Hub Labeling Algorithms

Andrew V. Goldberg

Amazon.com

Algorithms at Amazon

- Work on hub labeling was done while I was at Microsoft Research
- Amazon has may interesting algorithmic problems with OR and CS flavor
- Amazon is hiring PhDs as scientists and interns

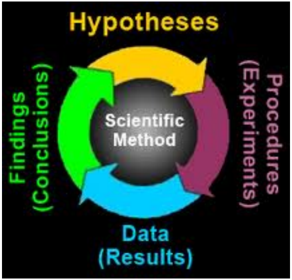
Collaborators

The work on hub labeling took place over many years

Joint work with

Ittai Abraham, Maxim Babenko, Daniel Delling, Haim Kaplan, Thomas Pajor, Ruslan Savchenko, Mathis Weller, Renato Werneck

Theory vs. Practice

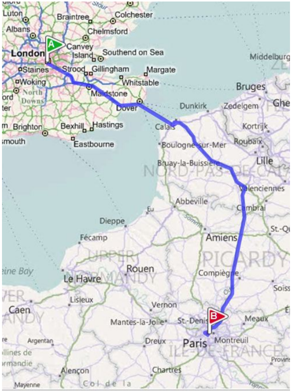


Outline

Motivation

Shortest path applications

- driving directions in road networks
- indoor and terrain navigation
- routing in communication/sensor networks
- moving agents on game maps
- proximity in social/collaboration networks



Challenges

- massive networks of varying structure
- real-time queries

Need a fast and robust approach

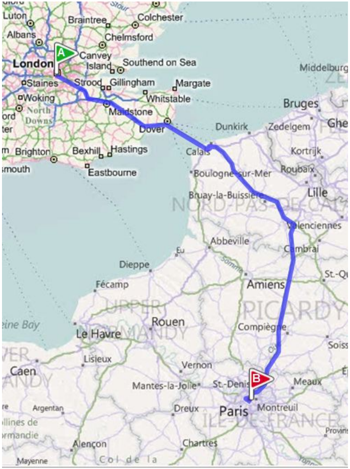
Single Pair Shortest Paths Problem

Input

- Graph $G = (V, E)$; $|V| = n$, $|E| = m$
- Length function ℓ
- Assume G is undirected (simpler notation)
- HL algorithm works for directed graphs

Query (multiple for the same network)

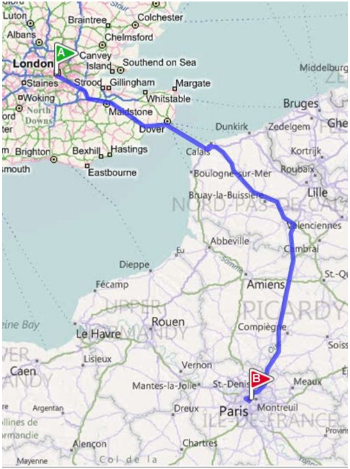
- Given origin s and destination t, find an optimal path from s to t



Single Pair Shortest Paths Problem

Input

- Graph $G = (V, E)$; $|V| = n$, $|E| = m$
- Length function ℓ
- Assume G is undirected (simpler notation)
- HL algorithm works for directed graphs



Query (multiple for the same network)

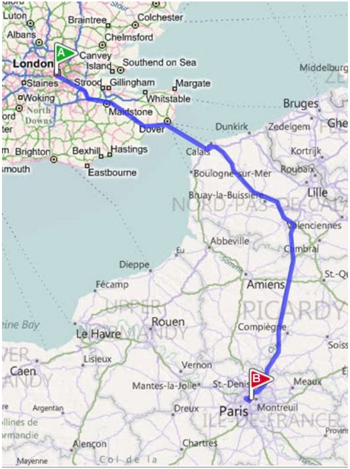
- Given origin s and destination t, find an optimal path from s to t

$n \times n$ distance table infeasible for large graphs

SP Algorithms with Preprocessing

Motivating application: driving directions

- preprocessing to speed up queries
  - may take much longer than a query
  - can use a more powerful machine
- queries are fast (e.g., real-time)

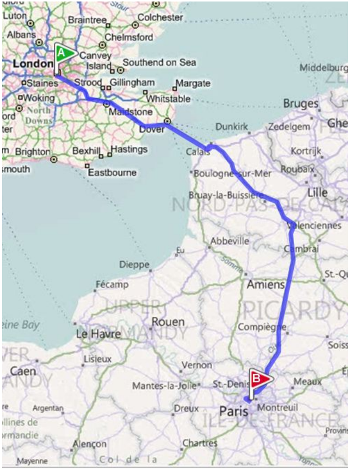


SP Algorithms with Preprocessing

Motivating application: driving directions

- preprocessing to speed up queries
  - may take much longer than a query
  - can use a more powerful machine
- queries are fast (e.g., real-time)

HL works very well for a static distance oracle implementation



Outline

Labeling Algorithms

Labeling Algorithm [P 99]

- precompute labels $L(v)$ for all $v \in V$
- answer s, t query using $L(s)$ and $L(t)$ only
- G used only for preprocessing

Labeling Algorithms

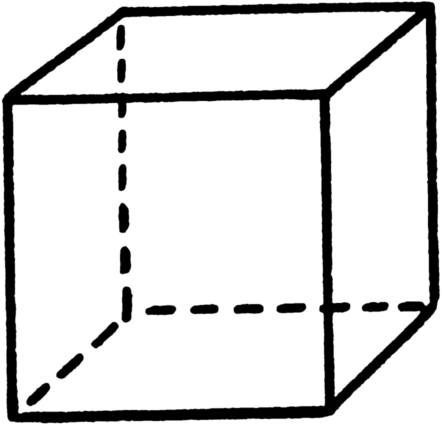
Labeling Algorithm [P 99]

- precompute labels $L(v)$ for all $v \in V$
- answer s, t query using $L(s)$ and $L(t)$ only
- G used only for preprocessing

Label Sizes

- some networks have small labels, some do not [GPPR 04]
  - trees: $O(\log n)$-size labels
  - planar graphs: $O^*(\sqrt{n})$, $\Omega^*(n^{1/3})$
  - general graphs: $\Omega^*(n)$
- graphs of highway dimension h: $O(h \log(h) \log(D))$ [ADFGW 11]

Example: Hypercube



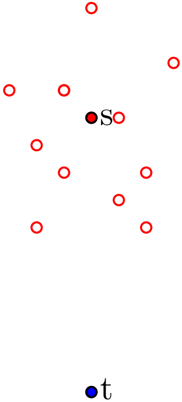
Standard binary vertex names yield $n \log n$ size labels

Outline

Hub Labeling Algorithm (HL)

Hub Labeling

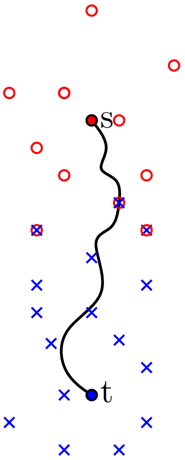
- $L(v) = \{(w, \text{dist}(v,w)) : w \in H(v)\}$, where $H(v) \subset V$ is a set of hubs of v



Hub Labeling Algorithm (HL)

Hub Labeling

- $L(v) = \{(w, \text{dist}(v, w)) : w \in H(v))\}$,
  where $H(v) \subset V$ is a set of hubs of v
- Labels satisfy the cover property:
  for all s, t, a shortest s-t path intersects $L(s) \cap L(t)$



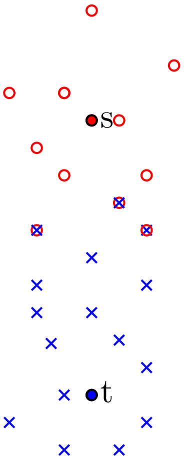
Hub Labeling Algorithm (HL)

Hub Labeling

- $L(v) = \{(w, \text{dist}(v,w)) : w \in H(v))\}$,
  where $H(v) \subset V$ is a set of hubs of v
- Labels satisfy the cover property:
  for all s, t, a shortest s-t path intersects $L(s) \cap L(t)$

s-t query

- Find vertex $w \in L(s) \cap L(t)$ ...



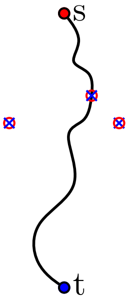
Hub Labeling Algorithm (HL)

Hub Labeling

- $L(v) = \{(w, \mathrm{dist}(v,w)) : w \in H(v))\}$, where $H(v) \subset V$ is a set of hubs of v
- Labels satisfy the cover property: for all s, t, a shortest s-t path intersects $L(s) \cap L(t)$

s-t query

- Find vertex $w \in L(s) \cap L(t)$ ...
- ... that minimizes $\mathrm{dist}(s,v) + \mathrm{dist}(v,t)$



Hub Labeling Algorithm (HL)

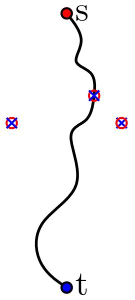
Hub Labeling

- $L(v) = \{(w, \text{dist}(v, w)) : w \in H(v))\}$, where $H(v) \subset V$ is a set of hubs of v
- Labels satisfy the cover property: for all s, t, a shortest s-t path intersects $L(s) \cap L(t)$

s-t query

- Find vertex $w \in L(s) \cap L(t)$ ...
- ... that minimizes $\text{dist}(s, v) + \text{dist}(v, t)$

Queries are efficient if labels are small



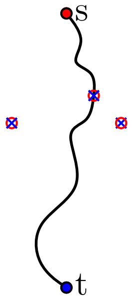
Hub Labeling Algorithm (HL)

Hub Labeling

- $L(v) = \{(w, \text{dist}(v,w)) : w \in H(v)\}$, where $H(v) \subset V$ is a set of hubs of v
- Labels satisfy the cover property: for all s, t, a shortest s-t path intersects $L(s) \cap L(t)$

s-t query

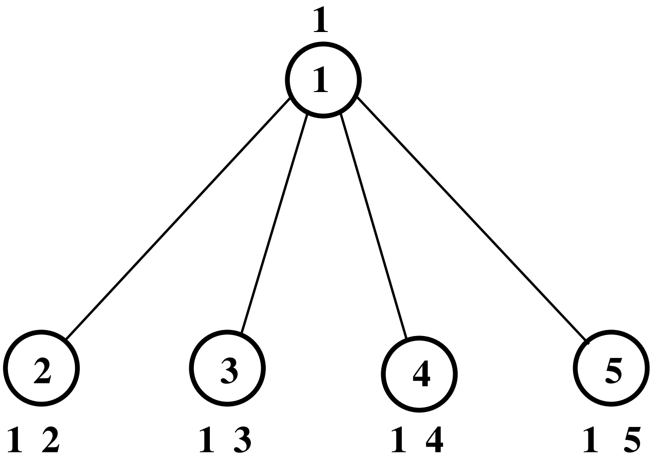
- Find vertex $w \in L(s) \cap L(t)$ ...
- ... that minimizes $\text{dist}(s,v) + \text{dist}(v,t)$



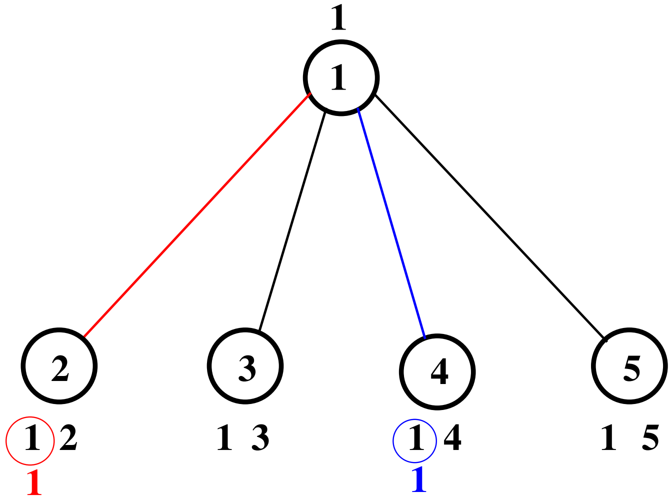
Queries are efficient if labels are small

Shortest paths are exact but label size may be suboptimal

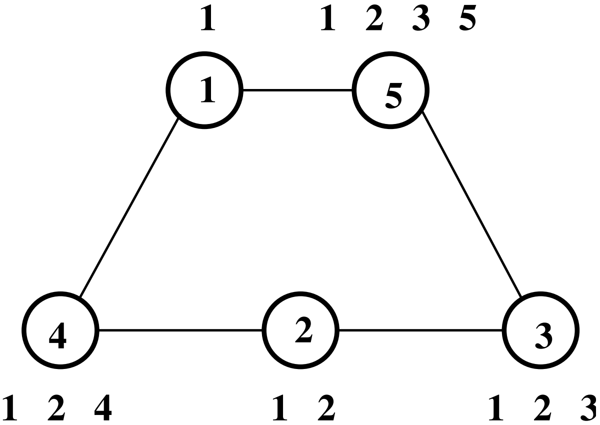
Example: Star Graph



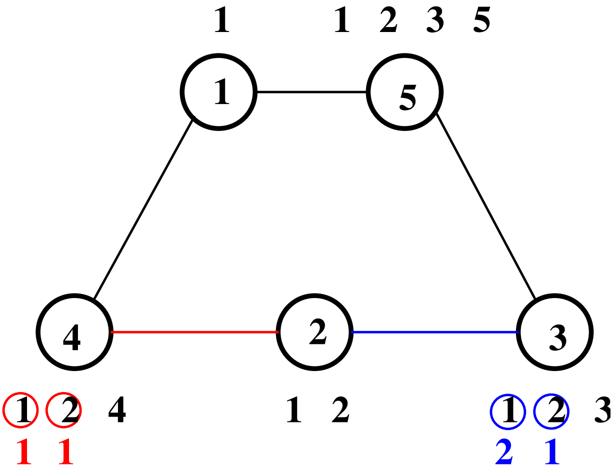
Example: Star Graph



Another Example



Another Example



Outline

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$

2	3	6	35	37	102	155	172

$L(t)$

2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$	2	3	6	35	37	102	155	172

$L(t)$	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$								
	2	3	6	35	37	102	155	172

$L(t)$						
	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$	2	3	6	35	37	102	155	172

$L(t)$	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$	2	3	6	35	37	102	155	172

$L(t)$	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$	2	3	6	35	37	102	155	172

$L(t)$	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$

$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$								
	2	3	6	35	37	102	155	172

$L(t)$						
	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$								
	2	3	6	35	37	102	155	172

$L(t)$						
	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$

$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$								
	2	3	6	35	37	102	155	172

$L(t)$						
	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$	2	3	6	35	37	102	155	172

$L(t)$	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$

$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$								
	2	3	6	35	37	102	155	172

$L(t)$						
	2	6	8	43	45	85

s-t query complexity

assume $|L(s)| \leq |L(t)|$

$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$

2	3	6	35	37	102	155	172

$L(t)$

2	6	8	43	45	85

Time estimate assuming memory-bound queries:

- $|L(s)| = |L(t)| = 100$
- 4 byte IDs and dist
- 128 byte cache lines
- 50ns latency
- $2 \cdot \lceil 100 \cdot 8/128 \rceil \cdot 50 =$ 700ns

s-t query complexity

assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality

Query Complexity

Label parameters

- $|L(v)|$: the number of hubs in $L(v)$
- size: $|L| = \sum_V |L(v)|$
- max label size: $M = \max_V |L(v)|$

$L(s)$	2	3	6	35	37	102	155	172

$L(t)$	2	6	8	43	45	85

Time estimate assuming memory-bound queries:

- $|L(s)| = |L(t)| = 100$
- 4 byte IDs and dist
- 128 byte cache lines
- 50ns latency
- $2 \cdot \lceil 100 \cdot 8/128 \rceil \cdot 50 =$ 700ns

s-t query complexity

assume $|L(s)| \leq |L(t)|$

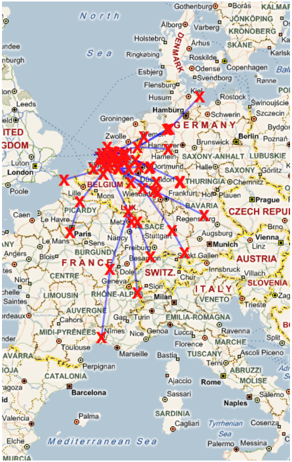
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

- $O(|L(s)| + |L(t)|) = O(M)$; good locality
- $O(|L(s)|)$ [ST 07]

Performance on Road Networks

Fast HL implementations

- implementation motivated by better query bounds [ADFGW 11]
- surprisingly small labels
- fastest distance oracles for road networks



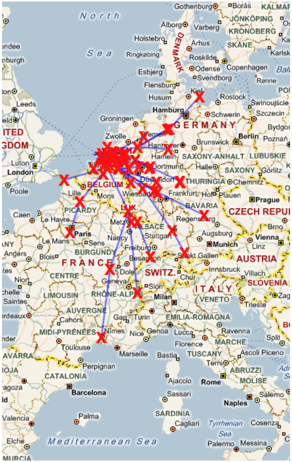
Performance on Road Networks

Fast HL implementations

- implementation motivated by better query bounds [ADFGW 11]
- surprisingly small labels
- fastest distance oracles for road networks

Western Europe, $n = 18$ Mil, $m = 42$ Mil

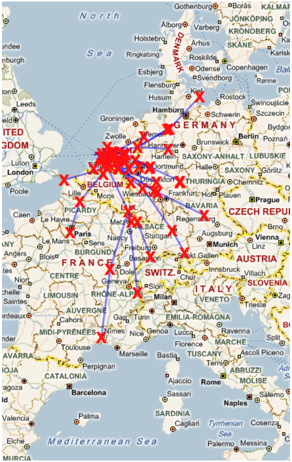
variant	prep (h:m)	$\lvert L\rvert / n$	GB	$[ns]$
HL	0:03	98	22.5	700
HL-15	0:05	78	18.8	556
HL-17	0:25	75	18.0	546
HL-R	5:43	69	17.7	508



Performance on Road Networks

Fast HL implementations

- implementation motivated by better query bounds [ADFGW 11]
- surprisingly small labels
- fastest distance oracles for road networks



Western Europe, $n = 18$ Mil, $m = 42$ Mil

variant	prep (h:m)	$\lvert L \rvert / n$	GB	[ns]
HL	0:03	98	22.5	700
HL-15	0:05	78	18.8	556
HL-17	0:25	75	18.0	546
HL-R	5:43	69	17.7	508

Memory-bound assumption verified

Beyond Road Networks: RXL [DGPW 14]

instance	$n(K)$	m/n	prep (h:m)	$\lvert L\rvert/n$	MB	[μs]
fla-t	1 070	2.5	0:02	41	261	0.5
buddha	544	6.0	0:02	92	180	0.9
buddha-w	544	6.0	0:11	336	953	2.9
rgg20	1 049	13.1	0:16	220	807	2.0
rgg20-w	1 049	13.1	1:00	589	3 154	4.9
WikiTalk	2 394	2.0	0:17	60	626	0.5
Indo	1 383	12.0	0:04	27	218	0.4
Skitter-u	1 696	13.1	0:47	274	1 075	2.3
MetrcS	2 250	19.2	0:38	117	593	0.8
eur-t	18 010	2.3	2:19	82	17 203	0.8
Hollywood	1 140	98.9	17:04	2 114	5 934	13.9
Indochin	7 415	25.8	4:07	66	3 917	0.7

External Memory and Database Queries

Queries require only two seek operations

External Memory and Database Queries

Queries require only two seek operations

Natural database implementation [ADFGW]

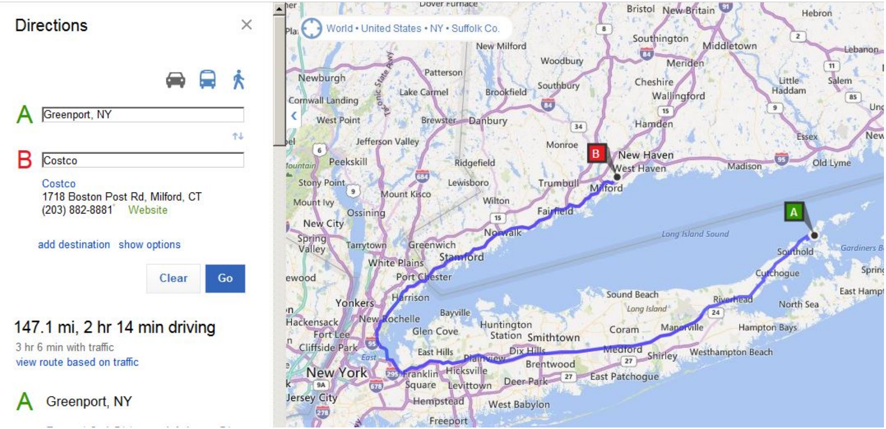
Algorithm 1: SQL_DIST

Input: source $s \in V$, target $t \in V$

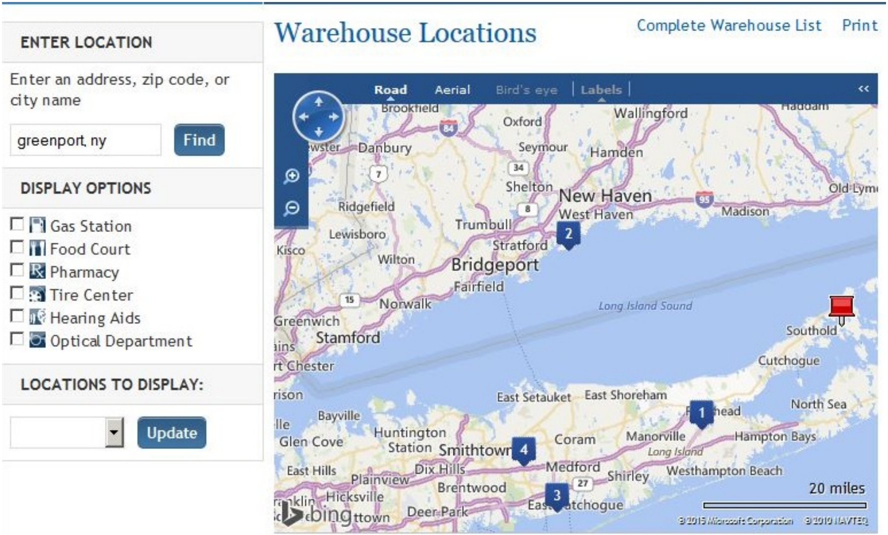
```
SELECT
    MIN(forward.dist+backward.dist)
FROM forward,backward
WHERE
    forward.node = s AND
    backward.node = t AND
    forward.hub = backward.hub
```

Example: Store Finder

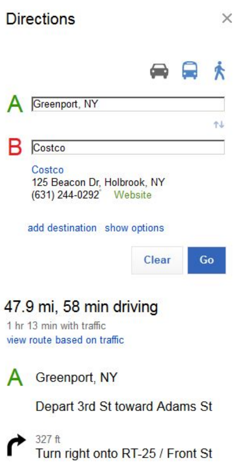
Example: Store Finder

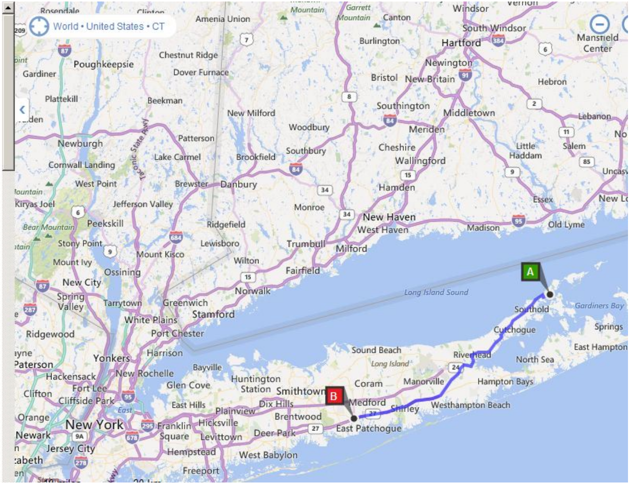


Example: Store Finder



Example: Store Finder





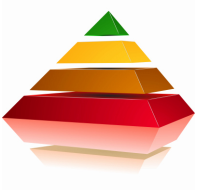
Example: Store Finder

Outline

Hierarchical Labels

Hierarchical Hub Labels (HHL) [ADGW 12]

$v \lesssim w$ if w is a hub of v (w more important)
L is hierarchical if $\lesssim$ is a partial order



- special class of HL
- can be polynomially bigger than HL [GPS 13]
- in practice, HHL are often small
- in practice, HHL can be computed faster than HL

Canonical HHL

- L respects a total order of vertices, r, if $\lesssim$ is consistent with r
- P_{uw} is the set of vertices on shortest paths from u to w

Canonical Labeling

- start with an empty labeling
- $\forall\ u, w$, let $v = \operatorname{argmax}_{v \in P_{uw}} r(x)$
- add v to $L(u)$ and $L(w)$

Canonical HHL

- L respects a total order of vertices, r, if $\lesssim$ is consistent with r
- P_{uw} is the set of vertices on shortest paths from u to w

Canonical Labeling

- start with an empty labeling
- $\forall\ u, w$, let $v = \text{argmax}_{v \in P_{uw}} r(x)$
- add v to $L(u)$ and $L(w)$

Canonical labels are exactly the minimal valid labels

- necessary: $v = \text{argmax}_{v \in P_{uw}} r(x)$ must be in $L(u)$ and $L(w)$
- sufficient: all vertex pairs are covered

Canonical HHL

- L respects a total order of vertices, r, if $\lesssim$ is consistent with r
- P_{uw} is the set of vertices on shortest paths from u to w

Canonical Labeling

- start with an empty labeling
- $\forall\ u, w$, let $v = \operatorname{argmax}_{v \in P_{uw}} r(x)$
- add v to $L(u)$ and $L(w)$

Canonical labels are exactly the minimal valid labels

- necessary: $v = \operatorname{argmax}_{v \in P_{uw}} r(x)$ must be in $L(u)$ and $L(w)$
- sufficient: all vertex pairs are covered

The definition implies a poly-time, but impractical, algorithm

Pruned Labeling (PL) Algorithm

[AIY 13]: compute canonical labeling form an order r

PL algorithm

- start with an empty L
- process vertices v in the order given by r (highest to lowest)
- run Dijkstra's search from v
  - before scanning w check the following condition
  - is $d(w) \geq$ (*estimate given by current labels*)?
  - if yes, prune w (do not scan)
- add v to the labels of all w scanned by Dijkstra

Pruned Labeling (PL) Algorithm

[AIY 13]: compute canonical labeling form an order r

PL algorithm

- start with an empty L
- process vertices v in the order given by r (highest to lowest)
- run Dijkstra's search from v
  - before scanning w check the following condition
  - is $d(w) \geq$ (*estimate given by current labels*)?
  - if yes, prune w (do not scan)
- add v to the labels of all w scanned by Dijkstra

Approximate PL complexity

- every scanned vertex is added to L
- $\approx O(|L|\frac{|L|}{n})$
- efficient if $|L|/n$ is small

PL Correctness

- For simplicity, assume unique shortest paths.
- Prove by induction on $|P_{uv}|$: $v = \text{argmax}_{x \in P_{uv}} r(x) \Rightarrow$ PL adds v
  - basis is trivial
  - by the inductive hypothesis, statement holds for the predecessor u' of u on the shortest u-v path (since $P_{u'v} \subset P_{uv}$).
  - Dijkstra's search from v scans u', setting $d(u)$ to correct distance
  - when u is scanned, $L(u) \cap P_{uv} = \emptyset$, so $d(u) <$ *(estimate given by current labels)* $= \infty$
  - u scanned, v added to $L(u)$ with $d(u)$
- $v = \text{argmax}_{x \in P_{uw}} r(x) \Rightarrow$ $v = \text{argmax}_{x \in P_{uv}} r(x)$ and $v = \text{argmax}_{x \in P_{vw}} r(x) \Rightarrow$ $v \in L(u)$ and $v \in L(w)$, with correct distances.

Example: labels on a path

Ordering matters!

- Sequential ordering: $\Omega(n^2)$ label size
- recursive "split in the middle" ordering: $O(n \log n)$ label size

HHL Vertex Ordering

PL allows separation of vertex ordering from label generation

Ordering requirements

- label quality (small size)
- efficiency

HHL Vertex Ordering

PL allows separation of vertex ordering from label generation

Ordering requirements

- label quality (small size)
- efficiency

HHL orderings

- bottom up [ADFGW 11]: works well for road networks, but not robust
  - related to contraction hierarchies [GSSD 08]
- by degree [AIY 13]: very fast, works on some networks but not robust
- greedy [ADGW 12]: slow but robust
  - can be made faster by sampling [DGPW 14]

Simple Bottom-Up Ordering

Order vertices from least to most important

- Choose a maximal independent set I using the least degree heuristic
- Order vertices of I, in the same order they were added to I
- Delete I and continue

Simple Bottom-Up Ordering

Order vertices from least to most important

- Choose a maximal independent set I using the least degree heuristic
- Order vertices of I, in the same order they were added to I
- Delete I and continue

Remarks

- This folklore ordering is OK
- There are better ordering heuristics [GSSD 08]
- You can experiment with this and other orderings using Ruslan Savchenko's code for basic HHL primitives `https://github.com/savrus/hl`

Greedy Ordering [ADGW 12]

v covers $\{u, w\}$ is $\exists$ a u-w SP passing through v

Greedy ordering (most to least important)

[Abraham at al. 12]

- $U = V \times V$
- while there are unprocessed vertices
- pick a vertex v that covers most pairs in U as the next highest in the ordering
- update U by deleting the pairs that v covers

Greedy Ordering [ADGW 12]

v covers $\{u, w\}$ is $\exists$ a u-w SP passing through v

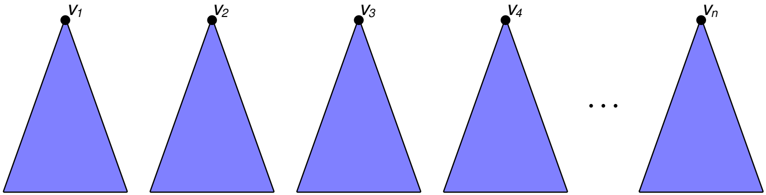
Greedy ordering (most to least important)

[Abraham at al. 12]

- $U = V \times V$
- while there are unprocessed vertices
- pick a vertex v that covers most pairs in U as the next highest in the ordering
- update U by deleting the pairs that v covers

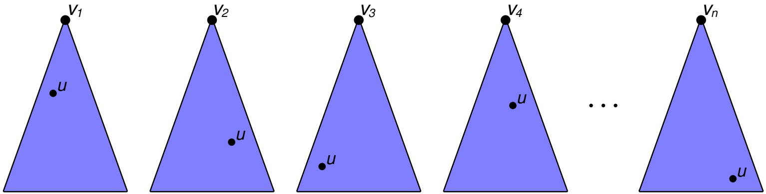
- next we describe data structures
- for simplicity assume that shortest paths are unique

Engineering Efficient Implementation of Greedy



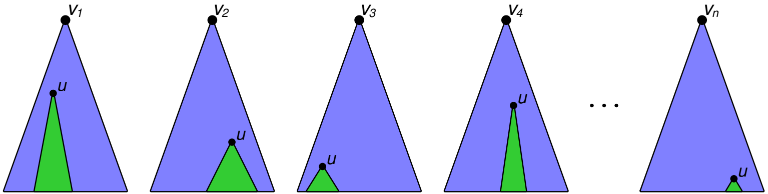
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i

Engineering Efficient Implementation of Greedy



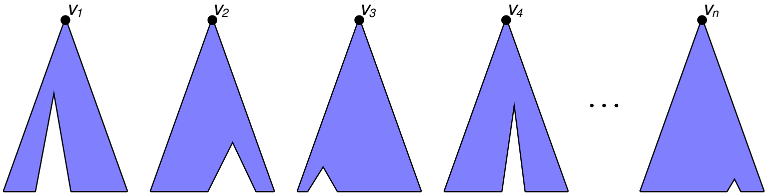
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u

Engineering Efficient Implementation of Greedy



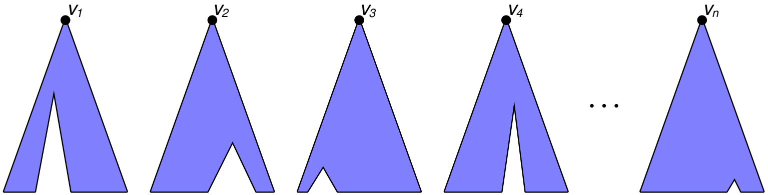
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order

Engineering Efficient Implementation of Greedy



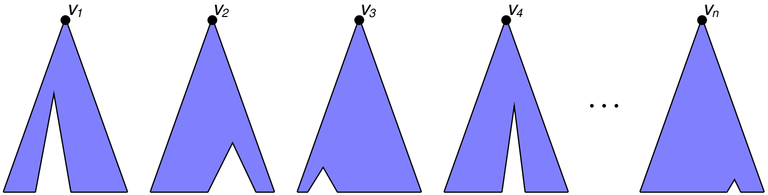
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts

Engineering Efficient Implementation of Greedy



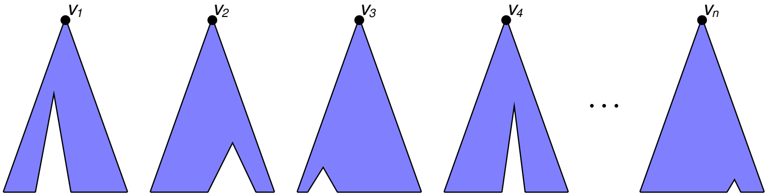
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts
- the trees represent U

Engineering Efficient Implementation of Greedy



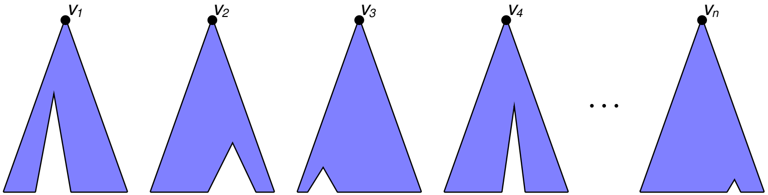
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts
- the trees represent U
- complexity $O(nDij(n,m))$ time, $O(n^2)$ space

Engineering Efficient Implementation of Greedy



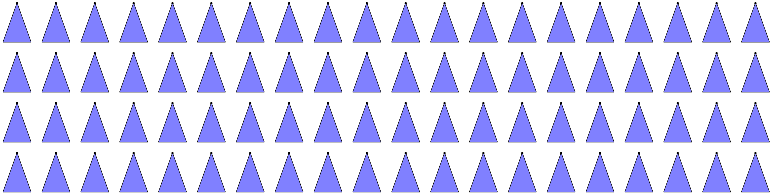
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts
- the trees represent U
- complexity $O(nDij(n,m))$ time, $O(n^2)$ space
- this is too much!

Engineering Efficient Implementation of Greedy



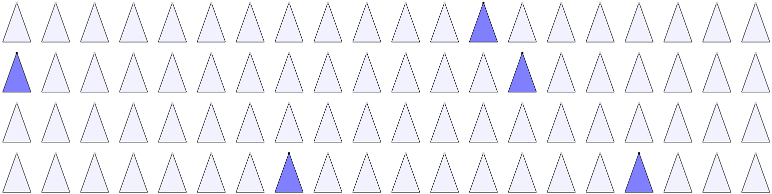
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts
- the trees represent U
- complexity $O(nDij(n,m))$ time, $O(n^2)$ space
- this is too much!
- use sampling to reduce time and space requirements

RXL: Relaxed Greedy Labeling [Delling et al. 14]



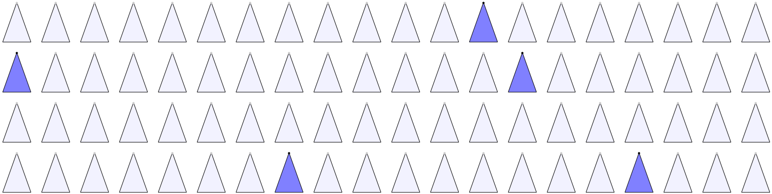
- maintain a sample of $\ll n$ trees (within tree node budget)

RXL: Relaxed Greedy Labeling [Delling et al. 14]



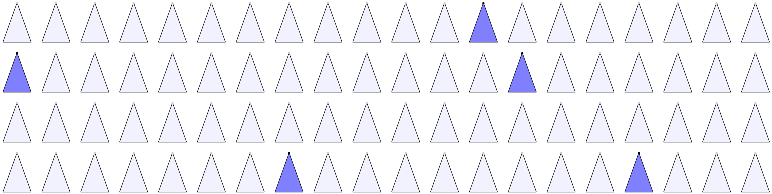
- maintain a sample of $\ll n$ trees (within tree node budget)

RXL: Relaxed Greedy Labeling [Delling et al. 14]



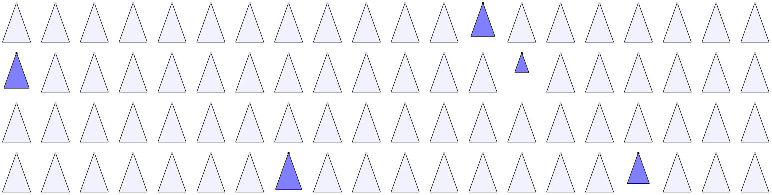
- maintain a sample of $\ll n$ trees (within tree node budget)
- use #descendants in the sample to estimate coverage of every u
  - sample is biased (e.g., vertices close to roots)
  - eliminate outliers

RXL: Relaxed Greedy Labeling [Delling et al. 14]



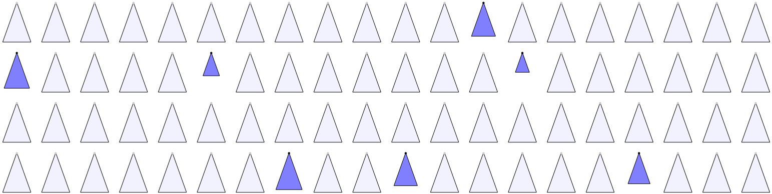
- maintain a sample of $\ll n$ trees (within tree node budget)
- use #descendants in the sample to estimate coverage of every u
  - sample is biased (e.g., vertices close to roots)
  - eliminate outliers
- add vertex u with the highest coverage estimate

RXL: Relaxed Greedy Labeling [Delling et al. 14]



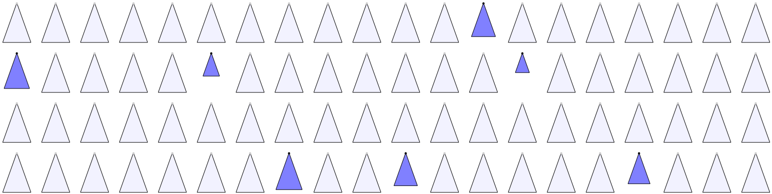
- maintain a sample of $\ll n$ trees (within tree node budget)
- use #descendants in the sample to estimate coverage of every u
  - sample is biased (e.g., vertices close to roots)
  - eliminate outliers
- add vertex u with the highest coverage estimate
- remove descendants of u in sampled trees

RXL: Relaxed Greedy Labeling [Delling et al. 14]



- maintain a sample of $\ll n$ trees (within tree node budget)
- use #descendants in the sample to estimate coverage of every u
  - sample is biased (e.g., vertices close to roots)
  - eliminate outliers
- add vertex u with the highest coverage estimate
- remove descendants of u in sampled trees
- add new (pruned using PL) trees as the budget permits

RXL: Relaxed Greedy Labeling [Delling et al. 14]



- maintain a sample of $\ll n$ trees (within tree node budget)
- use #descendants in the sample to estimate coverage of every u
  - sample is biased (e.g., vertices close to roots)
  - eliminate outliers
- add vertex u with the highest coverage estimate
- remove descendants of u in sampled trees
- add new (pruned using PL) trees as the budget permits
- sample size can be adjusted to trade time/space for quality

Degree vs. RXL

			degree		RXL	
instance	$n(K)$	m/n	prep (h:m)	$\|L\|/n$	prep (h:m)	$\|L\|/n$
fla-t	1 070	2.5	0:22	172	0:02	41
buddha	544	6.0	0:02	290	0:02	92
buddha-w	544	6.0	0:24	1 165	0:11	336
rgg20	1 049	13.1	0:47	1 136	0:16	220
rgg20-w	1 049	13.1	14:43	5 603	1:00	589
WikiTalk	2 394	2.0	0:05	68	0:17	60
Indo	1 383	12.0	0:04	172	0:04	27
Skitter-u	1 696	13.1	0:32	457	0:47	274
MetrcS	2 250	19.2	0:06	132	0:38	117
eur-t	18 010	2.3	–	–	2:19	82
Hollywood	1 140	98.9	10:40	2 921	17:04	2 114
Indochin	7 415	25.8	3:20	540	4:07	66

Performance on Road Networks (Revisited)

Western Europe, $n = 18M$, $m = 24M$

variant	prep (h:m)	$\lvert L \rvert / n$	GB	$[ns]$
HL	0:03	98	22.5	700
HL-15	0:05	78	18.8	556
HL-17	0:25	75	18.0	546
HL-R	5:43	69	17.7	508

Bottom-up ordering, plus

- greedy reordering of 2^{15} (HL-15) or 2^{17} (HL-17) top vertices
- range optimization (reordering of overlapping intervals)

Outline

Approximating Optimal Labels

Theoretical results

- optimizing $|L|$:
  - poly-time $O(\log n)$ approximation in $O(n^5)$ time [CHKZ 03]
  - $O(n^3 \log n)$ time improvement [DGSW 14]
  - NP-hard [BGKSW 15]
- optimizing M:
  - poly-time $O(\log n)$ approximation in $O(n^5)$ time [BGGN 13]
  - $O(n^3 \log^2 n)$ time improvement [DGSW 14]

Approximating Optimal Labels

Theoretical results

- optimizing $|L|$:
  - poly-time $O(\log n)$ approximation in $O(n^5)$ time [CHKZ 03]
  - $O(n^3 \log n)$ time improvement [DGSW 14]
  - NP-hard [BGKSW 15]
- optimizing M:
  - poly-time $O(\log n)$ approximation in $O(n^5)$ time [BGGN 13]
  - $O(n^3 \log^2 n)$ time improvement [DGSW 14]

[DGSW 14] improvement story

- introduced a heuristic improvement of [CHKZ 03]
- proved a better bound for the heuristic

Cohen at al. Algorithm ($\log$-HL)

- for a (partial) labeling L, a pair u, w is covered if $L(u) \cap L(w)$ contains a vertex on u–w SP
- v covers u, w if there is a u–w SP through v

Cohen at al. Algorithm ($\log$-HL)

- for a (partial) labeling L, a pair u, w is covered if $L(u) \cap L(w)$ contains a vertex on u–w SP
- v covers u, w if there is a u–w SP through v

$\log$-HL algorithm sketch

1. start with an empty L, U containing all vertex pairs
2. add a vertex v to the labels of a set of vertices S
3. remove covered pairs from U
4. if $U = \emptyset$ halt, otherwise go to 2

Cohen at al. Algorithm ($\log$-HL)

- for a (partial) labeling L, a pair u, w is covered if $L(u) \cap L(w)$ contains a vertex on u–w SP
- v covers u, w if there is a u–w SP through v

$\log$-HL algorithm sketch

1. start with an empty L, U containing all vertex pairs
2. add a vertex v to the labels of a set of vertices S

   Pick v and S as follows:

$$v, S = \operatorname{argmax} \max_{v \in V} \max_{S \subseteq V} \frac{\text{\# pairs covered if we add } v \text{ to } S}{|S|}$$

3. remove covered pairs from U
4. if $U = \emptyset$ halt, otherwise go to 2

The resulting labeling need not be hierarchical

Center Graphs and MDS

Center graph

$G_v = (V, E_v)$ where $(u, w) \in E_v$ if $u, w \in U$ and v covers u, w

- graph density: (#edges)/(#vertices)
- MDS problem: find a maximum density vertex-induced subgraph

Center Graphs and MDS

Center graph

$G_v = (V, E_v)$ where $(u, w) \in E_v$ if $u, w \in U$ and v covers u, w

- graph density: (#edges)/(#vertices)
- MDS problem: find a maximum density vertex-induced subgraph

MDS for G_v

$$\max_{S \subseteq V} \frac{\text{\# pairs covered if we add } v \text{ to } S}{|S|}$$

Step (2): maximize MDS over all G_v

MDS complexity

- polynomial using parametric flows
- linear time 2-approximation [KP 94] (2-MDS)

2-Approximate MDS

2-MDS Algorithm

1. while more than one vertex remains
2. delete a minimum degree vertex
3. update degrees
4. goto (1)
5. return the densest subgraph seen

2-Approximate MDS

2-MDS Algorithm

1. while more than one vertex remains
2. delete a minimum degree vertex
3. update degrees
4. goto (1)
5. return the densest subgraph seen

Correctness intuition

- A “small degree” vertex is not in MDS
- If vertex degrees of G are “not small”, G is a 2-MDS

2-Approximate MDS

2-MDS Algorithm

1. while more than one vertex remains
2. delete a minimum degree vertex
3. update degrees
4. goto (1)
5. return the densest subgraph seen

Correctness intuition

- A “small degree” vertex is not in MDS
- If vertex degrees of G are “not small”, G is a 2-MDS

Problem: deleting 2-MDS may not reduce MDS value

Eager-Lazy Algorithm [DGSW 14]

α-eager evaluation (a modification of 2-MDS algorithm)

- μ is an upper bound on MDS value of G, $\alpha > 1$
- while MDS value of $G < \mu/(2\alpha)$ delete min degree vertex
- G': remaining graph; $G - G'$ has MDS value $\leq \mu/\alpha$

Eager-Lazy Algorithm [DGSW 14]

α-eager evaluation (a modification of 2-MDS algorithm)

- μ is an upper bound on MDS value of G, $\alpha > 1$
- while MDS value of $G < \mu/(2\alpha)$ delete min degree vertex
- G': remaining graph; $G - G'$ has MDS value $\leq \mu/\alpha$

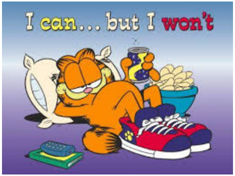
Center graph densities are monotone

Eager-lazy algorithm

- start with empty L, $U = V \times V$
- compute upper bounds μ_v on MDS values of G_v
- while $U \neq \emptyset$
- $v = \mathrm{argmax}(\mu_v)$; apply α-eager evaluation to G_v
- add v to the vertices of G', $G = G - G'$, update U
- $\mu_v = \mu_v/\alpha$

Eager-Lazy Algorithm Analysis

- each iteration is $O(n^2)$ (vs. $O(n^3)$) (lazy)
- decreases μ_v by a constant factor (eager)
- each v chosen $O(\log n)$ times (vs. $O(n^2)$)
- $O(n^3 \log n)$ bound (vs. $O(n^5)$)
- $O(n^2)$ space if center graphs maintained implicitly (vs. $O(n^3)$)

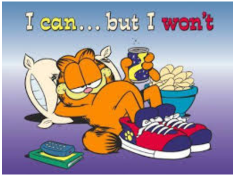


Eager-Lazy Algorithm Analysis

- each iteration is $O(n^2)$ (vs. $O(n^3)$) (lazy)
- decreases μ_v by a constant factor (eager)
- each v chosen $O(\log n)$ times (vs. $O(n^2)$)
- $O(n^3 \log n)$ bound (vs. $O(n^5)$)
- $O(n^2)$ space if center graphs maintained implicitly (vs. $O(n^3)$)

From practice to theory

- $\log$-HL picks same v consecutively
- use second-densest subgraph seen

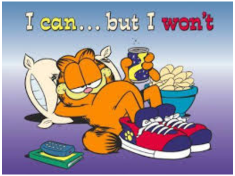


Eager-Lazy Algorithm Analysis

- each iteration is $O(n^2)$ (vs. $O(n^3)$) (lazy)
- decreases μ_v by a constant factor (eager)
- each v chosen $O(\log n)$ times (vs. $O(n^2)$)
- $O(n^3 \log n)$ bound (vs. $O(n^5)$)
- $O(n^2)$ space if center graphs maintained implicitly (vs. $O(n^3)$)

From practice to theory

- $\log$-HL picks same v consecutively
- ~~use second-densest subgraph seen~~
- use α-eager evaluation
- prove the new bound



Experimental Results for Approximation Algorithms

- log-HL: efficient implementation of [CHKZ 03]
- log-HL+: the implementation of [DGSW 14]
- log-HL+ labels are not much bigger
- log-HL+ is faster but still does not scale well

		time (s)		$\|L\|/n$	
instance	n	log-HL	log-HL+	log-HL	log-HL+
email	1133	109	47	30.0	30.4
polblogs	1222	376	145	25.2	25.5
venus	2838	978	558	27.3	28.0
alue5067	3524	2971	2486	23.4	24.5
ksw-64	4096	2319	901	81.4	82.3
hep-th	5835	6375	1479	38.7	39.2
berlin	10370	16027	8649	20.5	21.3
PGPgiant	10680	19114	3339	19.1	19.4

From Practice to Theory

Recent results on HHL [BGKSW 15]

- computing optimal HHL is NP-hard
- greedy algorithm approximation ratio
  - $O(n^{1/2} \log n)$ upper bound
  - $\Omega(n^{1/2})$ lower bound
- weighted greedy (similar to $\log$-HL) algorithm approx ratio
  - $O(n^{1/2} \log n)$ upper bound
  - $\Omega(n^{1/3})$ lower bound

From Practice to Theory

Recent results on HHL [BGKSW 15]

- computing optimal HHL is NP-hard
- greedy algorithm approximation ratio
  - $O(n^{1/2} \log n)$ upper bound
  - $\Omega(n^{1/2})$ lower bound
- weighted greedy (similar to $\log$-HL) algorithm approx ratio
  - $O(n^{1/2} \log n)$ upper bound
  - $\Omega(n^{1/3})$ lower bound
- distance greedy algorithm
  - $M = O(h \log n \log D)$ (h: highway dimension; D: diameter)
  - $O(n^{1/2} \log n \log D)$ upper and $\Omega(n^{1/2})$ lower bounds on approx ratio

From Practice to Theory

Recent results on HHL [BGKSW 15]

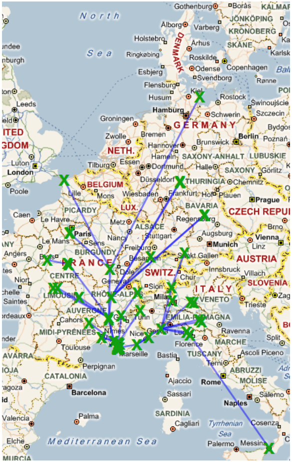
- computing optimal HHL is NP-hard
- greedy algorithm approximation ratio
  - $O(n^{1/2} \log n)$ upper bound
  - $\Omega(n^{1/2})$ lower bound
- weighted greedy (similar to $\log$-HL) algorithm approx ratio
  - $O(n^{1/2} \log n)$ upper bound
  - $\Omega(n^{1/3})$ lower bound
- distance greedy algorithm
  - $M = O(h \log n \log D)$ (h: highway dimension; D: diameter) 
  - $O(n^{1/2} \log n \log D)$ upper and $\Omega(n^{1/2})$ lower bounds on approx ratio

Great open problem

Is there an $O(\log n)$-approximation algorithm for optimal HHL?

Remarks

- fruitful interaction between theory and experimentation
- impact beyond mainstream algorithms community
- highly practical algorithms
- opportunities for technology transfer
- active area, open problems remain



Thank You!

Joint work with

Ittai Abraham, Maxim Babenko, Daniel Delling, Haim Kaplan, Thomas Pajor, Ruslan Savchenko, Mathis Weller, Renato Werneck