Hub Labeling Algorithms

Andrew V. Goldberg

Microsoft Research – Silicon Valley

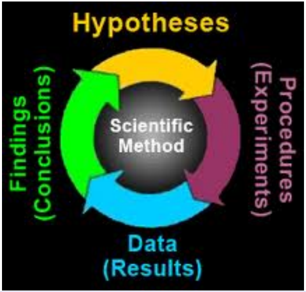
Hub Labeling Algorithms

Andrew V. Goldberg

~~Microsoft Research – Silicon Valley~~

Job Seeker

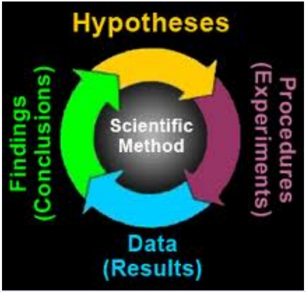
Science of Algorithmics



Iteration of

- algorithm design
- algorithm analysis
- algorithm engineering
- experimental evaluation
- analysis of the results

Science of Algorithmics



Iteration of

- algorithm design
- algorithm analysis
- algorithm engineering
- experimental evaluation
- analysis of the results

Goal: Practical and theoretically justified algorithms

Example: Theory vs. Practice

Example: Theory vs. Practice

Bellman-Ford algorithm (BF)

- fundamental algorithm
  - discovered in the 1950's [B 58, F 56, M 57]
  - taught to CS and OR undergraduates
  - poor practical performance

Example: Theory vs. Practice

Bellman-Ford algorithm (BF)

- fundamental algorithm
  - discovered in the 1950's [B 58, F 56, M 57]
  - taught to CS and OR undergraduates
  - poor practical performance
- claimed practical improvements
  - Pape [P 74], threshold [DGKK 79], Pallatino [P 84], ...
  - not robust, time bounds worse than those for BF

Example: Theory vs. Practice

Bellman-Ford algorithm (BF)

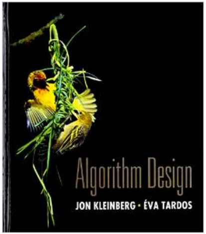
- fundamental algorithm
  - discovered in the 1950's [B 58, F 56, M 57]
  - taught to CS and OR undergraduates
  - poor practical performance
- claimed practical improvements
  - Pape [P 74], threshold [DGKK 79], Pallatino [P 84], ...
  - not robust, time bounds worse than those for BF
- BF with subtree disassambly
  - introduced by Tarjan [T 81] to amortize negative cycle detection over BF computation
  - unnoticed for 15 years

Example: Theory vs. Practice

Bellman-Ford algorithm (BF)

- fundamental algorithm
  - discovered in the 1950's [B 58, F 56, M 57]
  - taught to CS and OR undergraduates
  - poor practical performance
- claimed practical improvements
  - Pape [P 74], threshold [DGKK 79], Pallatino [P 84], ...
  - not robust, time bounds worse than those for BF
- BF with subtree disassambly
  - introduced by Tarjan [T 81] to amortize negative cycle detection over BF computation
  - unnoticed for 15 years
  - greatly improves practical performance [CG 96]

Example: Theory vs. Practice



Bellman-Ford algorithm (BF)

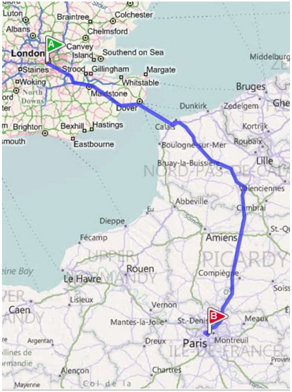
- fundamental algorithm
  - discovered in the 1950's [B 58, F 56, M 57]
  - taught to CS and OR undergraduates
  - poor practical performance
- claimed practical improvements
  - Pape [P 74], threshold [DGKK 79], Pallatino [P 84], ...
  - not robust, time bounds worse than those for BF
- BF with subtree disassambly
  - introduced by Tarjan [T 81] to amortize negative cycle detection over BF computation
  - unnoticed for 15 years
  - greatly improves practical performance [CG 96]
  - ... while integrating negative cycle detection
  - now in a textbook [KT 05]

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

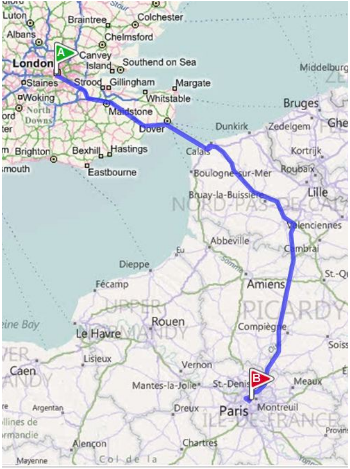
Need a fast and robust approach

Input

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

Query (multiple for the same network)

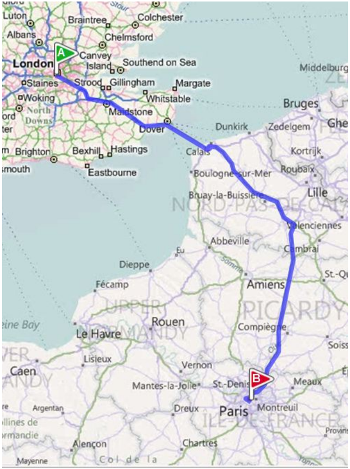
- Given origin s and destination t, find an optimal path from s to t



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)



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

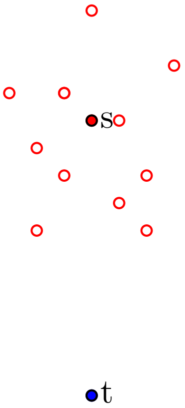
Label Sizes

- 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]

Hub Labeling Algorithm (HL)

Hub Labeling

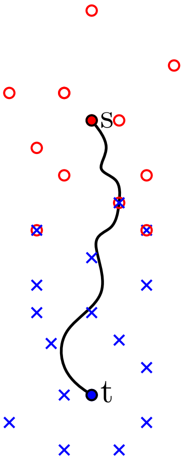
- $L(v) = \{(w, \text{dist}(v, w)) : w \in H(v))\}$, where $H(v) \subset V$ is a set of hubs of v



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 $L(s) \cap L(t)$



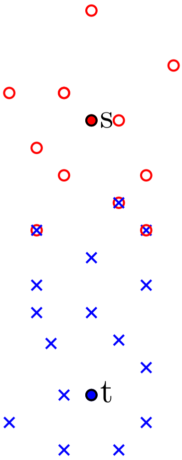
Hub Labeling Algorithm (HL)

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 $w \in L(s) \cap L(t)$ ...



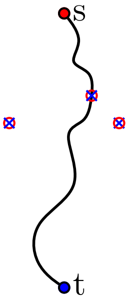
Hub Labeling Algorithm (HL)

Hub Labeling

- $L(v) = \{(w, \text{dist}(v,w)) : w \in H(v)\}$, 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

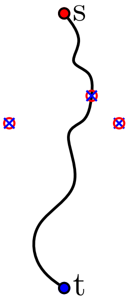
- Find vertex $w \in L(s) \cap L(t) \ldots$
- $\ldots$ that minimizes $\text{dist}(s,v) + \text{dist}(v,t)$



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 $L(s) \cap L(t)$

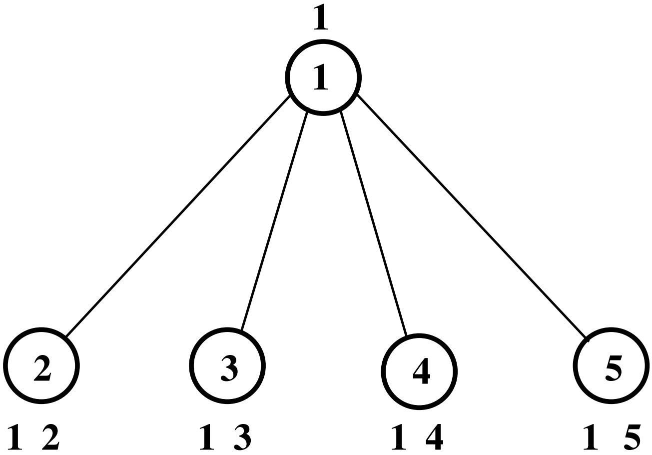


s-t query

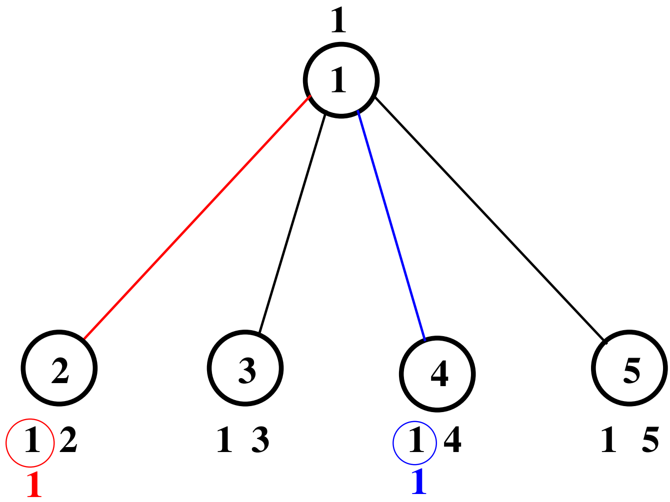
- Find vertex $w \in L(s) \cap L(t) \ldots$
- $\ldots$ that minimizes dist$(s,v)+$ dist(v,t)

Queries are efficient if labels are small

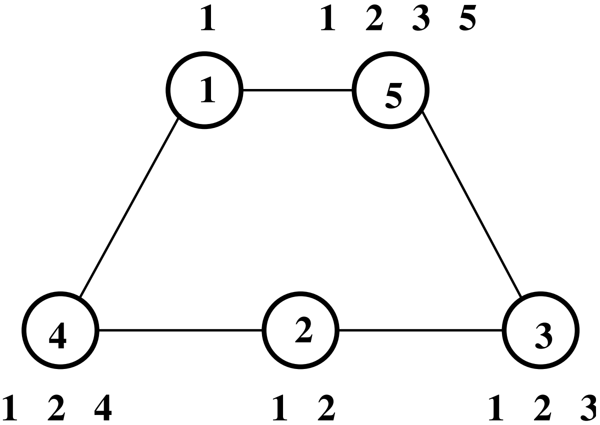
Example: Star Graph



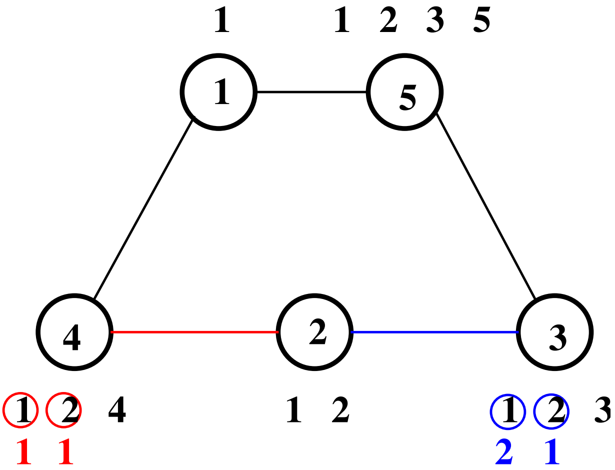
Example: Star Graph



Another Example



Another Example



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: $M = \max_V |L(v)|$

$L(s)$

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: $M = \max_V |L(v)|$

$$L(s)\ \boxed{2}\boxed{3}\boxed{6}\boxed{35}\boxed{37}\boxed{102}\boxed{155}\boxed{172}$$

$$L(t)\ \boxed{2}\boxed{6}\boxed{8}\boxed{43}\boxed{45}\boxed{85}$$

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

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 $|L(s)| \le |L(t)|$

$\forall v$, 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 $|L(s)| \le |L(t)|$

$\forall v$, 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: $M = \max_V |L(v)|$

$$L(s)\ \begin{array}{|c|c|c|c|c|c|c|c|}\hline 2 & 3 & 6 & 35 & 37 & 102 & 155 & 172 \\ \hline \end{array}$$

$$L(t)\ \begin{array}{|c|c|c|c|c|c|}\hline 2 & 6 & 8 & 43 & 45 & 85 \\ \hline \end{array}$$

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

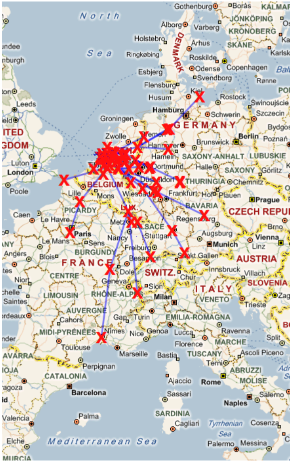
assume $|L(s)| \leq |L(t)|$
$\forall v$, sort v's hubs by vertex IDs; query intersects sorted lists

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



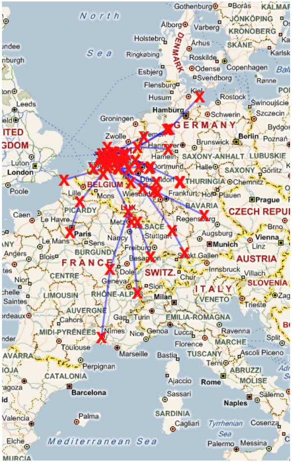
Performance on Road Networks

Fast HL implementations

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

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

variant	prep (h:m)	$\|L\|/n$	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



Beyond Road Networks: RXL Implementation

instance	$n(K)$	m/n	prep (h:m)	$\|L\|/n$	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

Outline

Approximating Optimal Labels

Theoretical results

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

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

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

- $\text{density} = \frac{\Sigma_V \mathsf{deg}(v)}{2|V|}$
- if for all v deg$(v) \geq \mu$, then $\text{density} \geq \mu/2$
- if μ is the MDS value and $\mathsf{deg}(v) < \mu$, then v is not in an MDS

2-Approximate MDS

- $\text{density} = \frac{\Sigma_V \text{deg}(v)}{2|V|}$
- if for all v deg$(v) \geq \mu$, then $\text{density} \geq \mu/2$
- if μ is the MDS value and $\text{deg}(v) < \mu$, then v is not in an MDS

2-MDS Algorithm

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

Eager-Lazy Algorithm [DGSW 14]

α-eager evaluation (suggested by a heuristic)

- 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 (suggested by a heuristic)

- 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 $\leq \mu/\alpha$

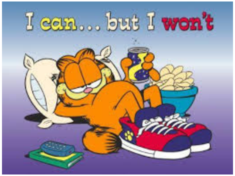
Center graph densities are monotone

Eager-lazy algorithm

- start with empty L, $U = V \times V$
- compute upper bounds μ_v on MDS values of G_v
- while $U \neq \emptyset$
- $v = \text{argmax}(\mu_v)$; 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. $O(n^3)$)
- decreases μ_v by a constant factor
- 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. $O(n^3)$)

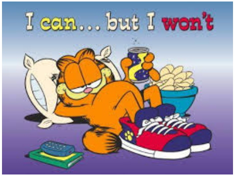


Eager-Lazy Algorithm Analysis

- each iteration is $O(n^2)$ (vs. $O(n^3)$)
- decreases μ_v by a constant factor
- 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

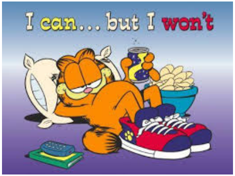


Eager-Lazy Algorithm Analysis

- each iteration is $O(n^2)$ (vs. $O(n^3)$)
- decreases μ_v by a constant factor
- 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: Eager-Lazy

- G1.0: efficient implementation of $\log$-HL
- G1.1: eager-lazy implementation with $\alpha = 1.1$
- G1.1 labels are not much bigger
- G1.1 is faster but still does not scale well

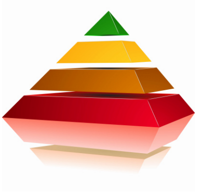
		time (s)		$\|L\|/n$	
instance	n	G1.0	G1.1	G1.0	G1.1
email	1133	109	47	30.0	30.4
polblogs	1222	376	145	25.2	25.5
venus	2838	978	558	27.3	28.0
alue5067	3524	2971	2486	23.4	24.5
ksw-64	4096	2319	901	81.4	82.3
hep-th	5835	6375	1479	38.7	39.2
berlin	10370	16027	8649	20.5	21.3
PGPgiant	10680	19114	3339	19.1	19.4

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 larger [GPS 13] than HL
- 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 $v = \text{argmax}_{v \in P_{uw}} r(x)$
- add v to $L(u)$ and $L(w)$

Canonical HHL

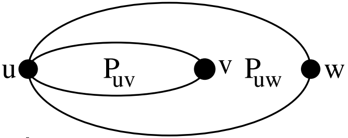
- L respects a total order of vertices, r, if $\lesssim$ is consistent with r
- P_{uw} is the set of vertices on shortest paths from u to w

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 $L(w)$

Validity and minimality

- cover property: v covers $[u, w]$
- v must be in $L(u)$ to cover $[u, v]$
- similarly v must be in $L(w)$
- in fact, all minimal HHL are canonical



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 of importance
- 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 of importance
- 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

Fact: PL computes canonical HHL

PL (cont.)

Approximate PL complexity

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

PL (cont.)

Approximate PL complexity

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

Can separate vertex ordering and label generation

HHL Vertex Ordering

Requirements

- label quality (small size)
- efficiency

HHL Vertex Ordering

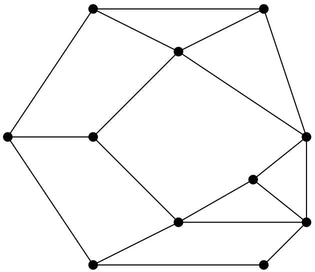
Requirements

- label quality (small size)
- efficiency

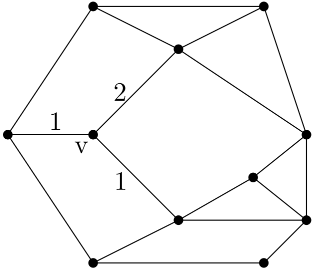
HHL orderings

- bottom up [ADFGW 11]: works well for road networks, but not robust
- 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]

- process vertices from least to most important

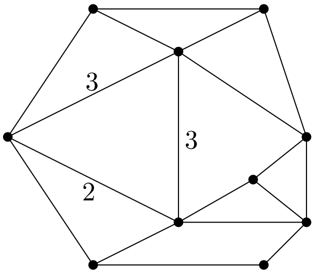


- process vertices from least to most important
- temporarily remove the next vertex v from the graph



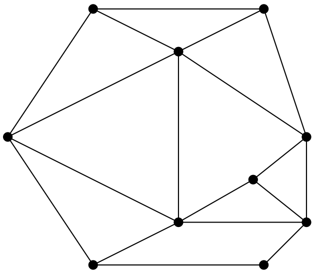
Bottom-Up Ordering [ADFGW 11]

- process vertices from least to most important
- temporarily remove the next vertex v from the graph
  - add shortcuts to preserve distances (if needed)



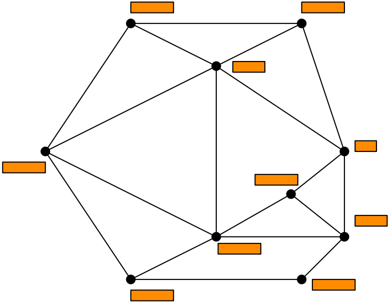
Bottom-Up Ordering [ADFGW 11]

- process vertices from least to most important
- temporarily remove the next vertex v from the graph
  - add shortcuts to preserve distances (if needed)



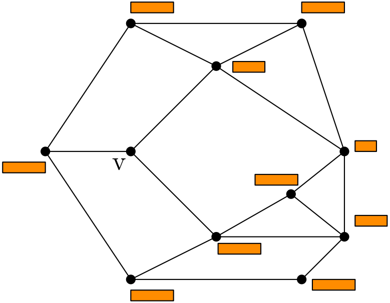
Bottom-Up Ordering [ADFGW 11]

- process vertices from least to most important
- temporarily remove the next vertex v from the graph
  - add shortcuts to preserve distances (if needed)
- recursively compute labels on smaller graph



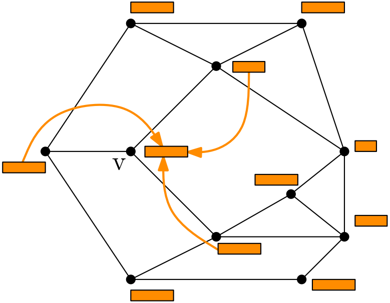
Bottom-Up Ordering [ADFGW 11]

- process vertices from least to most important
- temporarily remove the next vertex v from the graph
  - add shortcuts to preserve distances (if needed)
- recursively compute labels on smaller graph
- reinsert v and build label from its neighbors



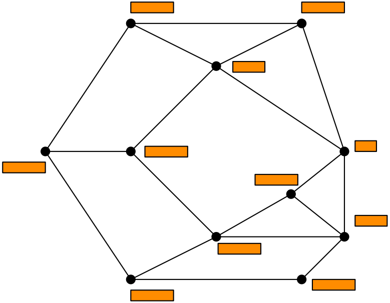
Bottom-Up Ordering [ADFGW 11]

- process vertices from least to most important
- temporarily remove the next vertex v from the graph
  - add shortcuts to preserve distances (if needed)
- recursively compute labels on smaller graph
- reinsert v and build label from its neighbors



Bottom-Up Ordering [ADFGW 11]

- process vertices from least to most important
- temporarily remove the next vertex v from the graph
  - add shortcuts to preserve distances (if needed)
- recursively compute labels on smaller graph
- reinsert v and build label from its neighbors
  - works because any path from v must go through a neighbor



Greedy Ordering [ADGW 12]

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]

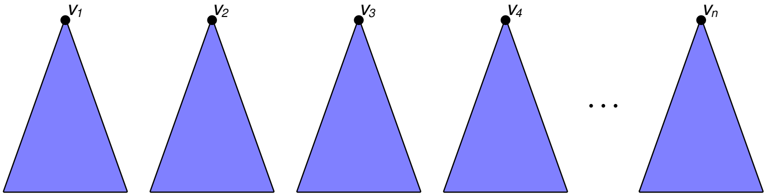
Greedy ordering (most to least important)

[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

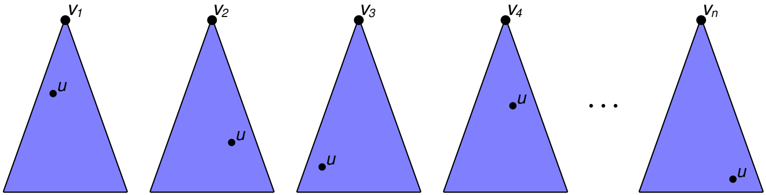
- next we describe data structures
- for simplicity assume that shortest paths are unique

Engineering Efficient Implementation



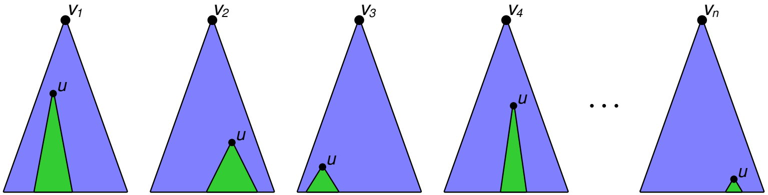
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i

Engineering Efficient Implementation



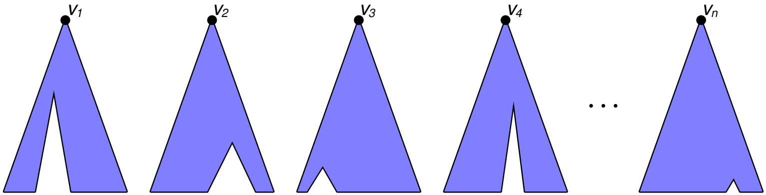
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u

Engineering Efficient Implementation



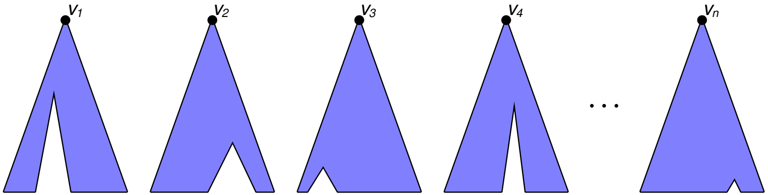
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order

Engineering Efficient Implementation



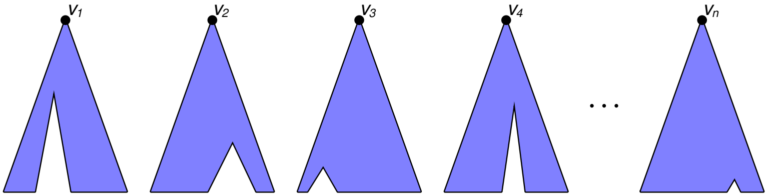
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts

Engineering Efficient Implementation



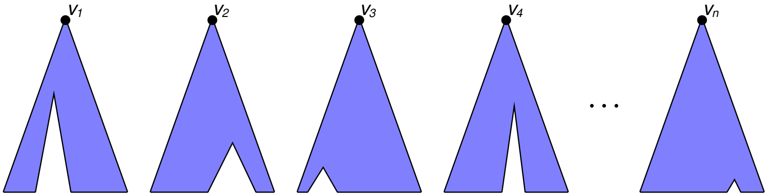
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts
- the trees represent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

Engineering Efficient Implementation



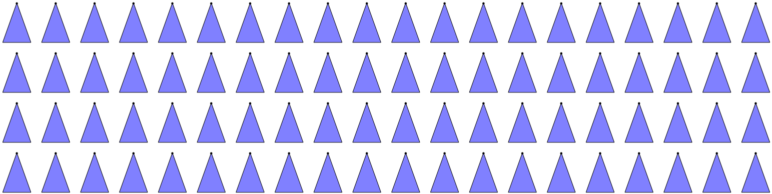
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts
- the trees represent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

Engineering Efficient Implementation



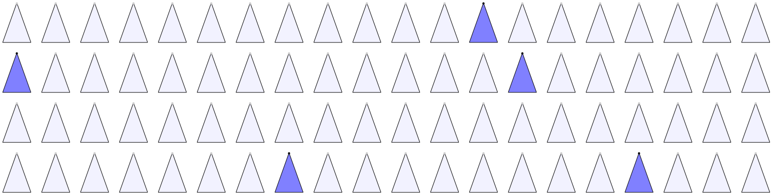
- build shortest path trees from each vertex
  - tree rooted at v_i represents all SPs from v_i
- invariant: # (descendants of u in T_i) $=$ # if SP from v_i hit by u
- add a vertex u with the most (total) decedents to the order
- delete subtrees rooted at u and update descendant counts
- the trees represent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

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



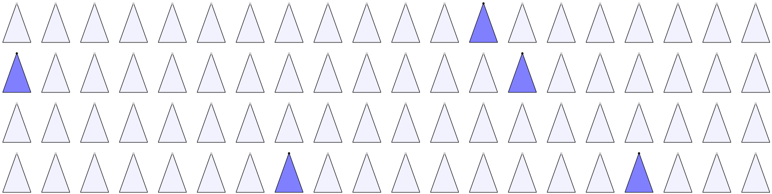
- maintain a sample of $\ll n$ trees (within tree node budget)

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



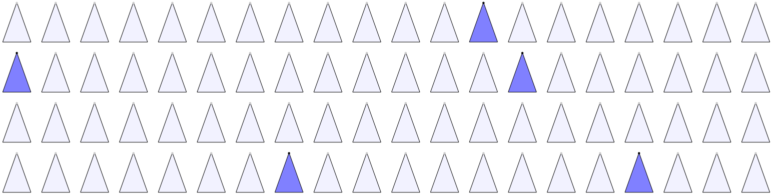
- maintain a sample of $\ll n$ trees (within tree node budget)

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



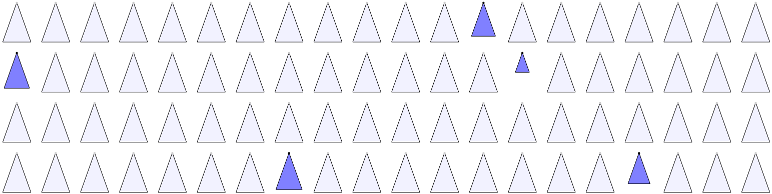
- maintain a sample of $\ll n$ trees (within tree node budget)
- use #descendants in the sample to estimate coverage of every u
  - sample is biased (e.g., vertices close to roots)
  - eliminate outliers

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



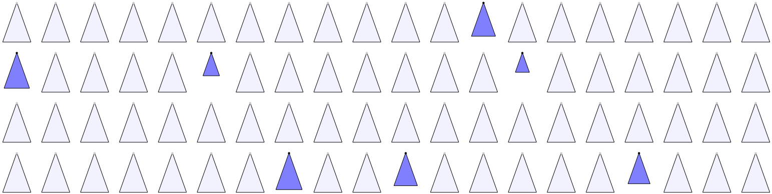
- maintain a sample of $\ll n$ trees (within tree node budget)
- use #descendants in the sample to estimate coverage of every u
  - sample is biased (e.g., vertices close to roots)
  - eliminate outliers
- add vertex u with the highest coverage estimate

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



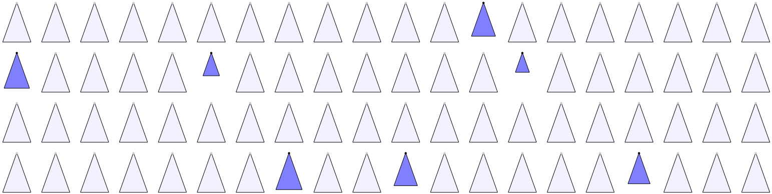
- maintain a sample of $\ll n$ trees (within tree node budget)
- use #descendants in the sample to estimate coverage of every u
  - sample is biased (e.g., vertices close to roots)
  - eliminate outliers
- add vertex u with the highest coverage estimate
- remove descendants of u in sampled trees

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)	$\lvert L\rvert/n$	prep (h:m)	$\lvert L\rvert/n$
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)

From Practice to Theory

Recent results on HHL [BGKSW 14]

- 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 14]

- 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 14]

- 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

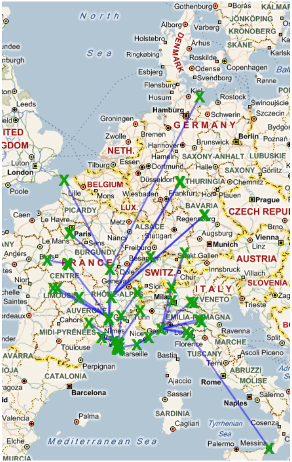
Great open problem

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

Concluding Remarks

Remarks

- label compression: trades query speed for memory footprint
- HL in external memory and data bases
- many technical details omitted to simplify presentation
- recent results on HL
  - theoretical work
  - experimental work
  - impact beyond mainstream algorithms community
- active area, open problems remain



Other Work

Recent

- social networks
- learning from GPS traces
- maximum flows, including vision applications
- minimum cost flows
- other routing and location-based services algorithms

Not so recent

- mechanism design
- combinatorial optimization
- distributed systems
  (e.g., intermemory)

Thank You!

Joint work with

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