Fast Geosocial Reachability Queries

Panagiotis Bouros

Daniel Kowalski

Theodoros Chondrogiannis

Geosocial networks

Networks that model both

- User social connections or interactions
- Geo-referenced actions

In academia, research on

- Modeling
- Indexing and query processing
- In relation to Recommender systems
- Analysis
  - Influence maximization, community detection etc.

Geosocial reachability

[M. Sarwat and Y. Sun, *Answering Location-Aware Graph Reachability Queries on GeoSocial Data*, IEEE ICDE 2017]

Geosocial network $G = (V, E, P)$

- Set of vertices V, set of directed edges E
- Set of 2D points P assigned to a subset of V -> spatial vertices

RangeReach query(G, v, R)

- Determine whether user v can socially relate to a spatial vertex inside region R
  - Have v's friends or friends of his/her friends visited a bar in Barcelona's city center?
- Hybrid query
  - Graph reachability problem GReach(G, v, u)
  - Spatial range query

Applications

- PoIs recommendation
- GeoAdvertizing

Existing solutions

Spatial-first approach

SpaReach

- RangeReach(G, v, R) queries in two steps
  - Determine spatial vertices u located inside R
  - GReach(G,v,u): if u is socially reachable from v

Spatial-first approach

SpaReach

- RangeReach(*G*, *v*, *R*) queries in two steps
  - Determine spatial vertices *u* located inside R
  - GReach(*G*,*v*,*u*): if *u* is socially reachable from *v*

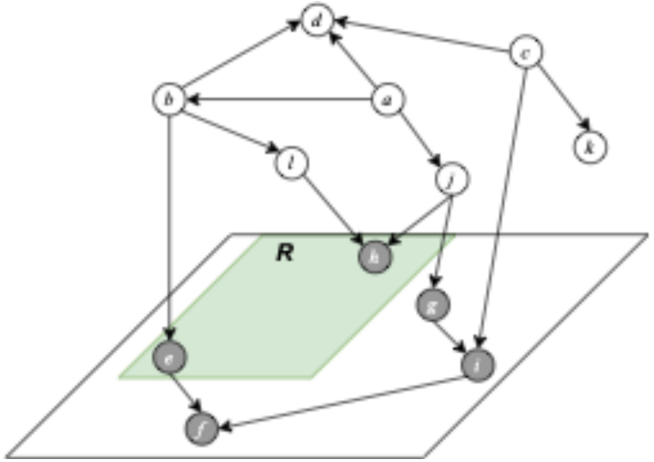
Challenges

- Spatial range query defined by *R*
  - Use spatial indexing
- Reachability queries GReach(*G*,*v*,*u*)
  - Use a labeling scheme

Spatial-first approach

SpaReach

- RangeReach(*G*, *v*, *R*) queries in two steps
  - Determine spatial vertices *u* located inside R
  - GReach(*G*,*v*,*u*): if *u* is socially reachable from *v*



Challenges

- Spatial range query defined by *R*
  - Use spatial indexing
- Reachability queries GReach(*G*,*v*,*u*)
  - Use a labeling scheme

RangeReach(*G*, *a*, *R*)

- Region *R* contains vertices *e*, *h*
- GReach(*G*, *a*, *e*) = TRUE

RangeReach(*G*, *c*, *R*)

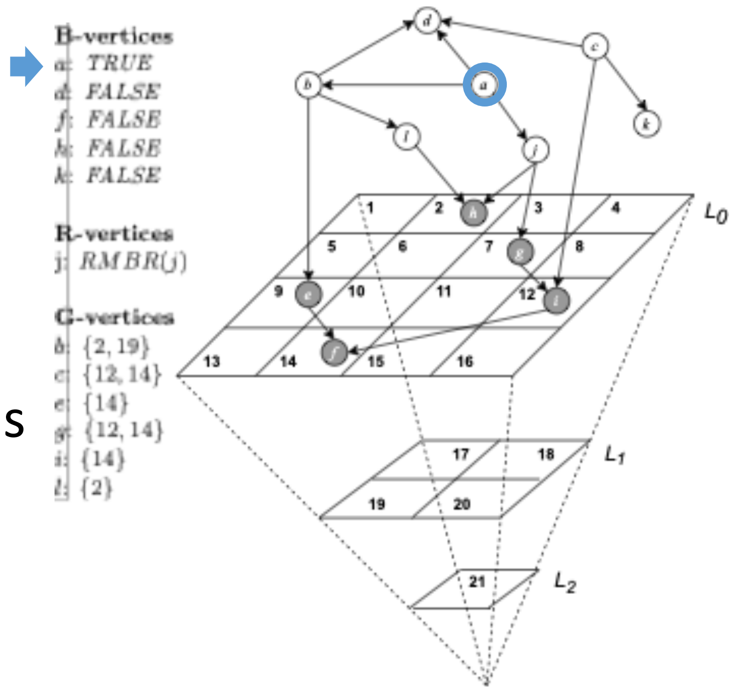
- Region *R* contains vertices *e*, *h*
- GReach(*G*, c, *e*) = FALSE
- GReach(*G*, *c*, *h*) = FALSE

GeoReach

[M. Sarwat and Y. Sun, *Answering Location-Aware Graph Reachability Queries on GeoSocial Data*, IEEE ICDE 2017]

SPA-Graph

- Hierarchical grid
- Precompute spatial and graph reachability
  - Three levels of detail
  - B-vertex: bit *GeoB*(*v*) if *v* can reach a spatial vertex
  - R-vertex: *RMBR*(*v*), *MBR* for all reachable spatial vertices
  - G-vertex: set of cells with all reachable spatial vertices

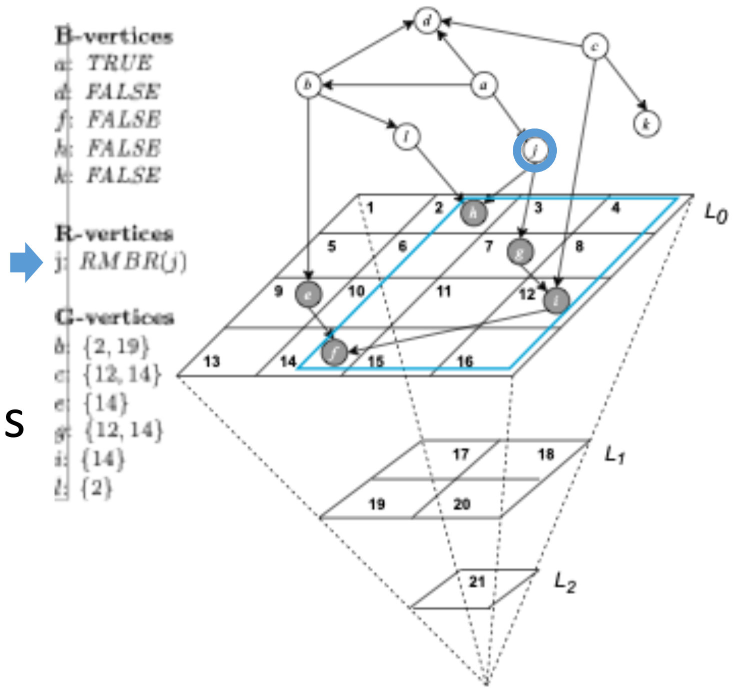


GeoReach

[M. Sarwat and Y. Sun, *Answering Location-Aware Graph Reachability Queries on GeoSocial Data*, IEEE ICDE 2017]

SPA-Graph

- Hierarchical grid
- Precompute spatial and graph reachability
  - Three levels of detail
  - B-vertex: bit *GeoB*(*v*) if *v* can reach a spatial vertex
  - R-vertex: *RMBR*(*v*), *MBR* for all reachable spatial vertices
  - G-vertex: set of cells with all reachable spatial vertices

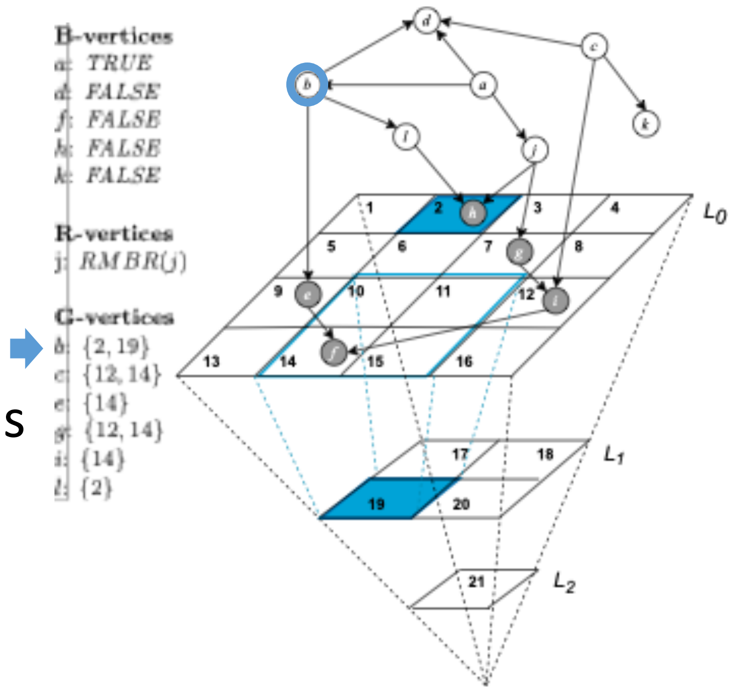


GeoReach

[M. Sarwat and Y. Sun, *Answering Location-Aware Graph Reachability Queries on GeoSocial Data*, IEEE ICDE 2017]

SPA-Graph

- Hierarchical grid
- Precompute spatial and graph reachability
  - Three levels of detail
  - B-vertex: bit *GeoB*(*v*) if *v* can reach a spatial vertex
  - R-vertex: *RMBR*(*v*), *MBR* for all reachable spatial vertices
  - G-vertex: set of cells with all reachable spatial vertices

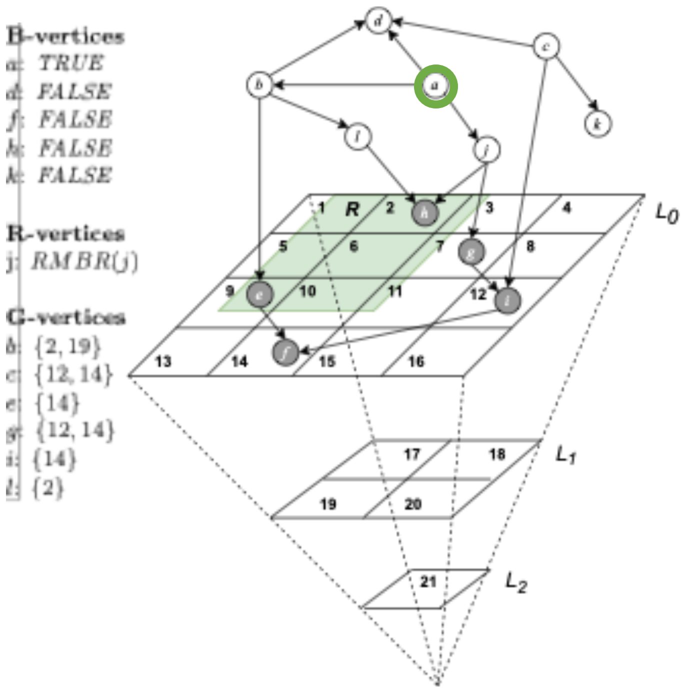


GeoReach

[M. Sarwat and Y. Sun, *Answering Location-Aware Graph Reachability Queries on GeoSocial Data*, IEEE ICDE 2017]

SPA-Graph

- Hierarchical grid
- Precompute spatial and graph reachability
  - Three levels of detail
  - B-vertex: bit *GeoB*(*v*) if *v* can reach a spatial vertex
  - R-vertex: *RMBR*(*v*), *MBR* for all reachable spatial vertices
  - G-vertex: set of cells with all reachable spatial vertices



RangeReach queries

- Traverse graph
  - Prune on the type of vertex and metadata
  - Early terminate

RangeReach(*G*, *a*, *R*)

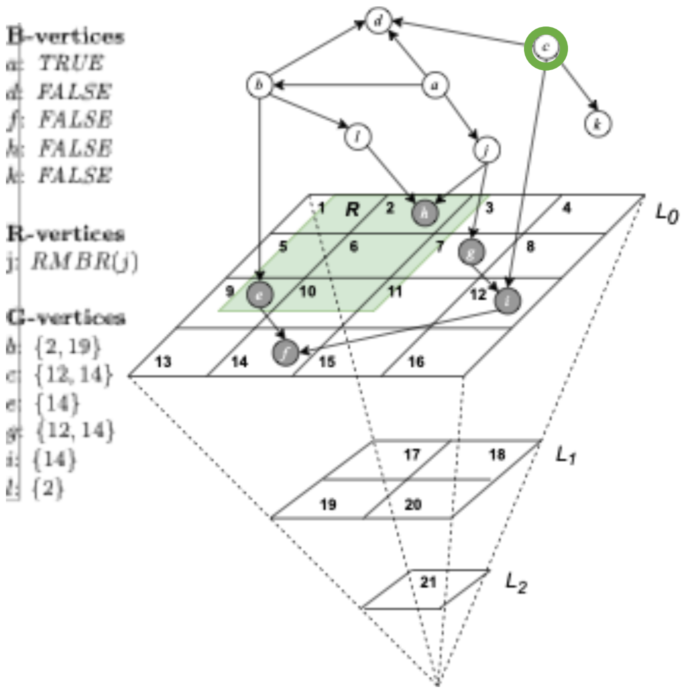
- B-vertex *a*, with *GeoB*(*a*) = TRUE
  - consider edges (*a*,*b*), (*a*,*d*), (*a*,*j*)
- G-vertex *b*:
  - Cell 2 inside query region *R*

GeoReach

[M. Sarwat and Y. Sun, *Answering Location-Aware Graph Reachability Queries on GeoSocial Data*, IEEE ICDE 2017]

SPA-Graph

- Hierarchical grid
- Precompute spatial and graph reachability
  - Three levels of detail
  - B-vertex: bit *GeoB*(*v*) if *v* can reach a spatial vertex
  - R-vertex: *RMBR*(*v*), *MBR* for all reachable spatial vertices
  - G-vertex: set of cells with all reachable spatial vertices



RangeReach queries

- Traverse graph
  - Prune on the type of vertex and metadata
  - Early terminate

RangeReach(*G*, *a*, *R*)

- B-vertex *a*, with *GeoB*(*a*) = TRUE
  - consider edges (*a*,*b*), (*a*,*d*), (*a*,*j*)
- G-vertex *b*:
  - Cell 2 inside query region *R*

RangeReach(*G*, *c*, *R*)

- G-vertex *c*:
  - Cells 12 and 14 don't overlap with query region *R*

Critique and motivation

SpaReach

- Performance for RangeReach queries with negative answer
  - Execute all possible reachability tests
- Priority to spatial query component
  - Large query region *R* means a lot of spatial vertices

GeoReach

- No query component prioritised
- Need to traverse the network graph
  - Large part for queries with negative answer
- No graph reachability labeling/indexing used

Methods on interval-based labeling

Interval-based labeling

Connectivity encoding scheme

- Compresses transitive closure
- Assign each vertex *v* a set of interval labels $\mathcal{L}(v)$
- Multiple variants
  - Some work only on trees

Efficient Management of Transitive Relationships in Large Data and Knowledge Bases, ACM SIGMOD 1989

- Compute a spanning tree and an initial set of labels
- Examine non-spanning tree edges to compute the final labels
- Vertex u is reachable from v iff
  - Exists a label [*x*,*y*] in $\mathcal{L}$(v) so that *x* <= *post(u)* <= *y*

Challenges

- Above scheme designed for knowledge hierarchies not arbitrary graphs

Interval-based labeling

Connectivity encoding scheme

- Compresses transitive closure
- Assign each vertex *v* a set of interval labels $\mathcal{L}(v)$
- Multiple variants
  - Some work only on trees

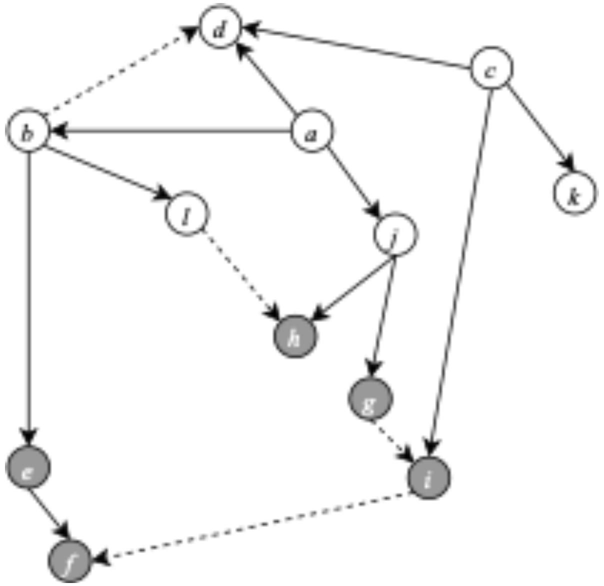
Efficient Management of Transitive Relationships in Large Data and Knowledge Bases, ACM SIGMOD 1989

- Compute a spanning tree and an initial set of labels
- Examine non-spanning tree edges to compute the final labels
- Vertex u is reachable from v iff
  - Exists a label $[x,y]$ in $\mathcal{L}(v)$ so that $x <= post(u) <= y$

Challenges

- Above scheme designed for knowledge hierarchies not arbitrary graphs

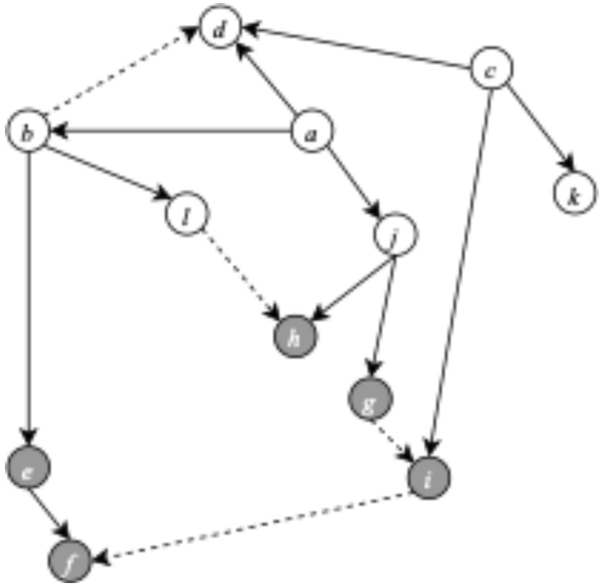
Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)			
b (4)			
c (12)			
d (5)			
e (2)			
f (1)			
g (6)			
h (7)			
i (10)			
j (8)			
k (11)			
l (3)			

- Compute spanning forest
- Compute *post*( ) numbers

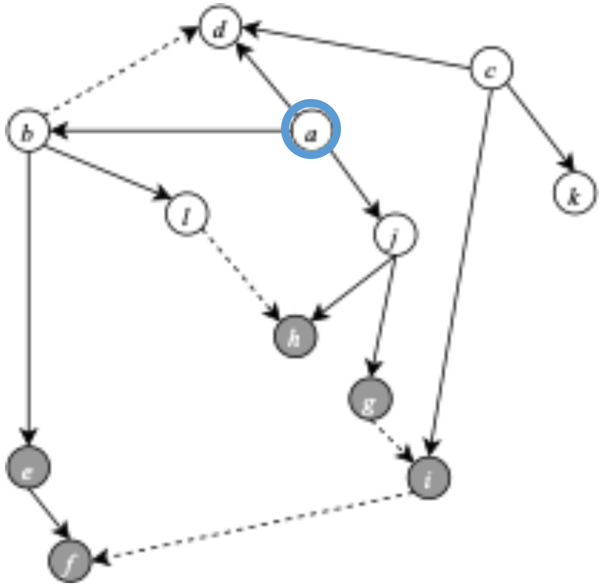
Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9]		
b (4)	[4,4]		
c (12)	[12,12]		
d (5)	[5,5]		
e (2)	[2,2]		
f (1)	[1,1]		
g (6)	[6,6]		
h (7)	[7,7]		
i (10)	[10,10]		
j (8)	[8,8]		
k (11)	[11,11]		
l (3)	[3,3]		

- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}

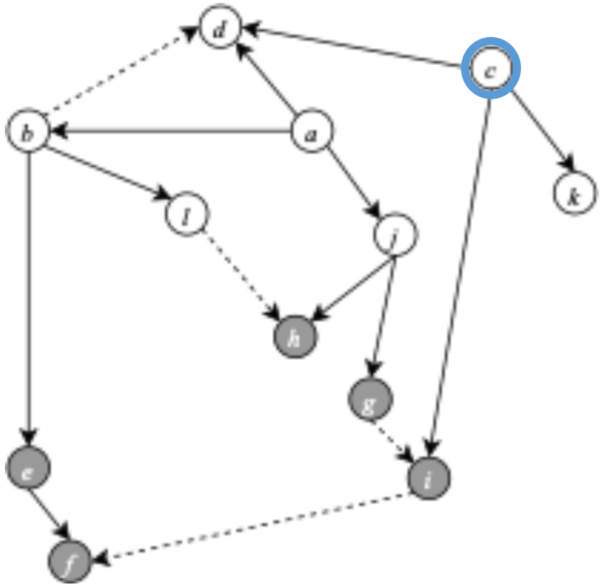
Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9] [4,4] [5,5] [8,8]		
b (4)	[4,4]		
c (12)	[12,12]		
d (5)	[5,5]		
e (2)	[2,2]		
f (1)	[1,1]		
g (6)	[6,6]		
h (7)	[7,7]		
i (10)	[10,10]		
j (8)	[8,8]		
k (11)	[11,11]		
l (3)	[3,3]		

- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}
- Visit *a*
  - Update $\mathcal{L}$(*a*), update Q

Labeling (geo)social networks

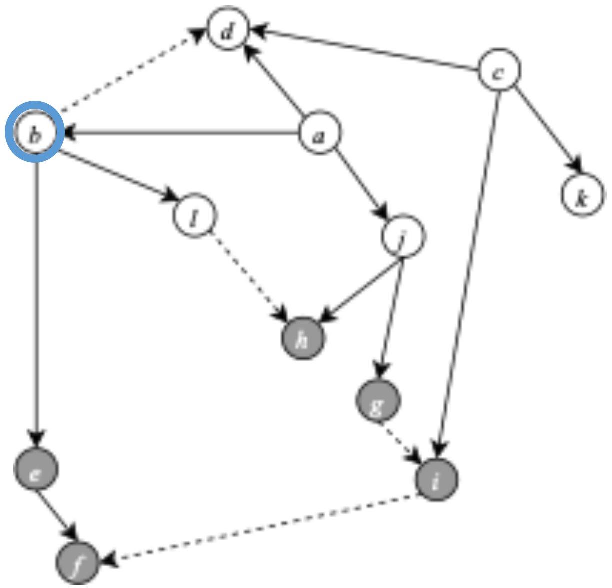


vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9] [4,4] [5,5] [8,8]		
b (4)	[4,4]		
c (12)	[12,12] [10,10] [11,11] [5,5]		
d (5)	[5,5]		
e (2)	[2,2]		
f (1)	[1,1]		
g (6)	[6,6]		
h (7)	[7,7]		
i (10)	[10,10]		
j (8)	[8,8]		
k (11)	[11,11]		
l (3)	[3,3]		

- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}
- Visit *a*
  - Update $\mathcal{L}$(*a*), update Q

- Q = {*c*, *b*, *j*, *d*}
- Visit *c*
  - Update $\mathcal{L}$(c), update Q

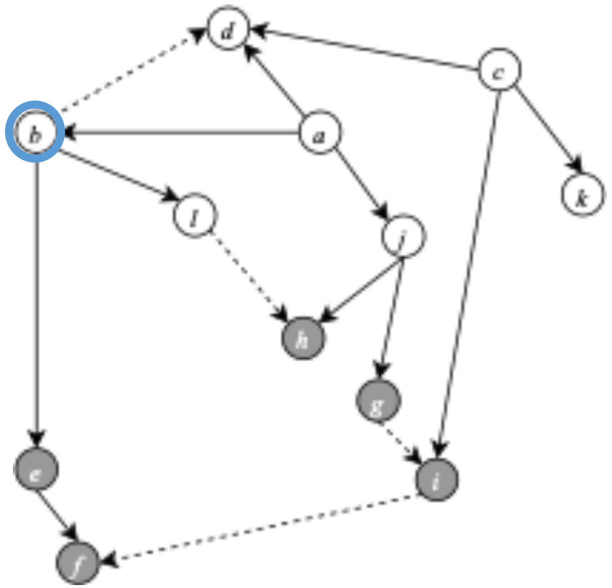
Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9] [4,4] [5,5] [8,8]		
b (4)	[4,4] [2,2] [3,3]		
c (12)	[12,12] [10,10] [11,11] [5,5]		
d (5)	[5,5]		
e (2)	[2,2]		
f (1)	[1,1]		
g (6)	[6,6]		
h (7)	[7,7]		
i (10)	[10,10]		
j (8)	[8,8]		
k (11)	[11,11]		
l (3)	[3,3]		

- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}
- Visit *a*
  - Update $\mathcal{L}$(*a*), update Q
- Q = {*c*, *b*, *j*, *d*}
- Visit *c*
  - Update $\mathcal{L}$(c), update Q
- Q = {*b*, *i*, *j*, *k*, *d*}
- Visit *b*
  - Update $\mathcal{L}$(*b*), $\mathcal{L}$(*a*), update Q

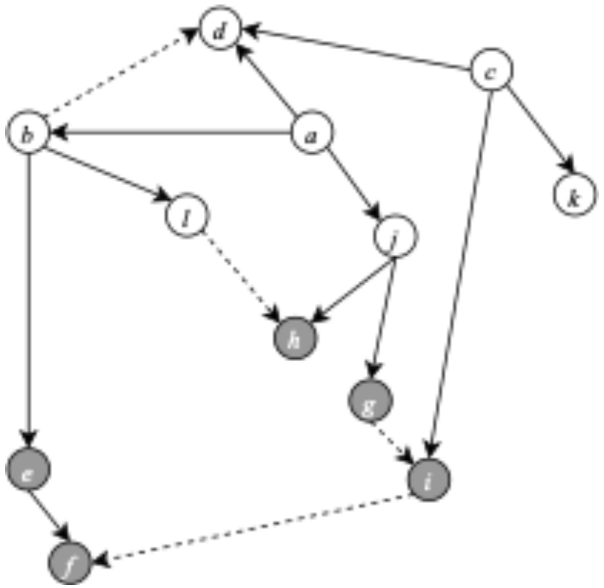
Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9] [4,4] [5,5] [8,8] [2,2] [3,3] [1,1]		
b (4)	[4,4] [2,2] [3,3]		
c (12)	[12,12] [10,10] [11,11] [5,5]		
d (5)	[5,5]		
e (2)	[2,2]		
f (1)	[1,1]		
g (6)	[6,6]		
h (7)	[7,7]		
i (10)	[10,10]		
j (8)	[8,8]		
k (11)	[11,11]		
l (3)	[3,3]		

- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}
- Visit *a*
  - Update $\mathcal{L}$(*a*), update Q
- Q = {*c*, *b*, *j*, *d*}
- Visit *c*
  - Update $\mathcal{L}$(c), update Q
- Q = {*b*, *i*, *j*, *k*, *d*}
- Visit *b*
  - Update $\mathcal{L}$(*b*), $\mathcal{L}$(*a*), update Q

Labeling (geo)social networks

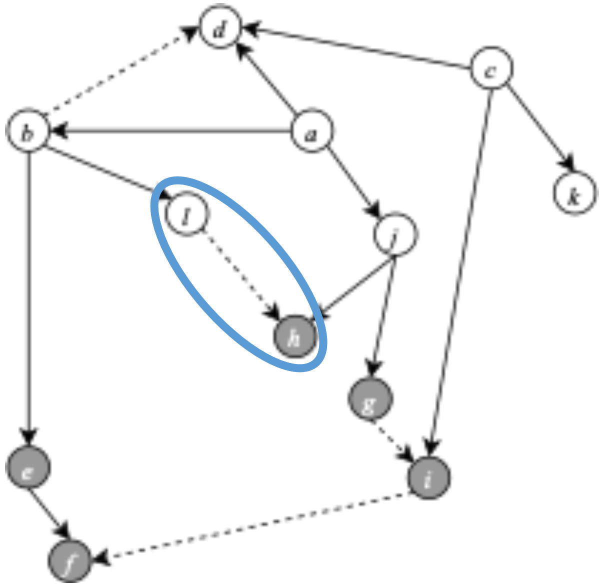


vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	**non-spanning edges**	**final**
a (9)	[9,9] [4,4] [5,5] [8,8] [2,2] [3,3] [1,1] [7,7] [6,6]		
b (4)	[4,4] [2,2] [3,3] [1,1]		
c (12)	[12,12] [10,10] [11,11] [5,5]		
d (5)	[5,5]		
e (2)	[2,2] [1,1]		
f (1)	[1,1]		
g (6)	[6,6]		
h (7)	[7,7]		
i (10)	[10,10]		
j (8)	[8,8] [7,7] [6,6]		
k (11)	[11,11]		
l (3)	[3,3]		

- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}
- Visit *a*
  - Update $\mathcal{L}$(*a*), update Q
- Q = {*c*, *b*, *j*, *d*}
- Visit *c*
  - Update $\mathcal{L}$(c), update Q
- Q = {*b*, *i*, *j*, *k*, *d*}
- Visit *b*
  - Update $\mathcal{L}$(*b*), $\mathcal{L}$(*a*), update Q

…

Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9] [4,4] [5,5] [8,8] [2,2] [3,3] [1,1] [7,7] [6,6]	[7,7]	
b (4)	[4,4] [2,2] [3,3] [1,1]	[7,7]	
c (12)	[12,12] [10,10] [11,11] [5,5]		
d (5)	[5,5]		
e (2)	[2,2] [1,1]		
f (1)	[1,1]		
g (6)	[6,6]		
h (7)	[7,7]		
i (10)	[10,10]		
j (8)	[8,8] [7,7] [6,6]		
k (11)	[11,11]		
l (3)	[3,3]	[7,7]	

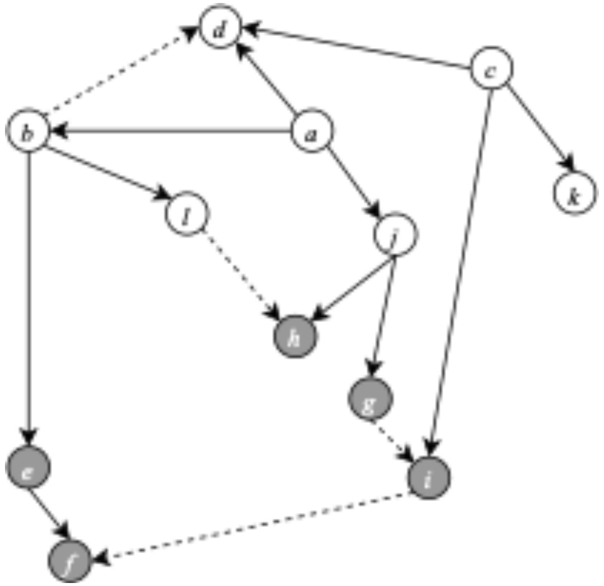
- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}
- Visit *a*
  - Update $\mathcal{L}$(*a*), update Q

- Q = {*c*, *b*, *j*, *d*}
- Visit *c*
  - Update $\mathcal{L}$(c), update Q
- Q = {*b*, *i*, *j*, *k*, *d*}
- Visit *b*
  - Update $\mathcal{L}$(*b*), $\mathcal{L}$(*a*), update Q

…

- Examine non-spanning edge (*l*, h)
  - Update $\mathcal{L}$(*l*), $\mathcal{L}$(*b*), $\mathcal{L}$(*a*)

Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$ spanning forest	non-spanning edges	final
a (9)	[9,9] [4,4] [5,5] [8,8] [2,2] [3,3] [1,1] [7,7] [6,6]	[7,7] [5,5] [1,1] [10,10]	
b (4)	[4,4] [2,2] [3,3] [1,1]	[7,7] [5,5]	
c (12)	[12,12] [10,10] [11,11] [5,5]	[1,1]	
d (5)	[5,5]		
e (2)	[2,2] [1,1]		
f (1)	[1,1]		
g (6)	[6,6]	[1,1] [10,10]	
h (7)	[7,7]		
i (10)	[10,10]	[1,1]	
j (8)	[8,8] [7,7] [6,6]	[1,1] [10,10]	
k (11)	[11,11]		
l (3)	[3,3]	[7,7]	

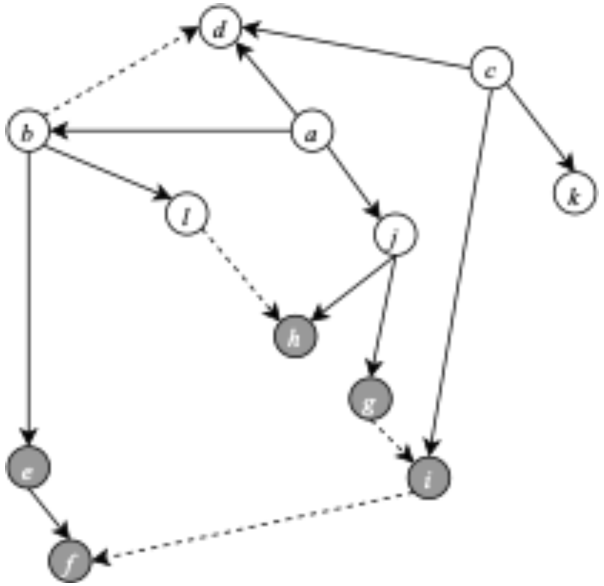
- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}
- Visit *a*
  - Update $\mathcal{L}$(*a*), update Q

- Q = {*c*, *b*, *j*, *d*}
- Visit *c*
  - Update $\mathcal{L}$(c), update Q
- Q = {*b*, *i*, *j*, *k*, *d*}
- Visit *b*
  - Update $\mathcal{L}$(*b*), $\mathcal{L}$(*a*), update Q

- Examine non-spanning edge (*l*, h)
  - Update $\mathcal{L}$(*l*), $\mathcal{L}$(*b*), $\mathcal{L}$(*a*)

…

Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9] [4,4] [5,5] [8,8] [2,2] [3,3] [1,1] [7,7] [6,6]	[7,7] [5,5] [1,1] [10,10]	[1,10]
b (4)	[4,4] [2,2] [3,3] [1,1]	[7,7] [5,5]	[1,5] [7,7]
c (12)	[12,12] [10,10] [11,11] [5,5]	[1,1]	[1,1] [5,5] [10,12]
d (5)	[5,5]		[5,5]
e (2)	[2,2] [1,1]		[1,2]
f (1)	[1,1]		[1,1]
g (6)	[6,6]	[1,1] [10,10]	[1,1] [6,6] [10,10]
h (7)	[7,7]		[7,7]
i (10)	[10,10]	[1,1]	[1,1] [10,10]
j (8)	[8,8] [7,7] [6,6]	[1,1] [10,10]	[1,1] [6,8] [10,10]
k (11)	[11,11]		[11,11]
l (3)	[3,3]	[7,7]	[3,3] [7,7]

- Compute spanning forest
- Compute *post*( ) numbers
- Prioritize vertices on in-degree
- Initialize priority Q = {*a*,*c*}
- Visit *a*
  - Update $\mathcal{L}$(*a*), update Q
- Q = {*c*, *b*, *j*, *d*}
- Visit *c*
  - Update $\mathcal{L}$(c), update Q
- Q = {*b*, *i*, *j*, *k*, *d*}
- Visit *b*
  - Update $\mathcal{L}$(*b*), $\mathcal{L}$(*a*), update Q

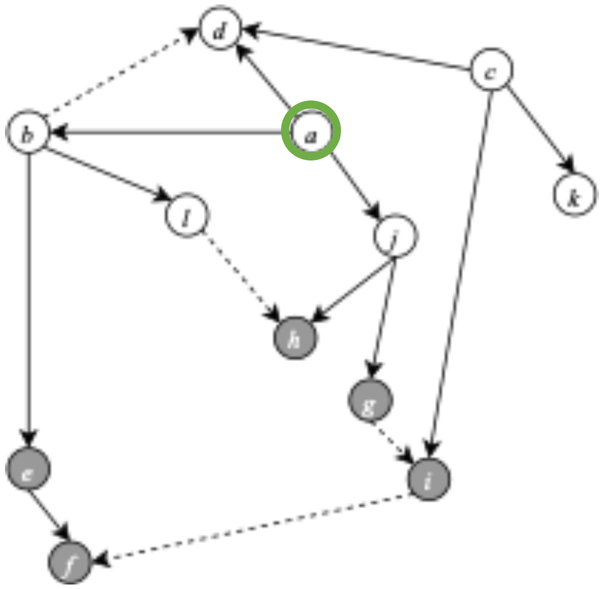
…

- Examine non-spanning edge (*l*, h)
  - Update $\mathcal{L}$(*l*), $\mathcal{L}$(*b*), $\mathcal{L}$(*a*)

…

- Unify and compress labels

Labeling (geo)social networks



vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	**non-spanning edges**	**final**
a (9)	[9,9] [4,4] [5,5] [8,8] [2,2] [3,3] [1,1] [7,7] [6,6]	[7,7] [5,5] [1,1] [10,10]	[1,10]
b (4)	[4,4] [2,2] [3,3] [1,1]	[7,7] [5,5]	[1,5] [7,7]
c (12)	[12,12] [10,10] [11,11] [5,5]	[1,1]	[1,1] [5,5] [10,12]
d (5)	[5,5]		[5,5]
e (2)	[2,2] [1,1]		[1,2]
f (1)	[1,1]		[1,1]
g (6)	[6,6]	[1,1] [10,10]	[1,1] [6,6] [10,10]
h (7)	[7,7]		[7,7]
i (10)	[10,10]	[1,1]	[1,1] [10,10]
j (8)	[8,8] [7,7] [6,6]	[1,1] [10,10]	[1,1] [6,8] [10,10]
k (11)	[11,11]		[11,11]
l (3)	[3,3]	[7,7]	[3,3] [7,7]

GReach(G,a,e)

- $post(e) = 2$
- $\mathcal{L}(a) = [1,10]$
- YES
  - $1 <= post(e) <= 10$

GReach(G,a,k)

- $post(k) = 11$
- $\mathcal{L}(a) = [1,10]$
- NO
  - $post(k) > 10$

Social-first approach

SocReach

- RangeReach(G, v, R) queries in two steps
  - $\mathcal{D}(v)$: all socially reachable vertices from v
  - Check if exists a spatial vertex in $\mathcal{D}(v)$ inside R

Social-first approach

SocReach

- RangeReach(G, v, R) queries in two steps
  - $\mathcal{D}(v)$: all socially reachable vertices from v
  - Check if exists a spatial vertex in $\mathcal{D}(v)$ inside R

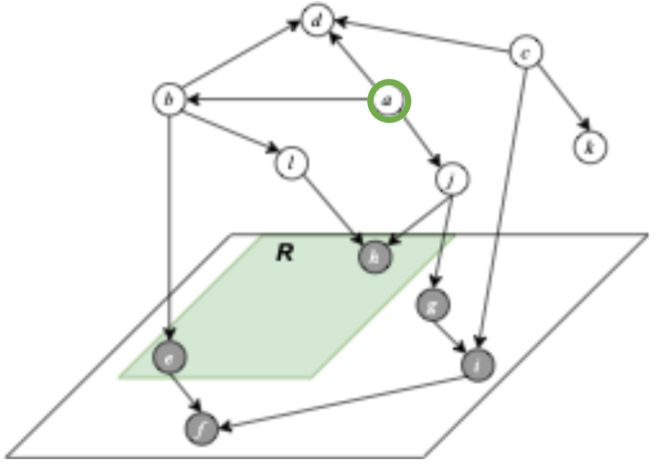
Challenge

- Compute $\mathcal{D}(v)$ fast
- Inefficient with most labeling schemes
  - Focus only on reachability test
- With interval-based labeling
  - Execute a typical range query for each label inside $\mathcal{L}(v)$ on vertex postorder

Social-first approach

SocReach

- RangeReach(G, v, R) queries in two steps
  - $\mathcal{D}(v)$: all socially reachable vertices from v
  - Check if exists a spatial vertex in $\mathcal{D}(v)$ inside R



RangeReach(G, a, R)

- $\mathcal{L}(a)$ = [1,10]
- $\mathcal{D}(a)$ = {f,e,l,b,d,g,h,j,a,i}

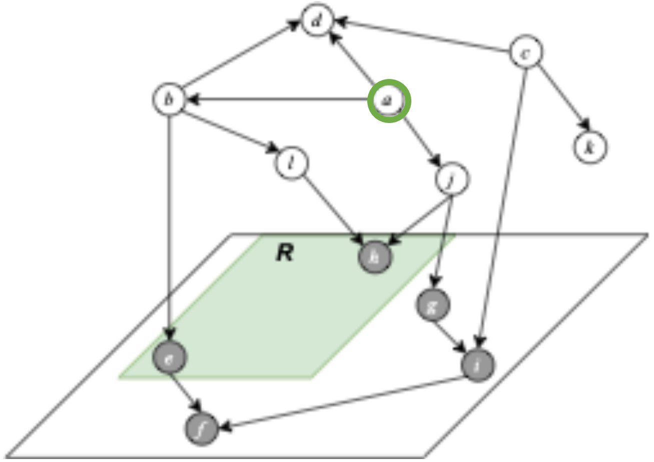
Challenge

- Compute $\mathcal{D}(v)$ fast
- Inefficient with most labeling schemes
  - Focus only on reachability test
- With interval-based labeling
  - Execute a typical range query for each label inside $\mathcal{L}(v)$ on vertex postorder

Social-first approach

SocReach

- RangeReach(G, v, R) queries in two steps
  - $\mathcal{D}$(v): all socially reachable vertices from v
  - Check if exists a spatial vertex in $\mathcal{D}$(v) inside R



RangeReach(G, a, R)

- $\mathcal{L}$(a) = [1,10]
- $\mathcal{D}$(a) = {f,e,l,b,d,g,h,j,a,i}

Challenge

- Compute $\mathcal{D}$(v) fast
- Inefficient with most labeling schemes
  - Focus only on reachability test
- With interval-based labeling
  - Execute a typical range query for each label inside $\mathcal{L}$(v) on vertex postorder

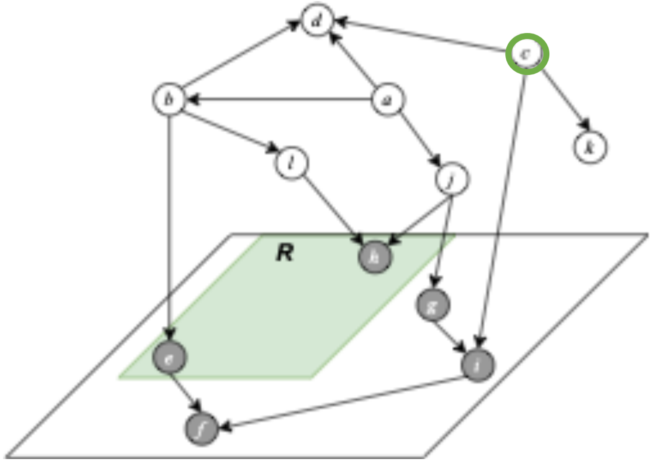
Social-first approach

SocReach

- RangeReach(G, v, R) queries in two steps
  - $\mathcal{D}$(v): all socially reachable vertices from v
  - Check if exists a spatial vertex in $\mathcal{D}$(v) inside R

Challenge

- Compute $\mathcal{D}$(v) fast
- Inefficient with most labeling schemes
  - Focus only on reachability test
- With interval-based labeling
  - Execute a typical range query for each label inside $\mathcal{L}$(v) on vertex postorder



RangeReach(G, a, R)

- $\mathcal{L}$(a) = [1,10]
- $\mathcal{D}$(a) = {f,e,l,b,d,g,h,j,a,i}

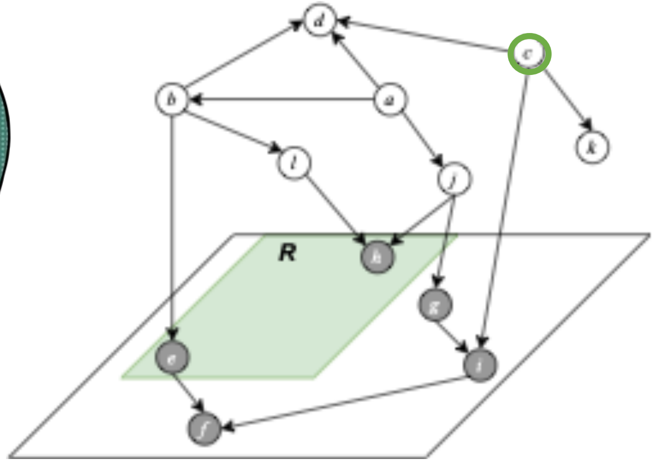
RangeReach(G, c, R)

- $\mathcal{L}$(c) = {[1,1], [5,5], [10,12]}
- $\mathcal{D}$(c) = {f,d,i,k,c}

Social-first approach

SocReach

- RangeReach(G, v, R) queries in two steps
  - $\mathcal{D}(v)$: all socially reachable vertices from v
  - Check if exists a spatial vertex in $\mathcal{D}(v)$ inside R



Challenge

- Compute $\mathcal{D}(v)$ fast
- Inefficient with most labeling schemes
  - Focus only on reachability test
- With interval-based labeling
  - Execute a typical range query for each label inside $\mathcal{L}(v)$ on vertex postorder

RangeReach(G, a, R)

- $\mathcal{L}(a)$ = [1,10]
- $\mathcal{D}(a)$ = {f,e,l,b,d,g,h,j,a,i}

RangeReach(G, c, R)

- $\mathcal{L}(c)$ = {[1,1], [5,5], [10,12]}
- $\mathcal{D}(c)$ = {f,d,i,k,c}

3DReach

Transformation

- Model the spatial vertices of the geosocial network in a 3D space
  - First two dimensions, the original spatial information
  - Third dimension, the domain of the vertex postorder numbers
- Two variants
  - Vertices as points using *post*( )
  - Vertices as sets of line segments using labels of the reverse interval labeling

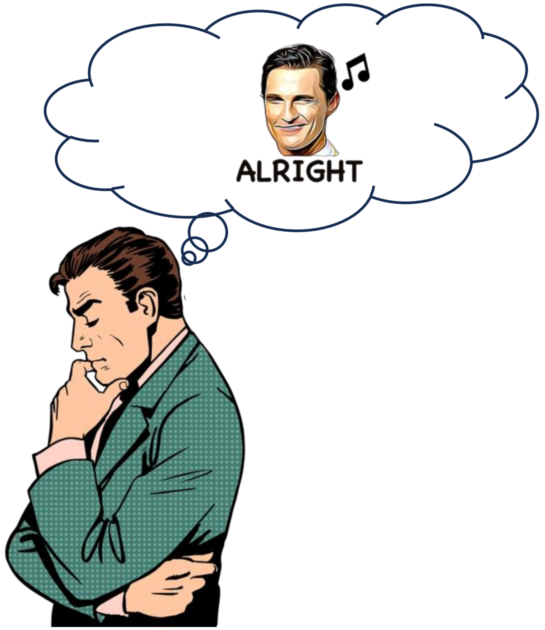
RangeReach queries

- Both query components treated equally
- Queries evaluated in one step, network graph not needed
- Rewritten in the new 3D space
- Two variants
  - A set of cuboids
  - A plane

3DReach

Transformation

- Model the spatial vertices of the geosocial network in a 3D space
  - First two dimensions, the original spatial information
  - Third dimension, the domain of the vertex postorder numbers
- Two variants
  - Vertices as points using *post*( )
  - Vertices as sets of line segments using labels of the reverse interval labeling

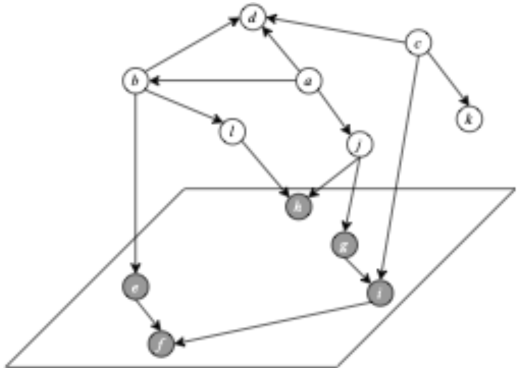


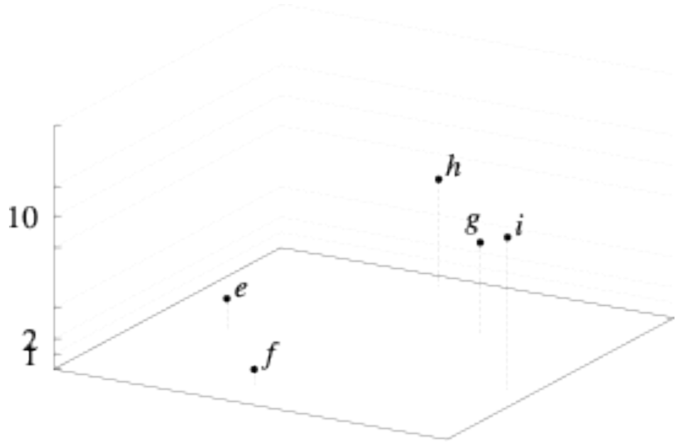
RangeReach queries

- Both query components treated equally
- Queries evaluated in one step, network graph not needed
- Rewritten in the new 3D space
- Two variants
  - A set of cuboids
  - A plane

3DReach (cont'd)

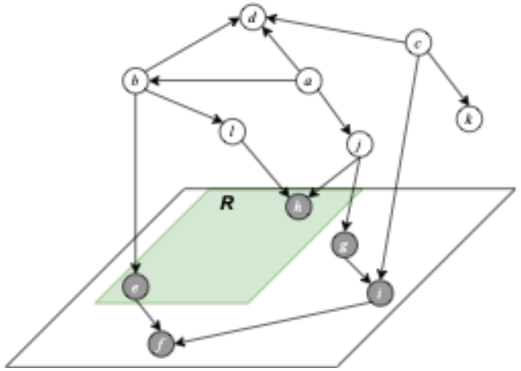
vertex v ($post(v)$)	final
a (9)	[1,10]
b (4)	[1,5] [7,7]
c (12)	[1,1] [5,5] [10,12]
d (5)	[5,5]
e (2)	[1,2]
f (1)	[1,1]
g (6)	[1,1] [6,6] [10,10]
h (7)	[7,7]
i (10)	[1,1] [10,10]
j (8)	[1,1] [6,8] [10,10]
k (11)	[11,11]
l (3)	[3,3] [7,7]

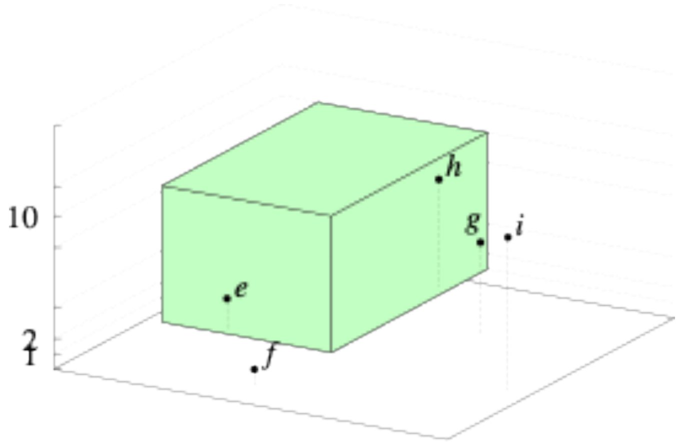




3DReach (cont'd)

vertex v ($post(v)$)	final
a (9)	[1,10]
b (4)	[1,5] [7,7]
c (12)	[1,1] [5,5] [10,12]
d (5)	[5,5]
e (2)	[1,2]
f (1)	[1,1]
g (6)	[1,1] [6,6] [10,10]
h (7)	[7,7]
i (10)	[1,1] [10,10]
j (8)	[1,1] [6,8] [10,10]
k (11)	[11,11]
l (3)	[3,3] [7,7]



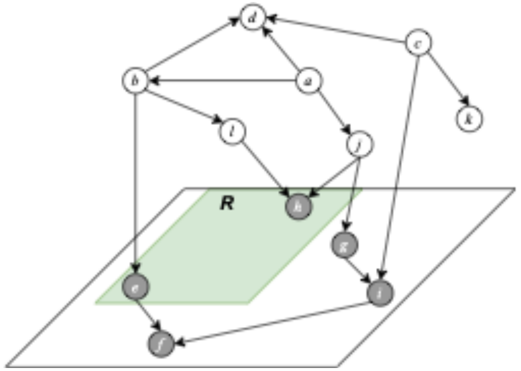


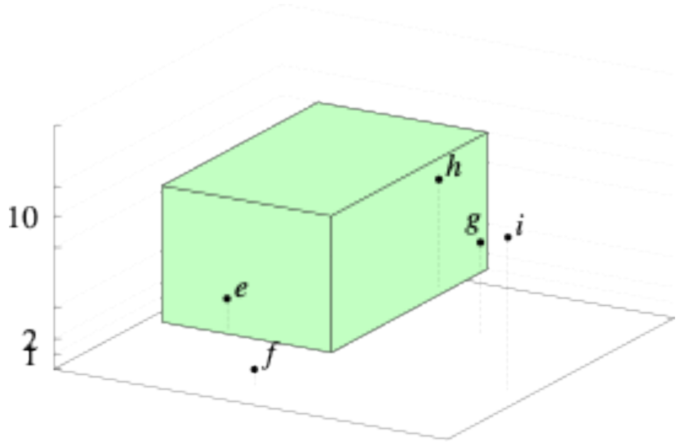
RangeReach(G, a, R)

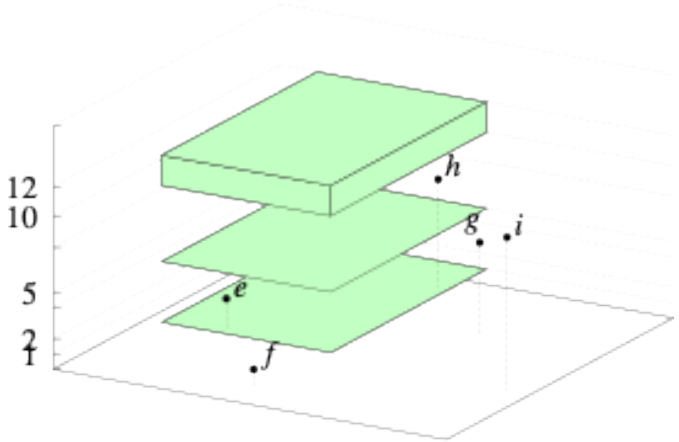
- Query cuboid
  - Region R as basis
  - Extent in the 3rd dimension based on $\mathcal{L}(a)$ = [1,10]
- Vertex e contained

3DReach (cont'd)

vertex v ($post(v)$)	final
a (9)	[1,10]
b (4)	[1,5] [7,7]
c (12)	[1,1] [5,5] [10,12]
d (5)	[5,5]
e (2)	[1,2]
f (1)	[1,1]
g (6)	[1,1] [6,6] [10,10]
h (7)	[7,7]
i (10)	[1,1] [10,10]
j (8)	[1,1] [6,8] [10,10]
k (11)	[11,11]
l (3)	[3,3] [7,7]







RangeReach(G, a, R)

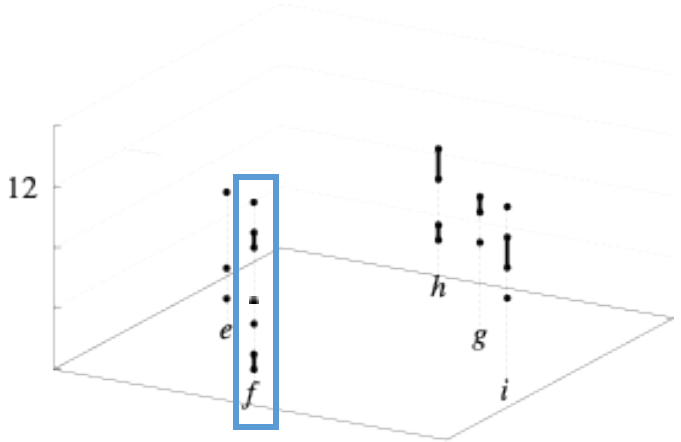
- Query cuboid
  - Region R as basis
  - Extent in the 3rd dimension based on $\mathcal{L}$ (a) = [1,10]
- Vertex e contained

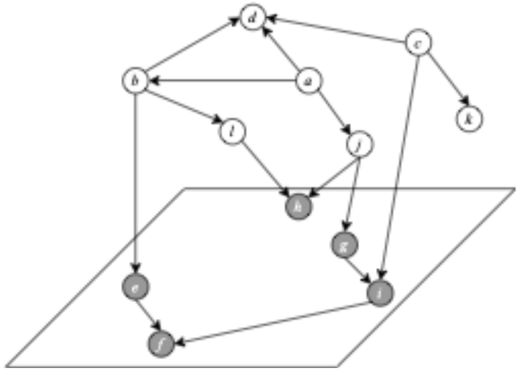
RangeReach(G, c, R)

- Query cuboids
  - Region R as basis
  - Extent in the 3rd dimension based on $\mathcal{L}$ (c) = {[1,1], [5,5], [10,12]}
- No vertex contained

3DReach-Rev

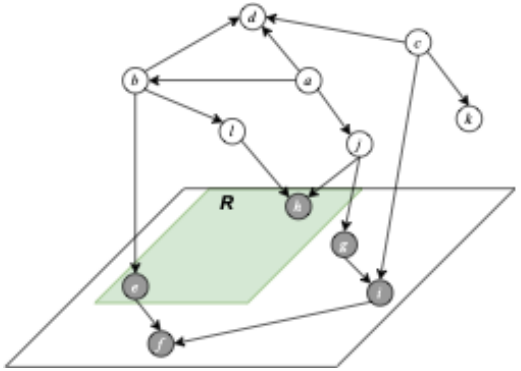
vertex v ($post(v)$)	final
a (9)	[9,9]
b (4)	[4,4] [9,9]
c (12)	[12,12]
d (5)	[4,5] [9,9] [12,12]
e (2)	[2,2] [4,4] [9,9]
f (1)	[1,2] [4,4] [6,6] [8,10] [12,12]
g (6)	[6,6] [8,9]
h (7)	[3,4] [7,9]
i (10)	[6,6] [8,10] [12,12]
j (8)	[8,9]
k (11)	[11,12]
l (3)	[3,4] [9,9]

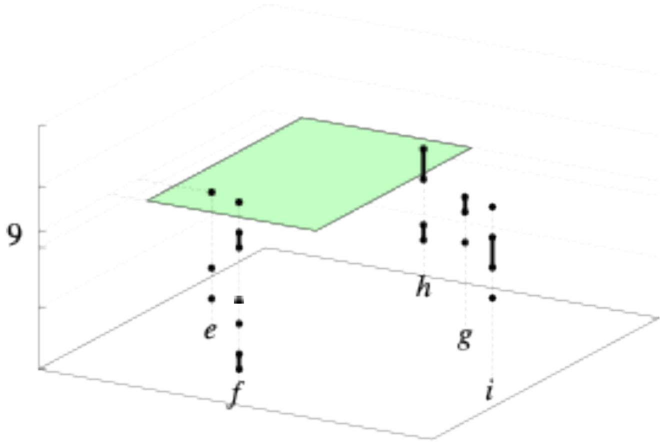


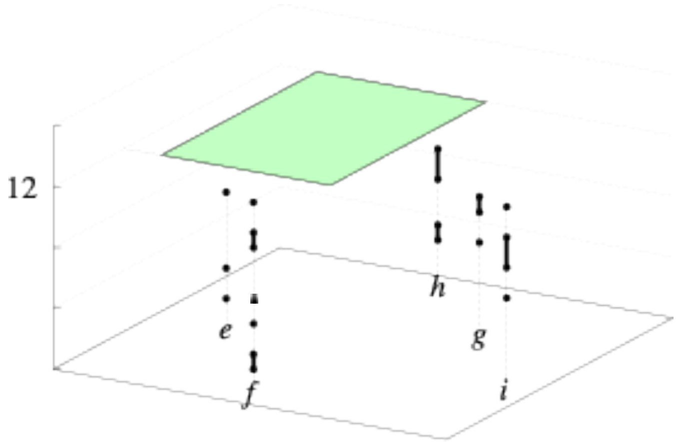


3DReach-Rev

vertex v ($post(v)$)	final
a (9)	[9,9]
b (4)	[4,4] [9,9]
c (12)	[12,12]
d (5)	[4,5] [9,9] [12,12]
e (2)	[2,2] [4,4] [9,9]
f (1)	[1,2] [4,4] [6,6] [8,10] [12,12]
g (6)	[6,6] [8,9]
h (7)	[3,4] [7,9]
i (10)	[6,6] [8,10] [12,12]
j (8)	[8,9]
k (11)	[11,12]
l (3)	[3,4] [9,9]







RangeReach(G, a, R)

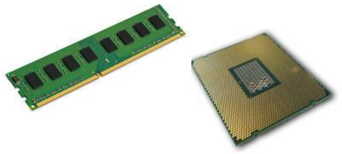
- Query plane
  - Region R as basis
  - Value in the 3rd dimension based on $post(a) = 9$
- Vertex e cut

RangeReach(G, c, R)

- Query plane
  - Region R as basis
  - Value in the 3rd dimension based on $post(c) = 12$
- No vertex cut

Experiments

Setup



Hardware

- Intel(R) Core i9(R) CPU @ 5.8 GHz with 64 GBs of RAM running Ubuntu Linux

Methods

- Spatial-first powered by an R-tree: SPAREACH-BFL with BFL scheme and SPAREACH-INT with interval-based labelling
- State-of-the-art: GEOREACH
- Our social first: SOCREACH
- Our 3DREACH and 3DREACH-REV powered by an 3DR-tree

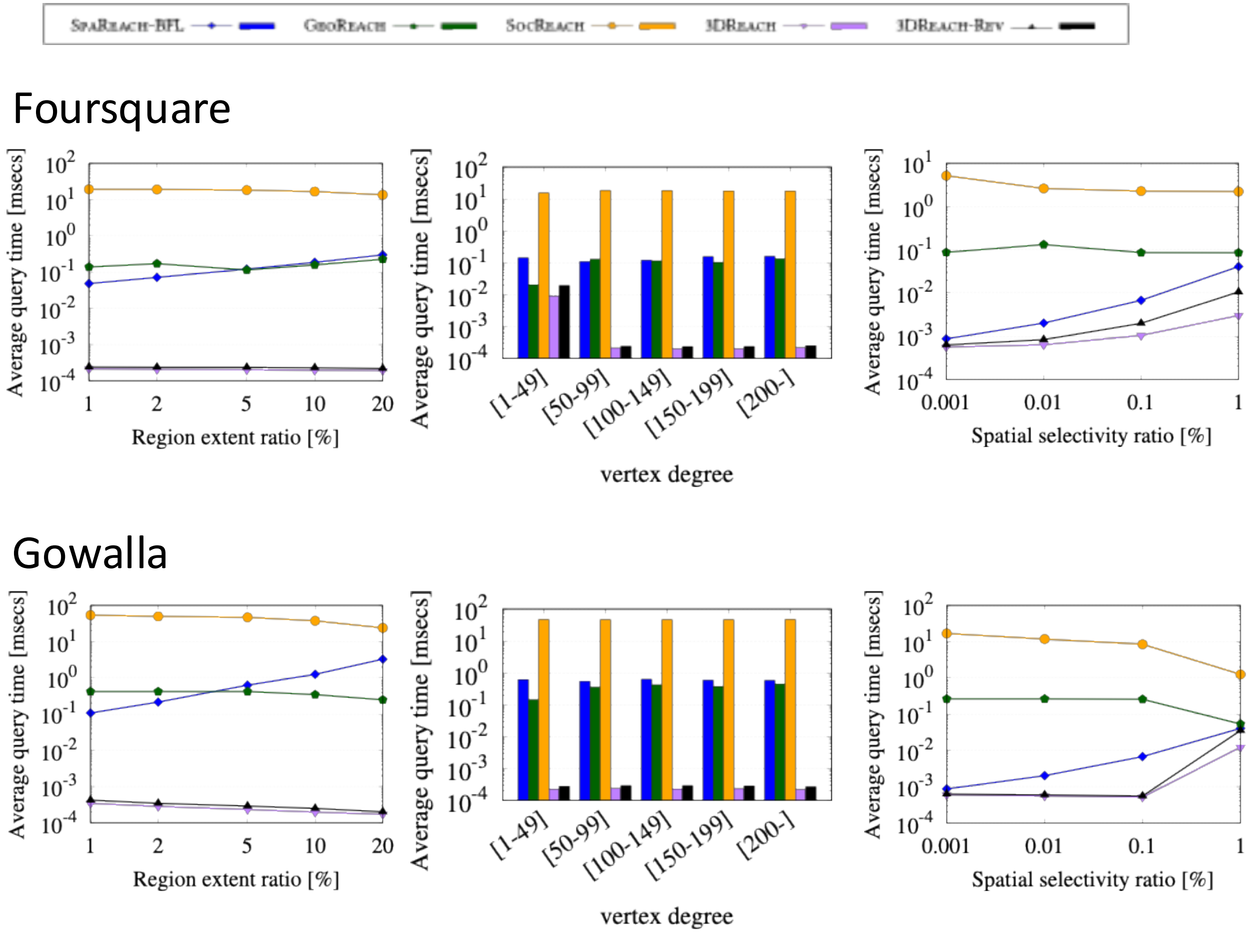
Datasets

- Non-spatial (social) vertices for users, spatial for venues
- Edges between social vertices for friend relationships and between social and spatial, for check-ins or ratings
- Containing one large SCC: Gowalla and WeePlaces
- Containing several SCCs: Foursquare and Yelp

Experiments

- Average runtime over 1000 queries
- Vary query region extent, query vertex degree, spatial selectivity

Methods comparison



3DReach methods are faster

- Rev typically slower

SocReach not competitive

- Prioritizing social (graph) query compenent does not work

To sum up…

Conclusions

- Studied computation of RangeReach queries in geosocial networks
  - A combination of graph reachability and spatial range queries
- Considered the application of interval-based labeling, proposing
  - A new construction process for arbitrary graphs
  - SocReach which prioritizes the graph query component
  - 3DReach which defines a 3D space to model the network and treats both query components equally

Future work

- Handle updates in the network how to share and save computations
- Apply our techniques to other queries

Thank you!

Questions

?

To download the source code and the datasets used, visit
https://github.com/pbour/rangereach