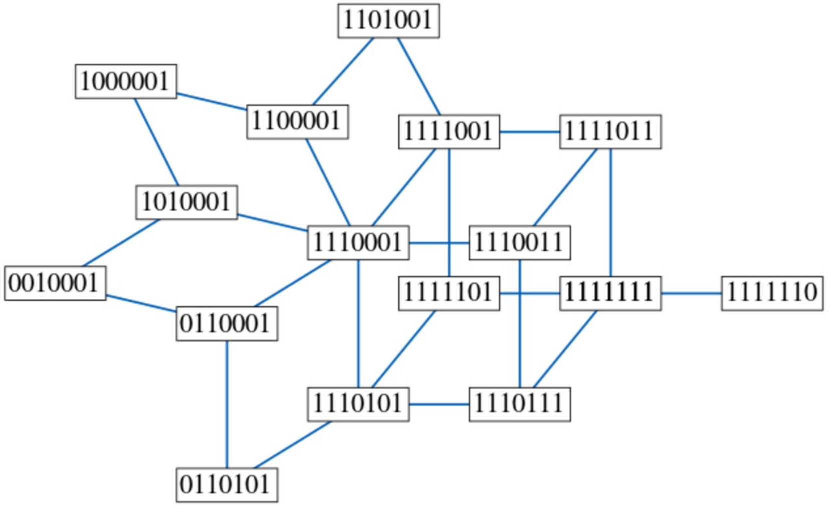


First approach: cube embedding – failed!

Distance-preserving embedding of the graph in a cube

- Use bit vectors of a given length to represent node labels *L*
- Choose labels so that: Hamming-Distance (L(u),L(v)) = dist_G (u,v).
- … ***only possible for some graphs***, regardless of allowed label length.

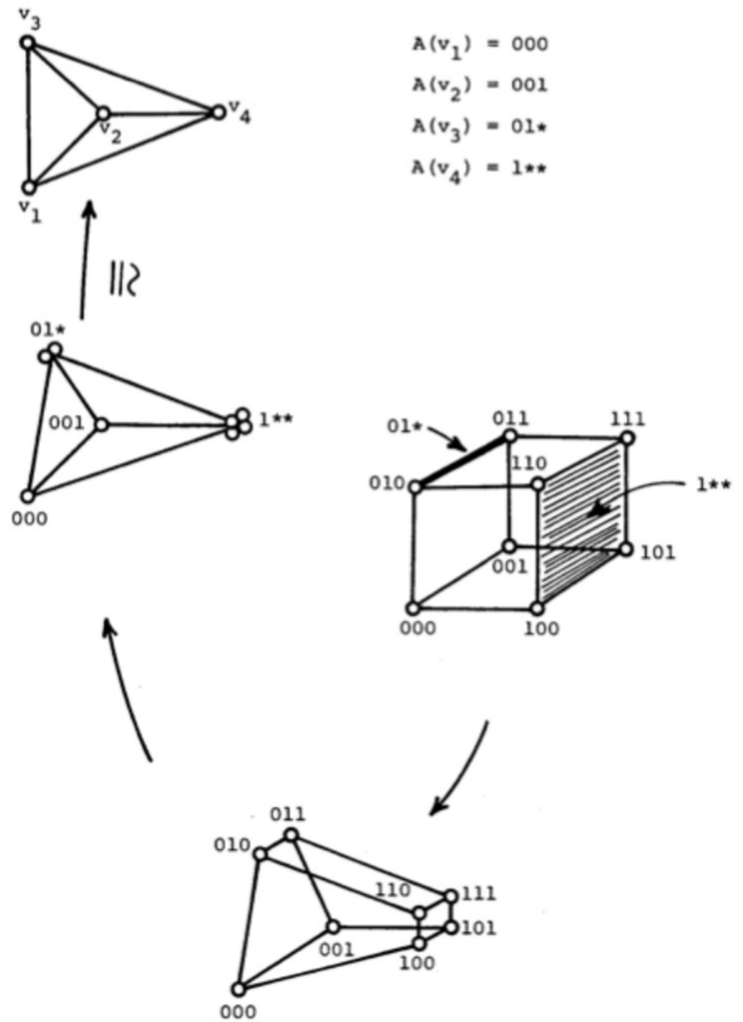
  [Firsov 1965, Djoković 1973, Winkler 1984]



First approach, revisited: squashed cube dimension

Mapping graph nodes to *hyperplanes* in a cube
[Graham, Pollack 1971]

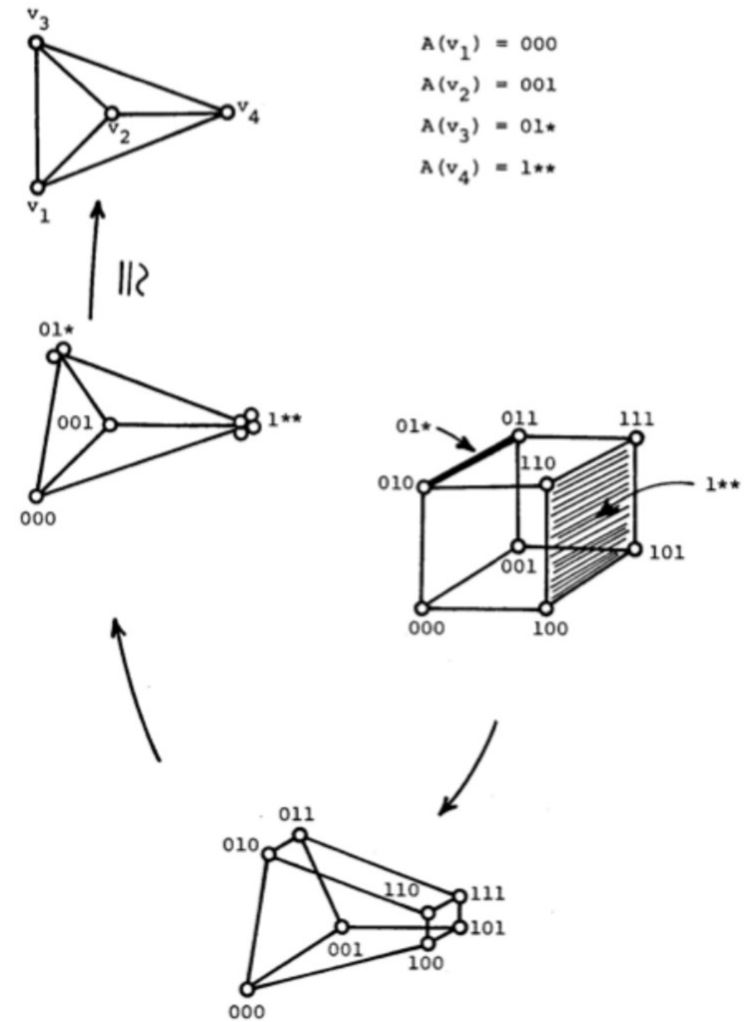
- Use vectors in $\{0,1,*\}^l$ to represent node labels L, for some dimension l
- Here, "*" denotes a wildcard symbol, whose Hamming distance to any other symbol is 0.
- Choose labels so that:
  Hamming-Distance $(L(u),L(v)) = dist_G\,(u,v)$.



First approach, revisited: squashed cube dimension

Mapping graph nodes to *hyperplanes* in a cube
[Graham, Pollack 1971]

- Use vectors in $\{0,1,*\}^l$ to represent node labels L, for some dimension l
- Here, "*" denotes a wildcard symbol, whose Hamming distance to any other symbol is 0.
- Choose labels so that:
  Hamming-Distance $(L(u),L(v)) = dist_G(u,v)$.



Theorem **[Winkler, 1983].** Every connected n-node graph admits such a mapping with $l \leq n$-1.

The above bound is tight for the complete graph.

Provided distance labels of $\log_2 3\ n \approx$ **1.58*n*** bits.

In the RAM model, decoding time for a query is **almost linear in *n*.** (Encoding time is polynomial.)

State-of-the-art: unweighted graphs

Distance oracles

$0.5\ n$ bits/node
$O(n^2)$ query time

(adjacency matrix)

$O(n \log n)$ bits/node
$O(1)$ query time

(store all distances)

$(0.5 \log_2 3)\ n$ bits/node
$O(1)$ query time

[Nitto and Venturini, CPM 2008]

Distance labeling

$O(n)$ bits/node
$O(n)$ query time

[Winkler 1984]

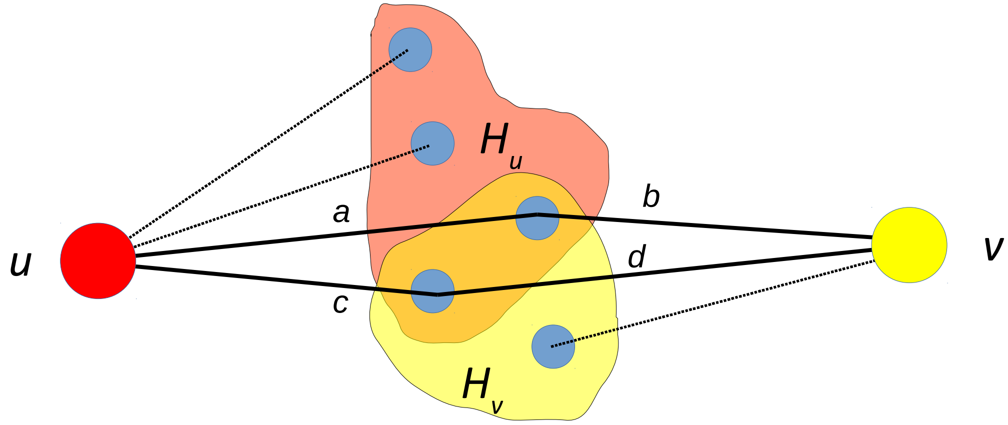
Second approach: hub labeling

Hub labeling scheme (a.k.a.: landmark labeling, 2-hop-cover)

- Each node v is assigned a *hub set* $H_v \subseteq V$
- $L(v) = [\ D_v(u) : u \in V\ ]$, where: $D_v(u) = \text{dist}\,(v, u)$, for $u \in H(v)$,
  $= +\infty$ (omitted), otherwise.

Decoder:

$$\text{dist}\,(u, v) = \min_{w \in V}\ (\ D_u\,(w) + D_v\,(w)\ )$$



Hub labeling

Distance decoding algorithm in practice

H_A	↓ 2	5	6	8	11	22
D_A	42	12	13	70	8	19

H_B	↓ 3	4	8	15	18	22	31
D_B	50	47	31	7	3	80	1002

Dist(A,B) = ∞

Hub labeling

Distance decoding algorithm in practice

H_A	2	5	6	↓ 8	11	22
D_A	42	12	13	70	8	19

H_B	3	4	↓ 8	15	18	22	31
D_B	50	47	31	7	3	80	1002

Dist(A,B) = 70 + 31 = 101

Hub labeling

Distance decoding algorithm in practice

H_A	2	5	6	8	11	22
D_A	42	12	13	70	8	19

H_B	3	4	8	15	18	22	31
D_B	50	47	31	7	3	80	1002

Dist(A,B) = 19 + 80 = 99

Hub label size

Size of L(v) for unweighted graphs:

- O (h log n) bits trivially, where $h = |H_v|$.
- **O (h log (n/h)) bits**: [folklore; overview in Gawrychowski, K., Uznanski, DISC 2016]
  - Trick: $L(v) = [\ D_v(u) : u \in V\ ]$, nodes u enumerated in a specific order u = 1..n (e.g., preorder traversal of a fixed spanning tree).
  - Use an optimal-entropy encoding of $D_v(u+1)-D_v(u)$ in the label.

Example:

Put $H_v = V$ and $h = n$. The latter bound gives labels of size $O(n)$.

State-of-the-art: unweighted graphs

Distance oracles

0.5 n bits/node
$O(n^2)$ query time

(adjacency matrix)

O(n log n) bits/node
$O(1)$ query time

(store all distances)

(0.5 $\log_2 3$) n bits/node
$O(1)$ query time

[Nitto and Venturini, CPM 2008]

Distance labeling

$O(n)$ bits/node
$O(n)$ query time

[Winkler 1984]

Hub labeling

$O(n)$ bits/node
$O(n)$ query time

(incremental entropy encoding of all distances)

State-of-the-art: unweighted graphs

Distance oracles

0.5 n bits/node
$O(n^2)$ query time

(adjacency matrix)

O(n log n) bits/node
$O(1)$ query time

(store all distances)

($0.5 \log_2 3$) n bits/node
$O(1)$ query time

[Nitto and Venturini, CPM 2008]

Distance labeling

$O(n)$ bits/node
$O(n)$ query time

[Winkler 1984]

11 n bits/node
$O(\log \log n)$ query time
[Gavoille, Peleg, Perennez, Raz SODA 2001]

...

Hub labeling

$O(n)$ bits/node
$O(n)$ query time

(incremental entropy encoding of all distances)

State-of-the-art: unweighted graphs

Distance oracles

0.5 n bits/node
$O(n^2)$ query time

(adjacency matrix)

$O(n \log n)$ bits/node
$O(1)$ query time

(store all distances)

$(0.5 \log_2 3)\, n$ bits/node
$O(1)$ query time

[Nitto and Venturini, CPM 2008]

Distance labeling

$O(n)$ bits/node
$O(n)$ query time

[Winkler 1984]

11 n bits/node
$O(\log \log n)$ query time
[Gavoille, Peleg, Perennez, Raz SODA 2001]

...

$(0.5 \log_2 3)\, n$ bits/node
$O(1)$ query time

[Alstrup, Gavoille, Halvorsen, and Petersen, SODA 2016]

Hub labeling

$O(n)$ bits/node
$O(n)$ query time

(incremental entropy encoding of all distances)

State-of-the-art: unweighted graphs

Distance oracles

0.5 n bits/node
$O(n^2)$ query time

(adjacency matrix)

$O(n \log n)$ bits/node
$O(1)$ query time

(store all distances)

$(0.5 \log_2 3)$ n bits/node
$O(1)$ query time

[Nitto and Venturini, CPM 2008]

Distance labeling

$O(n)$ bits/node
$O(n)$ query time

[Winkler 1984]

11 n bits/node
$O(\log \log n)$ query time
[Gavoille, Peleg, Perennez, Raz SODA 2001]

…

$(0.5 \log_2 3)$ n bits/node
$O(1)$ query time

[Alstrup, Gavoille, Halvorsen, and Petersen, SODA 2016]

Hub labeling

$O(n)$ bits/node
$O(n)$ query time

(incremental entropy encoding of all distances)

But this is **not** the final word of Hub labeling!

Hub labeling in **sparse graphs**

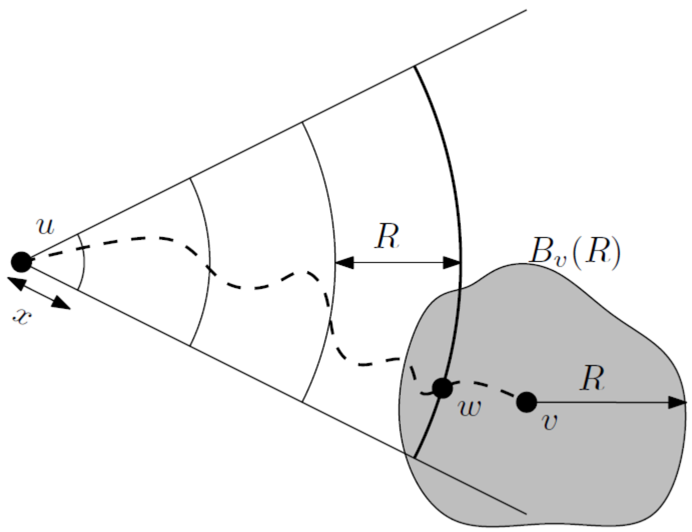
- Simplifying assumption: max degree = constant (i.e., 3)
- Define hub set H_v:

[Gawrychowski, K., Uznański, DISC 2016]

Ball of radius $R = \lfloor \varepsilon \log_2 n \rfloor$ around v

$\cup$

All nodes at distance x, $x + R$, $x + 2R$, $x + 3R$,... from v
($x < R$ is appropriately chosen)



$\mathcal{O}(\frac{n}{\log n} \log \log n)$ bits

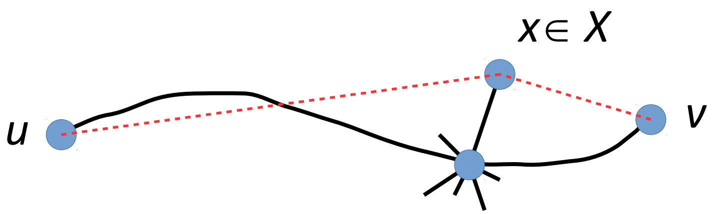
$O(n^{\varepsilon})$ decoding time

Back to the general case

Hub Labeling + (general idea)

- Hub labeling method for sparse graphs works also in dense graphs
  - Condition for small hub set size for *v*: ball around *v* must have small average degree

- Fix: handle high-degree nodes separately
  - Let $X \subseteq V$ be a dominating set for nodes of *V* with large degree (> log *n*)
  - Choose *X* with $|X| = o(n)$.
  - Add *X* to hub sets of all nodes.



- Caveat: not an exact distance scheme but 2-additive.

- Label size: $o(n)$, decoding time: $\omega(1)$

Unweighted graphs: cutoff in additive approximation

Any exact distance oracle requires $\geq n/2 - O(1)$ bits per node

- Decode the adjacency matrix of *G* from its (exact) distance oracle

Any 1-additive distance oracle requires $\geq n/4 - O(1)$ bits per node

- Decode the adjacency matrix of *G* from its 1-additive distance oracle if G is bipartite

A 2-additive hub labeling uses only $o(n)$ bits per node.

Is this some kind of universal issue?

- Similar story possible for time complexity of additive approximation of APSP [Dor, Halperin, Zwick, SICOMP 2000]

- Fixing 2-additive labeling $\rightarrow$ exact labeling: $0.5 \log_2 3\, n$ extra bits per node

State-of-the-art: unweighted graphs

Distance oracles

0.5 n bits/node
$O(n^2)$ query time

(adjacency matrix)

$O(n \log n)$ bits/node
$O(1)$ query time

(store all distances)

$(0.5 \log_2 3)\ n$ bits/node
$O(1)$ query time

[Nitto and Venturini, CPM 2008]

Distance labeling

$O(n)$ bits/node
$O(n)$ query time

[Winkler 1984]

11 n bits/node
$O(\log \log n)$ query time
[Gavoille, Peleg, Perennez, Raz SODA 2001]

...

$(0.5 \log_2 3)\ n$ bits/node
$O(1)$ query time

[Alstrup, Gavoille, Halvorsen, and Petersen, SODA 2016]

Hub labeling

$O(n)$ bits/node
$O(n)$ query time

(incremental entropy encoding of all distances)

State-of-the-art: unweighted graphs

Distance oracles

0.5 n bits/node
$O(n^2)$ query time

(adjacency matrix)

$O(n \log n)$ bits/node
$O(1)$ query time

(store all distances)

$(0.5 \log_2 3)$ n bits/node
$O(1)$ query time

[Nitto and Venturini, CPM 2008]

Distance labeling

$O(n)$ bits/node
$O(n)$ query time

[Winkler 1984]

11 n bits/node
$O(\log \log n)$ query time
[Gavoille, Peleg, Perennez, Raz SODA 2001]

...

$(0.5 \log_2 3)$ n bits/node
$O(1)$ query time

[Alstrup, Gavoille, Halvorsen, and Petersen, SODA 2016]

Hub labeling

$O(n)$ bits/node
$O(n)$ query time

(incremental entropy encoding of all distances)

Hub labeling +

$(0.5 \log_2 3)$ n bits/node
$\omega(1)$ query time

[Gawrychowski, K., Uznanski, DISC 2016]

State-of-the-art: unweighted graphs

Distance oracles

$0.5\ n$ bits/node
$O(n^2)$ query time

(adjacency matrix)

Distance labeling

Hub labeling +

$(0.5 \log_2 3)\ n$ bits/node
$\omega(1)$ query time

[Gawrychowski, K., Uznanski, DISC 2016]

The right constant: 0.5 or $0.5 \log_2 3$?

- Disclaimers:
  - Possibly neither is right.
  - Possibly different constants for the different regimes.

- We can handle nodes at large distances easily, it is constant distances which pose problems.

- If $0.5 \log_2 3$ is the right answer, then the entropy of the distance matrix must be sufficiently large ($\log_2 3$ bits to encode a single entry)

- Is there a graph in which a uniformly random pair of nodes has equal probability to be at distances 1, 2, and 3? [probably not...]

- Possible to construct a graph with equal probability of node distances 2, 3, and 4 - but there seem to be *few* such graphs.*

* - this does not preclude lower bounds, but makes finding them harder (no counting arguments).

Unweighted **sparse** graphs

Distance oracles

$O(1)$ bits/node
$O(n)$ query time

(adjacency list)

Distance labeling

Hub labeling

$\hat{O}(n / \log n)$ bits/node
$O(1)$ query time

[Alstrup, Dahlgaard, Knudsen, Porat, ESA 2016]
[Gawrychowski, K., Uznański, DISC 2016]

Unweighted sparse graphs

Distance oracles

$O(1)$ bits/node
$O(n)$ query time

(adjacency list)

Distance labeling

Trade-offs?

Hub labeling

$\hat{O}(n / \log n)$ bits/node
$O(1)$ query time

[Alstrup, Dahlgaard, Knudsen, Porat, ESA 2016]
[Gawrychowski, K., Uznański, DISC 2016]

Any smaller distance labelings in sparse graphs?

- Conjectured answer: **no**.
  - Existence of Ruzsa-Szemerédi graphs kills **hub labeling**.

    [= Very dense graphs with almost a linear number of edge-disjoint induced matchings.]

    [More details provided during the talk]

What's going on for planar graphs?

Weighted planar graphs:

- $O(n^{1/2}\log n)$ bits distance labeling, tight up to polylog factors. [Gavoille et al. 2001]

Unweighted planar graphs:

- Upper bound: $\mathbf{O(n^{1/2})}$ bits for hub labeling [Gawrychowski, Uznański: arXiv: 1611.06529]
- Lower bound: $\mathbf{\Omega(n^{1/3})}$ bits distance labeling [Gavoille et al. 2001]

What's going on for planar graphs?

Weighted planar graphs:

- $O(n^{1/2}\log n)$ bits distance labeling, tight up to polylog factors. [Gavoille et al. 2001]

Unweighted planar graphs:

- Upper bound: $\mathbf{O(n^{1/2})}$ bits for hub labeling [Gawrychowski, Uznański: arXiv: 1611.06529]
- Lower bound: $\mathbf{\Omega(n^{1/3})}$ bits distance labeling [Gavoille et al. 2001]

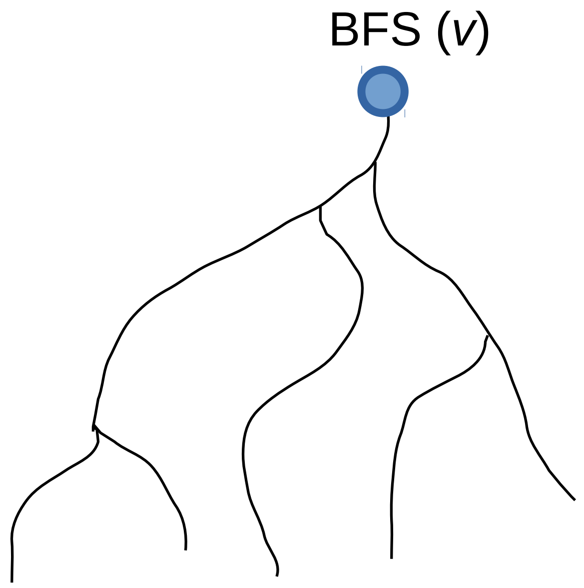
- Evidence that the lower bound technique cannot be improved further without significantly new ideas [Abboud, Gawrychowski, Mozes, Weimann, SODA 2018]
- Proof that distance labelings are **not** the best distance oracle possible when we are only interested in distances between some subset of the nodes. [Abboud, Gawrychowski, Mozes, Weimann, SODA 2018]
- Non-trivial fast distance oracles: $\tilde{O}(1)$ decoding time with $\tilde{O}(n^{2/3})$ space per node [Cabello, SODA 2017]
  - No comparable fast distance labelings are known.

Distance Labelings ≈ Hub Labelings?

- Hub labeling techniques are practical and in practice can be implemented in a parallelizable way.
- There seem to be no (non-artificial) graph classes where a distance labeling technique visibly outperforms hub labeling... [Open problem: change this state of affairs!]
- Polynomial-time algorithms to O(log *n*)-approximate average/maximum hub set size [Cohen et al. 2003; Goldberg et al. ICALP 2013]
- Polynomial-time algorithm to O(log diam)-approximate average hub set size for (weighted) graphs with unique shortest paths [Angelidakis, Makarychev, Oparin, SODA 2017]
- For planar graphs, hub labelings are not the best distance oracle known.
  - But: for practical planar instances (road/infrastructure networks), they seem to be among the best.
  - Attempts at theoretical explanation: [Abraham et al. SODA 2011, K. & Viennot SODA 2017]
  - Q: What's the situation for percolation graphs?

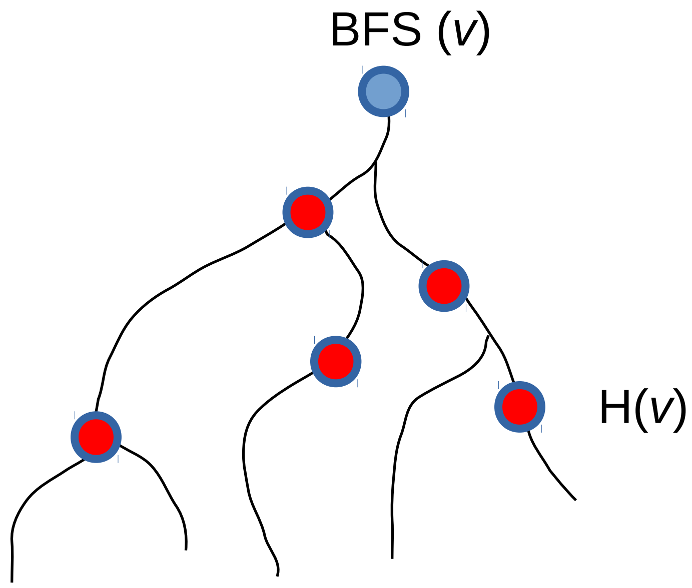
An alternative view for a hub labeling

- Assumption: don't care about log-factors in analysis; unique shortest path graph.
- Equivalence between a hub set and the BFS subtree it induces



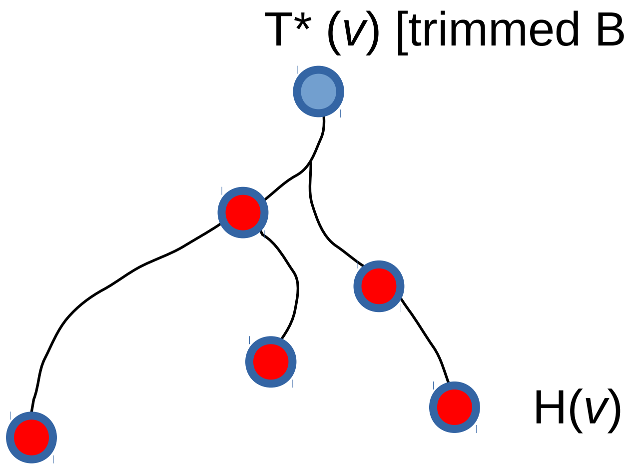
An alternative view for a hub labeling

- Assumption: don't care about log-factors in analysis; unique shortest path graph.
- Equivalence between a hub set and the BFS subtree it induces



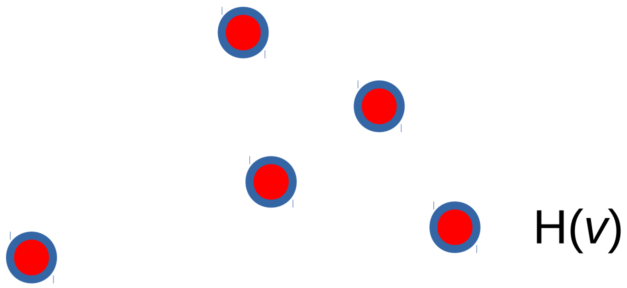
An alternative view for a hub labeling

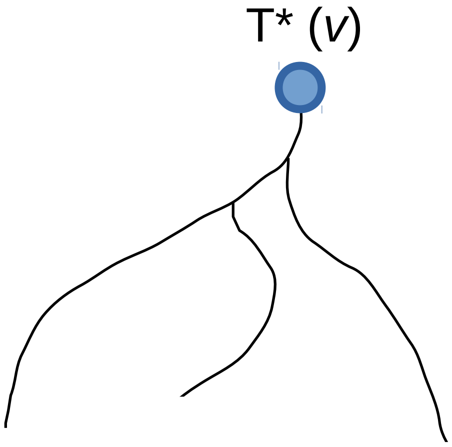
- Assumption: don't care about log-factors in analysis; unique shortest path graph.
- Equivalence between a hub set and the BFS subtree it induces



An alternative view for a hub labeling

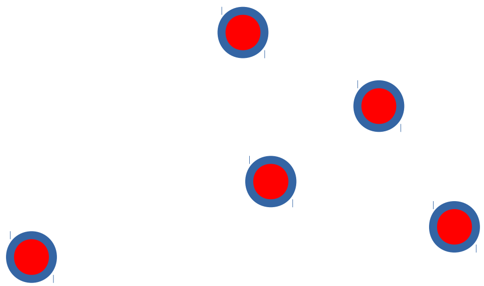
- Assumption: don't care about log-factors in analysis; unique shortest path graph.
- Equivalence between a hub set and the BFS subtree it induces





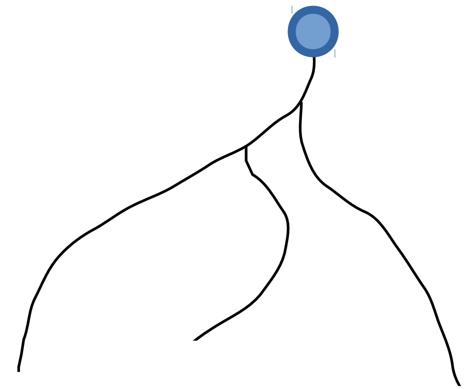
An alternative view for a hub labeling

- Assumption: don't care about log-factors in analysis; unique shortest path graph.
- Equivalence between a hub set and the BFS subtree it induces



H(v)

T* (v)

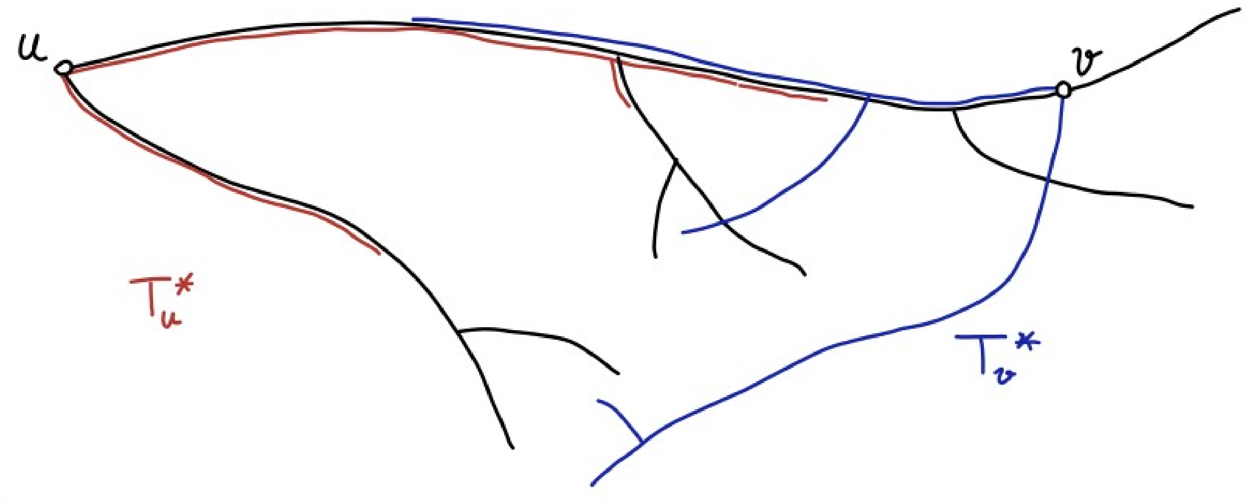


|H(v)| ≈ number of leaves of T* (v)

[Angelidakis, Makarychev, Oparin, SODA 2017] [K.& Viennot, SODA 2017]

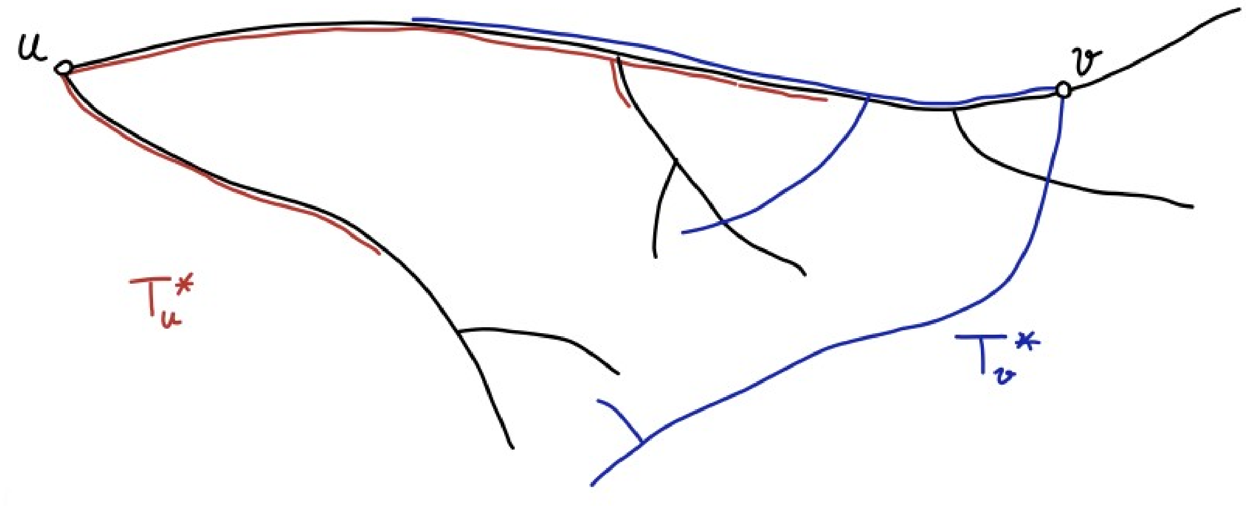
An alternative view for a hub labeling

- Idea: construct trees T* instead of hub sets
- **Condition:** Shortest u-v path is covered by union of T*(u) and T*(v)



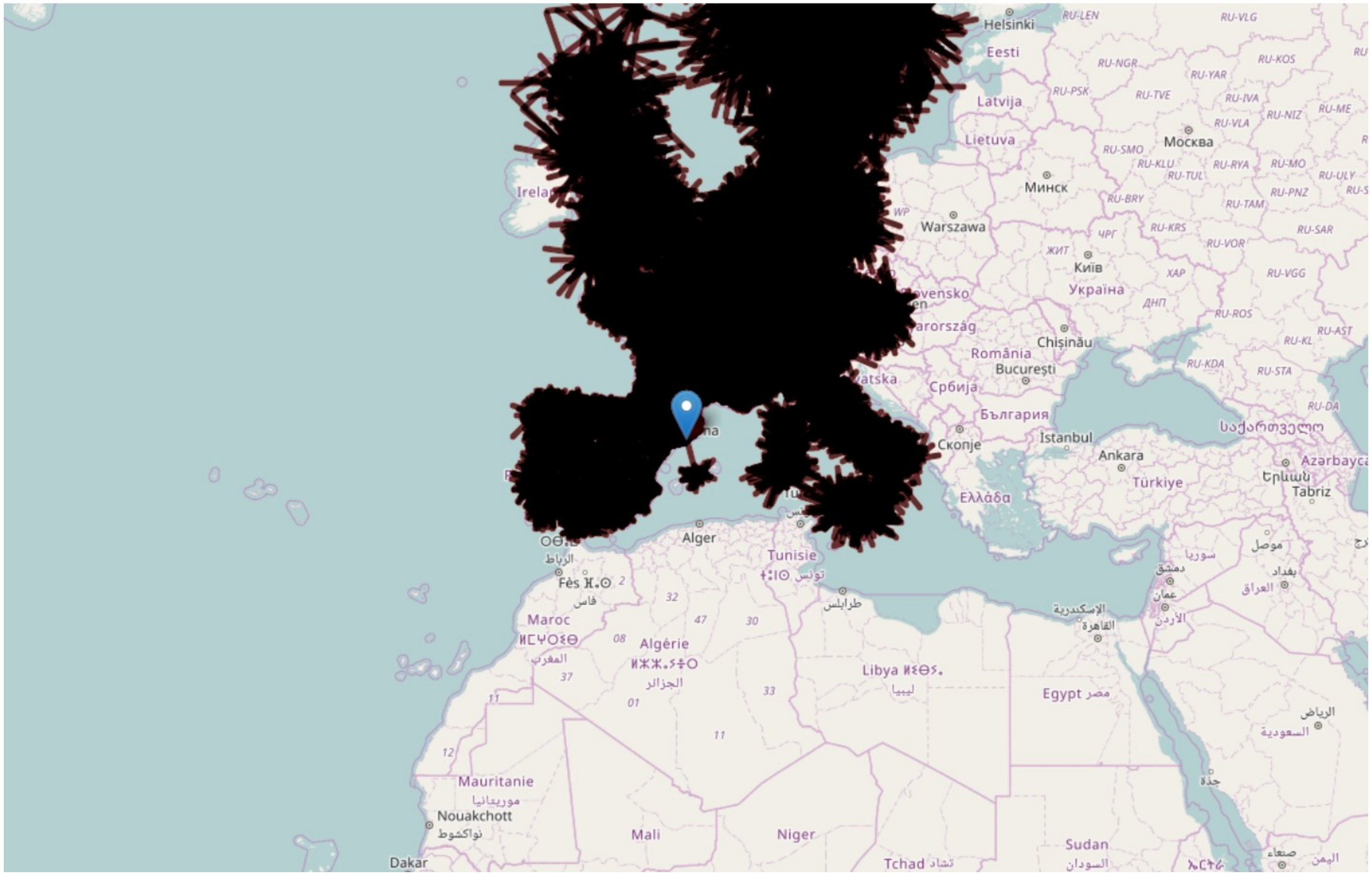
An alternative view for a hub labeling

- Idea: construct trees T* instead of hub sets
- **Condition:** Shortest u-v path is covered by union of T*(u) and T*(v)

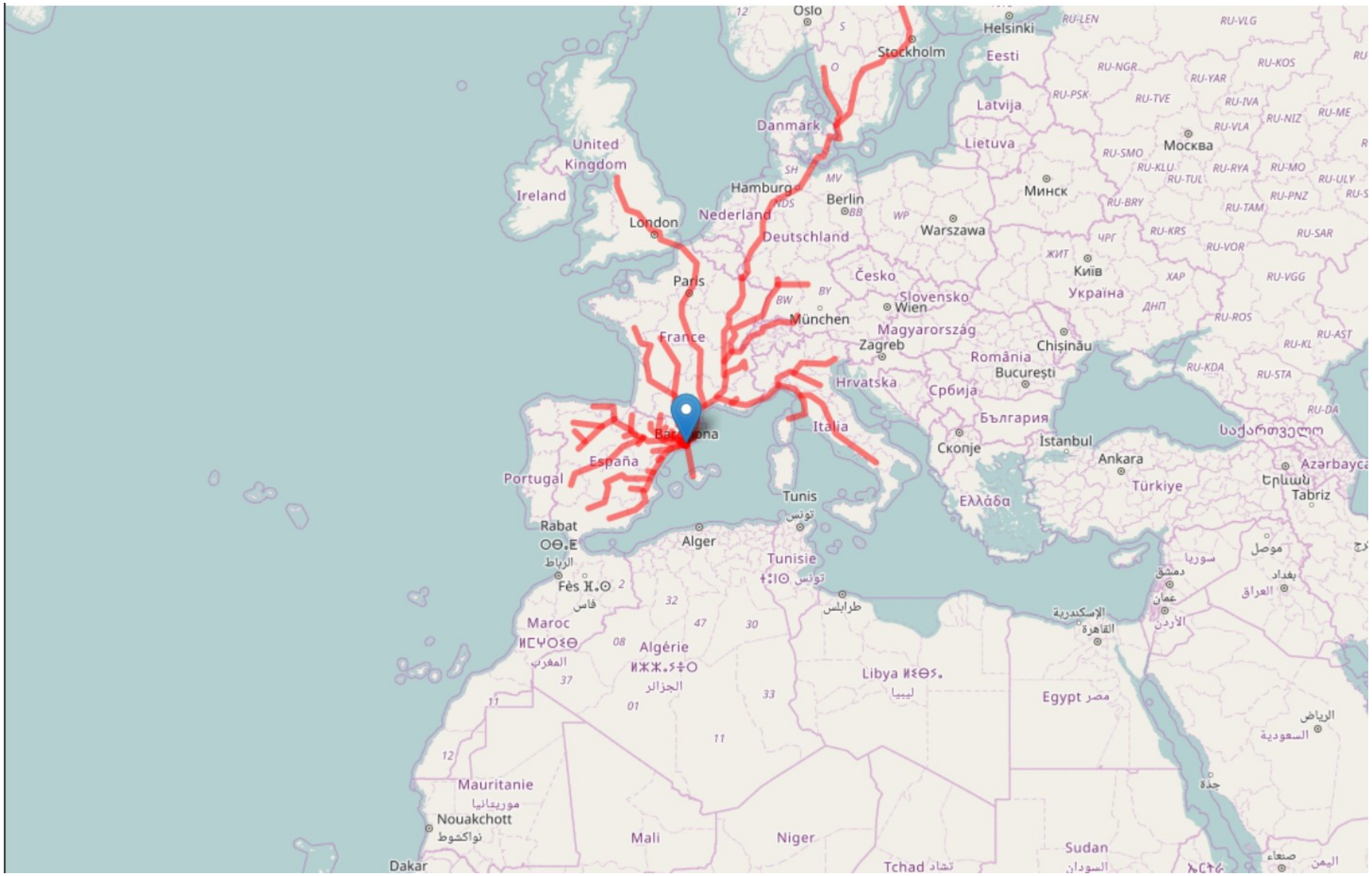


- Obtaining a small hub labeling ⇔ Choosing the right place to cut each tree BFS(v) to T*(v).
  - Polynomial time constant-factor approximation [Angelidakis, Makarychev, Oparin, SODA 2017]
  - Cutting tree branches in the middle works "in practice" [K.& Viennot, SODA 2017]

Shortest path tree for Barcelona



T*(Barcelona) after prunning ends of branches



Thank you!