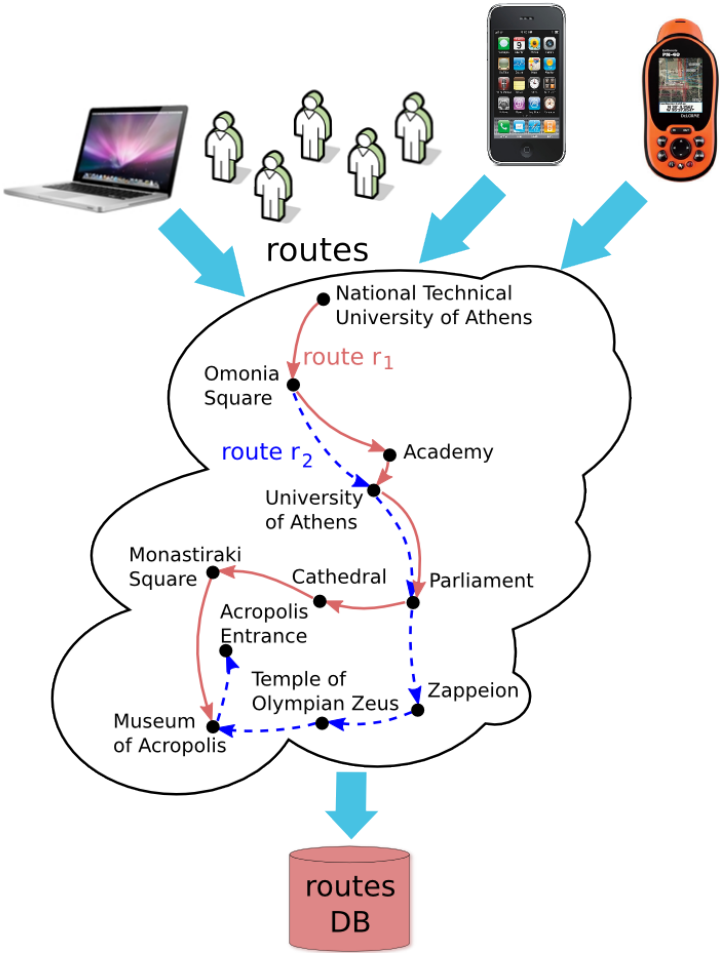
# Evaluating Path Queries over Route Collections

Panagiotis Bouros
NTUA, Greece
(supervised by Y. Vassiliou)

# Outline

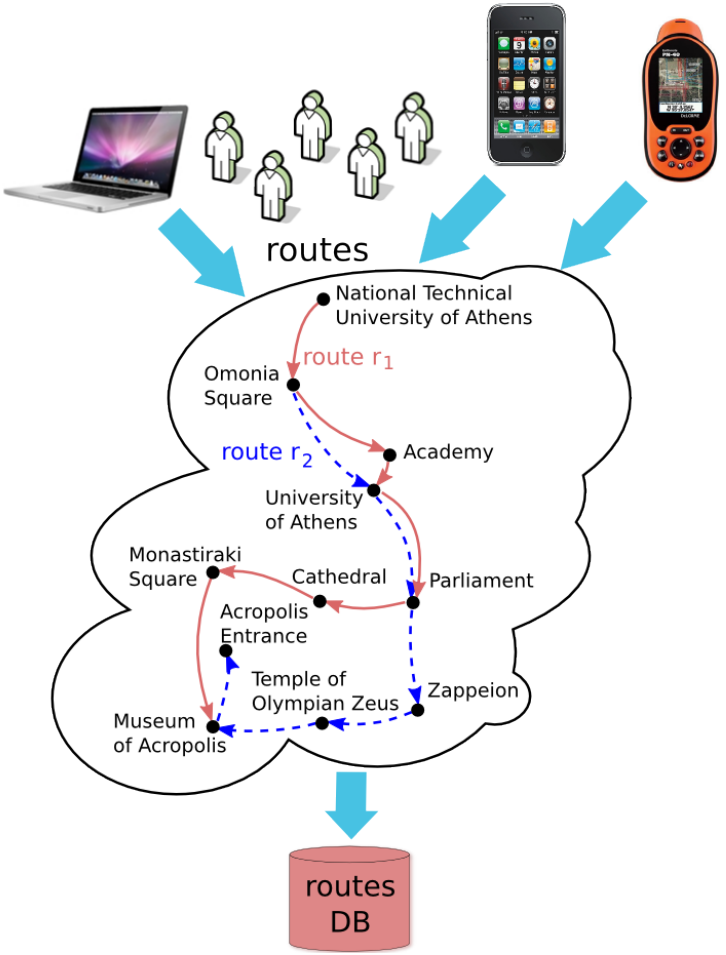
- Introduction
  - Route collections, queries & frequent updates
- Existing work
  - Graph-based solutions
- Our framework
  - PATH & FLSP queries
  - Indices, algorithms & handling updates
- Future work

# Examples of route collections



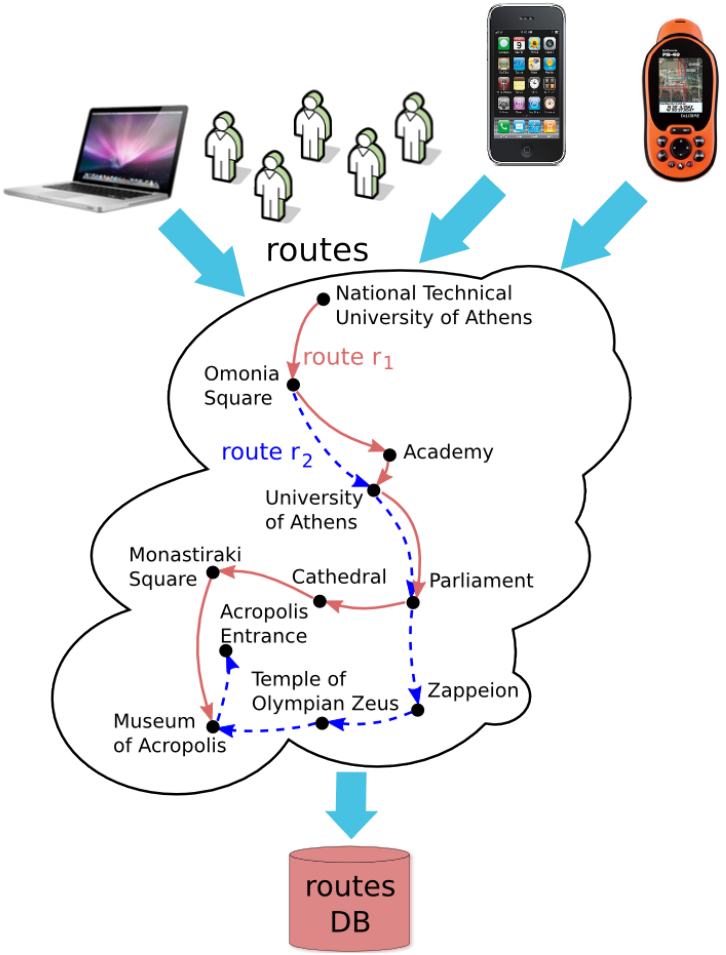

- People visiting Athens
  - Use e-devices to track their sightseeing
  - Create routes through interesting places
  - Upload/propose routes to Web sites like ShareMyRoutes.com

# Examples of route collections



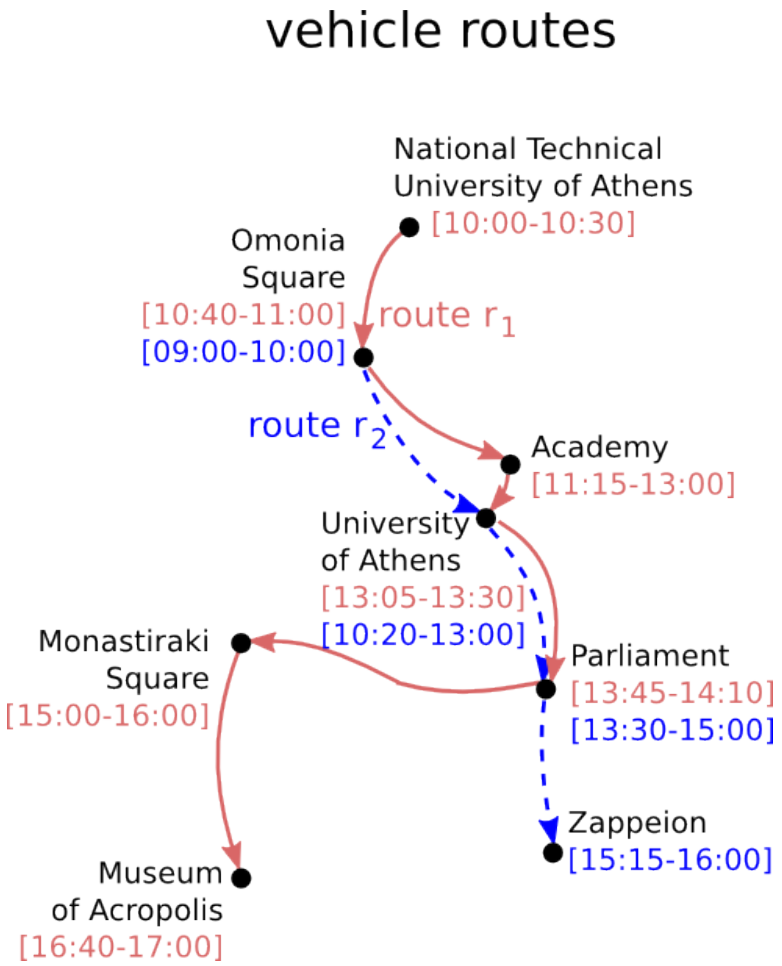

- People visiting Athens
  - Use e-devices to track their sightseeing
  - Create routes through interesting places
  - Upload/propose routes to Web sites like ShareMyRoutes.com
- Querying such route collection:
  - Find a sequence of interesting places from Omonia Square to Cathedral
  - Find a sequence of interesting places from Academy to Museum of Acropolis that passes through Zappeion

# Examples of route collections



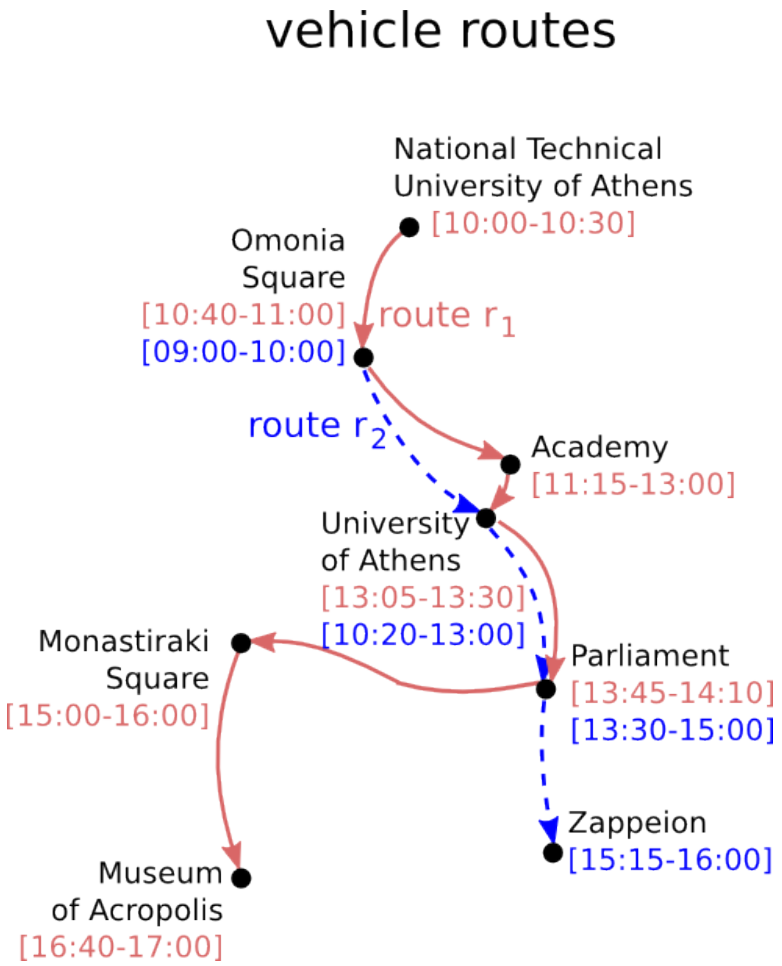

- People visiting Athens
  - Use e-devices to track their sightseeing
  - Create routes through interesting places
  - Upload/propose routes to Web sites like ShareMyRoutes.com
- Querying such route collection:
  - Find a sequence of interesting places from Omonia Square to Cathedral
  - Find a sequence of interesting places from Academy to Museum of Acropolis that passes through Zappeion
- Answers may involve places from more than one routes

# Examples of route collections



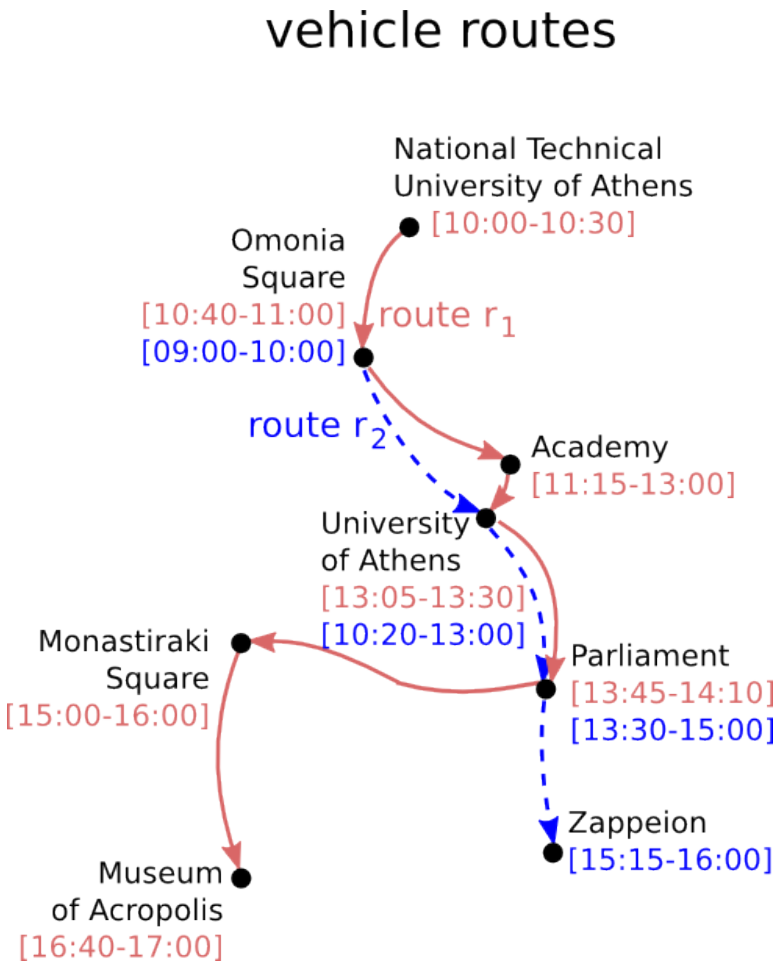

- Courier company offering same day pickup and delivery services
- Route collection created from previous day
  - Each route performed by a vehicle
  - Each point in routes is a point for picking-up, delivering or just waiting
  - Each point in a route has time interval I

# Examples of route collections



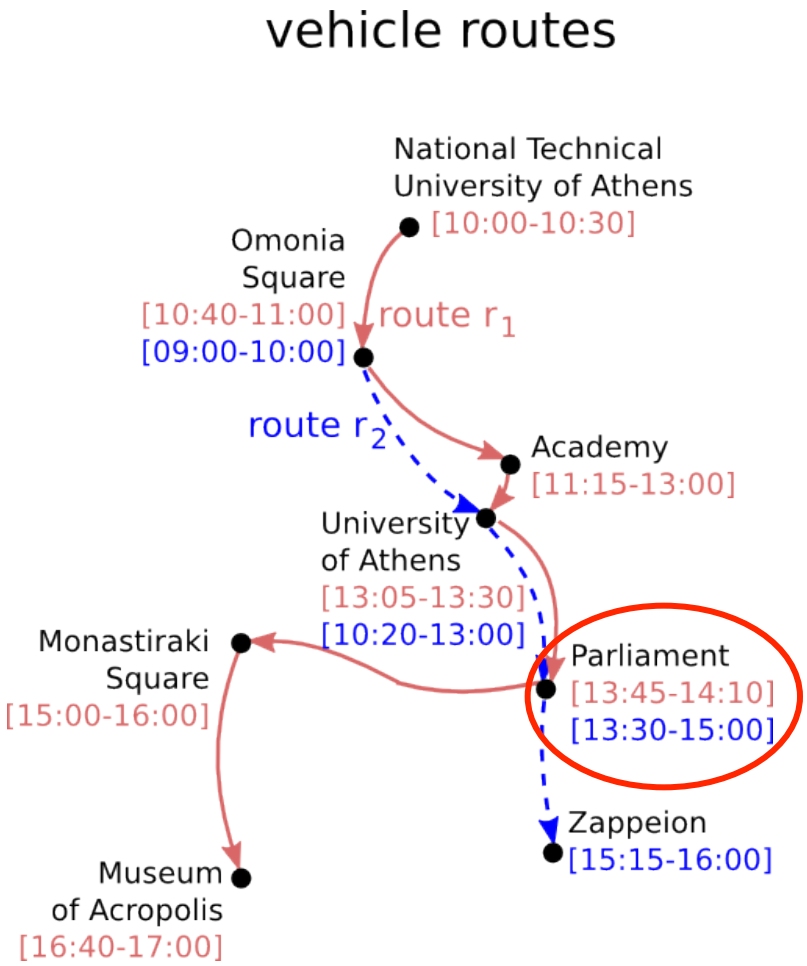

- Courier company offering same day pickup and delivery services
- Route collection created from previous day
  - Each route performed by a vehicle
  - Each point in routes is a point for picking-up, delivering or just waiting
  - Each point in a route has time interval I
- Querying such collection:
  - Ad-hoc customer requests
    - Pick-up parcel from Academy within $I_S$
    - Deliver parcel at Zappeion within $I_T$

# Examples of route collections




- Courier company offering same day pickup and delivery services
- Route collection created from previous day
  - Each route performed by a vehicle
  - Each point in routes is a point for picking-up, delivering or just waiting
  - Each point in a route has time interval I
- Querying such collection:
  - Ad-hoc customer requests
    - Pick-up parcel from Academy within $I_S$
    - Deliver parcel at Zappeion within $I_T$
- Serve ad-hoc request
  - Not add new route
  - Exploit one or more existing routes, pass parcel among vehicles

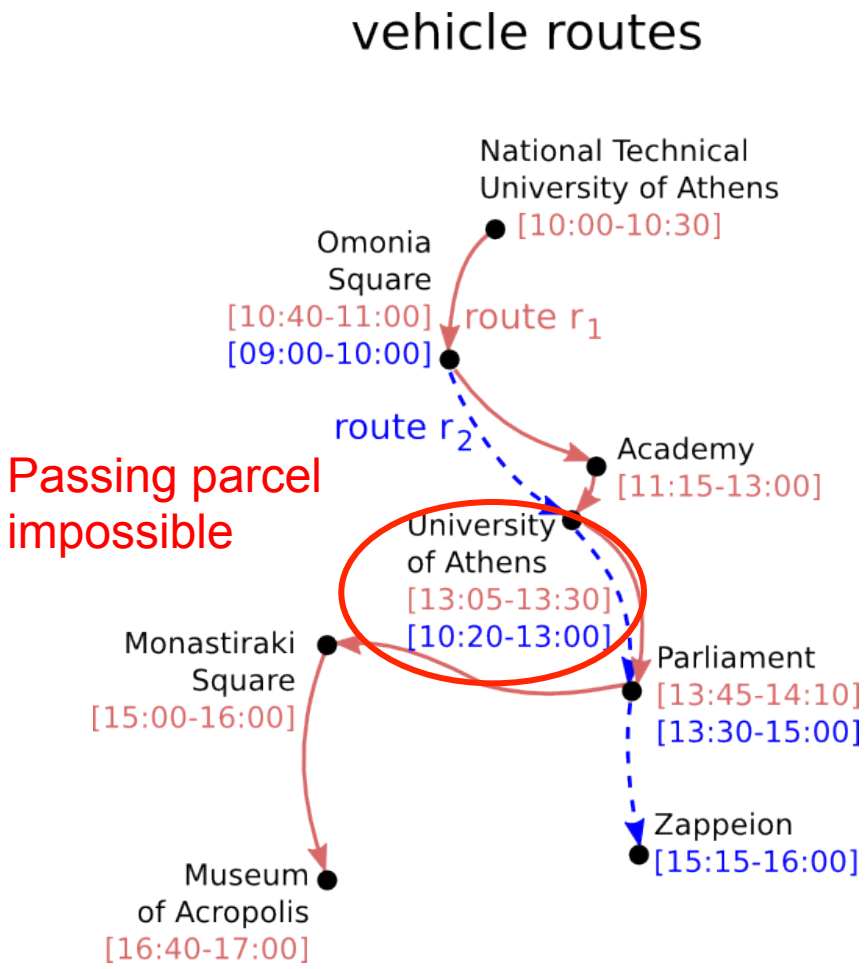
# Examples of route collections




- Courier company offering same day pickup and delivery services
- Route collection created from previous day
  - Each route performed by a vehicle
  - Each point in routes is a point for picking-up, delivering or just waiting
  - Each point in a route has time interval I
- Querying such collection:
  - Ad-hoc customer requests
    - Pick-up parcel from Academy within $I_S$
    - Deliver parcel at Zappeion within $I_T$
- Serve ad-hoc request
  - Not add new route
  - Exploit one or more existing routes, pass parcel among vehicles

Passing parcel possible

# Examples of route collections




- Courier company offering same day pickup and delivery services
- Route collection created from previous day
  - Each route performed by a vehicle
  - Each point in routes is a point for picking-up, delivering or just waiting
  - Each point in a route has time interval I
- Querying such collection:
  - Ad-hoc customer requests
    - Pick-up parcel from Academy within $I_S$
    - Deliver parcel at Zappeion within $I_T$
- Serve ad-hoc request
  - Not add new route
  - Exploit one or more existing routes, pass parcel among vehicles
- Related to dynamic pickup and delivery problem

# What's all about...

- Consider route collections
  - Very large, stored in secondary storage
  - Frequently updated with new routes
    - E.g. new routes are proposed or new vehicle routes are included

# What's all about...

- Consider route collections
  - Very large, stored in secondary storage
  - Frequently updated
    - E.g. new routes are proposed or new vehicle routes are included
- Evaluate queries that identify paths
  - Sequences of distinct nodes from one or more routes
  - In the latter we use links, i.e., shared nodes

# What's all about...

- Consider route collections
  - Very large, stored in secondary storage
  - Frequently updated
    - E.g. new routes are proposed or new vehicle routes are included
- Evaluate queries that identify paths
  - Sequences of distinct nodes from one or more routes
  - In the latter we use links, i.e., shared nodes
  - PATH(S,T)
    - Find a sequence of interesting places from Omonia Square to Cathedral

# What's all about…

- Consider route collections
  - Very large, stored in secondary storage
  - Frequently updated
    - E.g. new routes are proposed or new vehicle routes are included
- Evaluate queries that identify paths
  - Sequences of distinct nodes from one or more routes
  - In the latter we use links, i.e., shared nodes
  - PATH(S,T)
    - Find a sequence of interesting places from Omonia Square to Cathedral
  - FLSP($S,I_S,T,I_T$)
    - Moving cost m within routes
    - Changing cost c between routes via links
    - Find the sequence of points to pick-up parcel from Cathedral and deliver it to Zappeion that:
      - Abides with the temporal constraints imposed by time intervals
      - Minimizes primarily the changing cost and secondarily the moving cost

# Existing work

- Convert route collection to a graph
- Answer PATH and FLSP queries on the graph:
  - Direct evaluation
    - dfs for PATH
    - ucs, Dijkstra for FLSP
    - Low storage and maintenance cost
    - Slow query evaluation
  - Preprocessing
    - Precompute transitive closure, labeling & compression schems for PATH
    - Graph embedding + A* for FLSP
    - Fast query evaluation
    - High maintenance cost

# Our Framework

- Evaluate PATH and FLSP queries directly on route collections
- Our framework involves the following:
  - A search algorithm
    - Depth-first search for PATH queries
    - Uniform-Cost Search for FLSP queries
  - Indices on route collection for efficiently answering queries
    - Reduce iterations of the search algorithm
    - P-Index, H-Index, L-Index
  - Methods for handling frequent updates

# Indexing route collections

**Route collection**

$r_1$ (A,B,C)

$r_2$ (F,N,B,L)

$r_3$ (T,B,N,M)

# Indexing route collections

**Route collection**

$r_1$ (A,B,C)

$r_2$ (F,N,B,L)

$r_3$ (T,B,N,M)

**P-Index**

| node | *routes* list |
|---|---|
| A | <$r_1$:1> |
| B | <$r_1$:2>, <$r_2$:3>, <$r_3$:2> |
| C | <$r_1$:3> |
| F | <$r_2$:1> |
| L | <$r_2$:4> |
| M | <$r_3$:4> |
| N | <$r_2$:2>, <$r_3$:3> |
| T | <$r_3$:1> |

# Indexing route collections

**Route collection**

$r_1$ (A,B,C)

$r_2$ (F,N,B,L)

$r_3$ (T,B,N,M)

**P-Index**

| node | *routes* list |
|---|---|
| A | <$r_1$:1> |
| B | <$r_1$:2>, <$r_2$:3>, <$r_3$:2> |
| C | <$r_1$:3> |
| F | <$r_2$:1> |
| L | <$r_2$:4> |
| M | <$r_3$:4> |
| N | <$r_2$:2>, <$r_3$:3> |
| T | <$r_3$:1> |

# Indexing route collections

**Route collection**

| | |
|---|---|
| $r_1$ | (A,B,C) |
| $r_2$ | (F,N,B,L) |
| $r_3$ | (T,B,N,M) |

**H-Index**

| route | *edges* list |
|---|---|
| $r_1$ | <$r_2$,B:2:3>, <$r_3$,B:2:2> |
| $r_2$ | <$r_1$,B:3:2>, <$r_3$,B:3:2>, <$r_3$,N:3:2> |
| $r_3$ | <$r_1$,B:2:2>, <$r_2$,B:2:3>, <$r_2$,N:3:2> |

**P-Index**

| node | *routes* list |
|---|---|
| A | <$r_1$:1> |
| B | <$r_1$:2>, <$r_2$:3>, <$r_3$:2> |
| C | <$r_1$:3> |
| F | <$r_2$:1> |
| L | <$r_2$:4> |
| M | <$r_3$:4> |
| N | <$r_2$:2>, <$r_3$:3> |
| T | <$r_3$:1> |

# Indexing route collections

**Route collection**

| | |
|---|---|
| $r_1$ | (A,B,C) |
| $r_2$ | (F,N,B,L) |
| $r_3$ | (T,B,N,M) |

**H-Index**

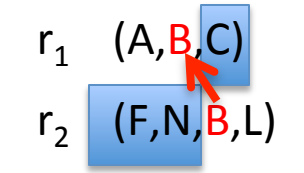
| route | *edges* list |
|---|---|
| $r_1$ | <$r_2$,B:2:3>, <$r_3$,B:2:2> |
| $r_2$ | <$r_1$,B:3:2>, <$r_3$,B:3:2>, <$r_3$,N:3:2> |
| $r_3$ | <$r_1$,B:2:2>, <$r_2$,B:2:3>, <$r_2$,N:3:2> |

**P-Index**

| node | *routes* list |
|---|---|
| A | <$r_1$:1> |
| B | <$r_1$:2>, <$r_2$:3>, <$r_3$:2> |
| C | <$r_1$:3> |
| F | <$r_2$:1> |
| L | <$r_2$:4> |
| M | <$r_3$:4> |
| N | <$r_2$:2>, <$r_3$:3> |
| T | <$r_3$:1> |

# Indexing route collections

**Route collection**




**H-Index**

| route | *edges* list |
|---|---|
| $r_1$ | $<r_2$,B:2:3>, $<r_3$,B:2:2> |
| $r_2$ | $<r_1$,B:3:2>, $<r_3$,B:3:2>, $<r_3$,N:3:2> |
| $r_3$ | $<r_1$,B:2:2>, $<r_2$,B:2:3>, $<r_2$,N:3:2> |

**P-Index**

| node | *routes* list |
|---|---|
| A | $<r_1$:1> |
| B | $<r_1$:2>, $<r_2$:3>, $<r_3$:2> |
| C | $<r_1$:3> |
| F | $<r_2$:1> |
| L | $<r_2$:4> |
| M | $<r_3$:4> |
| N | $<r_2$:2>, $<r_3$:3> |
| T | $<r_3$:1> |

# Indexing route collections

**Route collection**

| | |
|---|---|
| $r_1$ | (A,B,C) |
| $r_2$ | (F,N,B,L) |
| $r_3$ | (T,B,N,M) |

**H-Index**

| route | *edges* list |
|---|---|
| $r_1$ | <$r_2$,B:2:3>, <$r_3$,B:2:2> |
| $r_2$ | <$r_1$,B:3:2>, <$r_3$,B:3:2>, <$r_3$,N:3:2> |
| $r_3$ | <$r_1$,B:2:2>, <$r_2$,B:2:3>, <$r_2$,N:3:2> |

**P-Index**

| node | *routes* list |
|---|---|
| A | <$r_1$:1> |
| B | <$r_1$:2>, <$r_2$:3>, <$r_3$:2> |
| C | <$r_1$:3> |
| F | <$r_2$:1> |
| L | <$r_2$:4> |
| M | <$r_3$:4> |
| N | <$r_2$:2>, <$r_3$:3> |
| T | <$r_3$:1> |

# Indexing route collections

**Route collection**

| | |
|---|---|
| $r_1$ | (A,B,C) |
| $r_2$ | (F,N,B,L) |
| $r_3$ | (T,B,N,M) |

**H-Index**

| route | *edges* list |
|---|---|
| $r_1$ | <$r_2$,B:2:3>, <$r_3$,B:2:2> |
| $r_2$ | <$r_1$,B:3:2>, <$r_3$,B:3:2>, <$r_3$,N:3:2> |
| $r_3$ | <$r_1$,B:2:2>, <$r_2$,B:2:3>, <$r_2$,N:3:2> |

**P-Index**

| node | *routes* list |
|---|---|
| A | <$r_1$:1> |
| B | <$r_1$:2>, <$r_2$:3>, <$r_3$:2> |
| C | <$r_1$:3> |
| F | <$r_2$:1> |
| L | <$r_2$:4> |
| M | <$r_3$:4> |
| N | <$r_2$:2>, <$r_3$:3> |
| T | <$r_3$:1> |

# Indexing route collections

**Route collection**

| | |
|---|---|
| $r_1$ | (A,B,C) |
| $r_2$ | (F,N,B,L) |
| $r_3$ | (T,B,N,M) |

**H-Index**

| route | *edges* list |
|---|---|
| $r_1$ | <$r_2$,B:2:3>, <$r_3$,B:2:2> |
| $r_2$ | <$r_1$,B:3:2>, <$r_3$,B:3:2>, <$r_3$,N:3:2> |
| $r_3$ | <$r_1$,B:2:2>, <$r_2$,B:2:3>, <$r_2$,N:3:2> |

**P-Index**

| node | *routes* list |
|---|---|
| A | <$r_1$:1> |
| B | <$r_1$:2>, <$r_2$:3>, <$r_3$:2> |
| C | <$r_1$:3> |
| F | <$r_2$:1> |
| L | <$r_2$:4> |
| M | <$r_3$:4> |
| N | <$r_2$:2>, <$r_3$:3> |
| T | <$r_3$:1> |

**L-Index**

| route | *links* list |
|---|---|
| $r_1$ | <B:2> |
| $r_2$ | <B:3>, <N:2> |
| $r_3$ | <B:2>, <N:3> |

# Indexing route collections

**Route collection**

| | |
|---|---|
| $r_1$ | (A,B,C) |
| $r_2$ | (F,N,B,L) |
| $r_3$ | (T,B,N,M) |

**H-Index**

| route | *edges* list |
|---|---|
| $r_1$ | <$r_2$,B:2:3>, <$r_3$,B:2:2> |
| $r_2$ | <$r_1$,B:3:2>, <$r_3$,B:3:2>, <$r_3$,N:3:2> |
| $r_3$ | <$r_1$,B:2:2>, <$r_2$,B:2:3>, <$r_2$,N:3:2> |

**P-Index**

| node | *routes* list |
|---|---|
| A | <$r_1$:1> |
| B | <$r_1$:2>, <$r_2$:3>, <$r_3$:2> |
| C | <$r_1$:3> |
| F | <$r_2$:1> |
| L | <$r_2$:4> |
| M | <$r_3$:4> |
| N | <$r_2$:2>, <$r_3$:3> |
| T | <$r_3$:1> |

**L-Index**

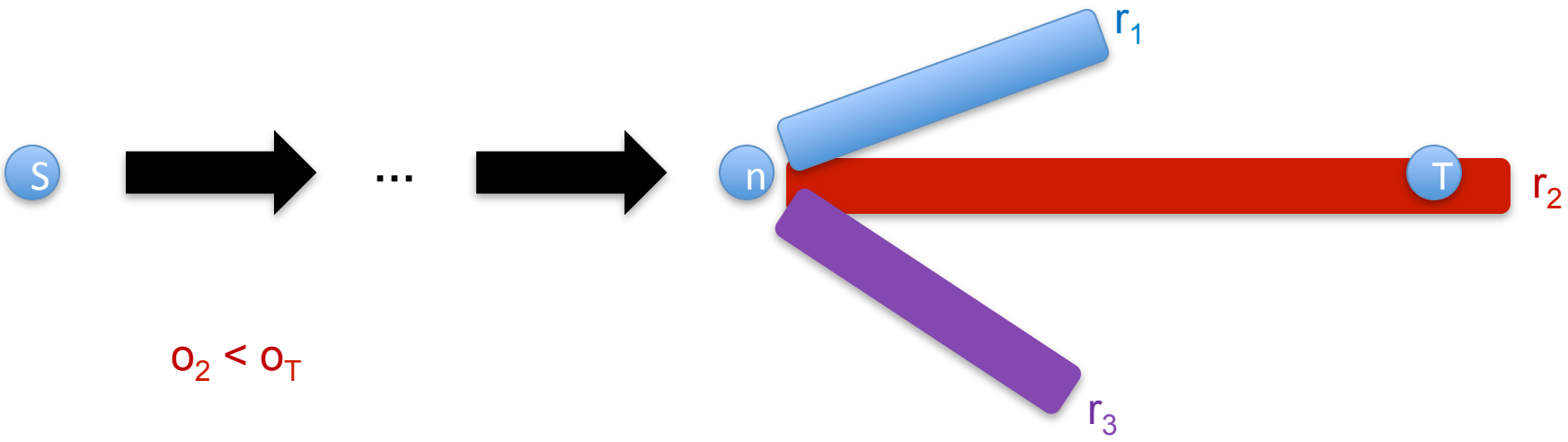
| route | *links* list |
|---|---|
| $r_1$ | <B:2> |
| $r_2$ | <B:3>, <N:2> |
| $r_3$ | <B:2>, <N:3> |

# Evaluating PATH queries

- Basic idea
  - Traverse nodes similar to depth-first search
  - Exploit indices on route collection to terminate search
    - P-Index => method pfsP
    - H-Index => method pfsH
    - L-Index => method pfsL
  - At each iteration:
    - Pop current node n
    - Check termination condition
    - Access routes containing n, *routes*[n] from P-Index
      - For each route, push nodes after n in search stack

# Evaluating PATH queries cont.

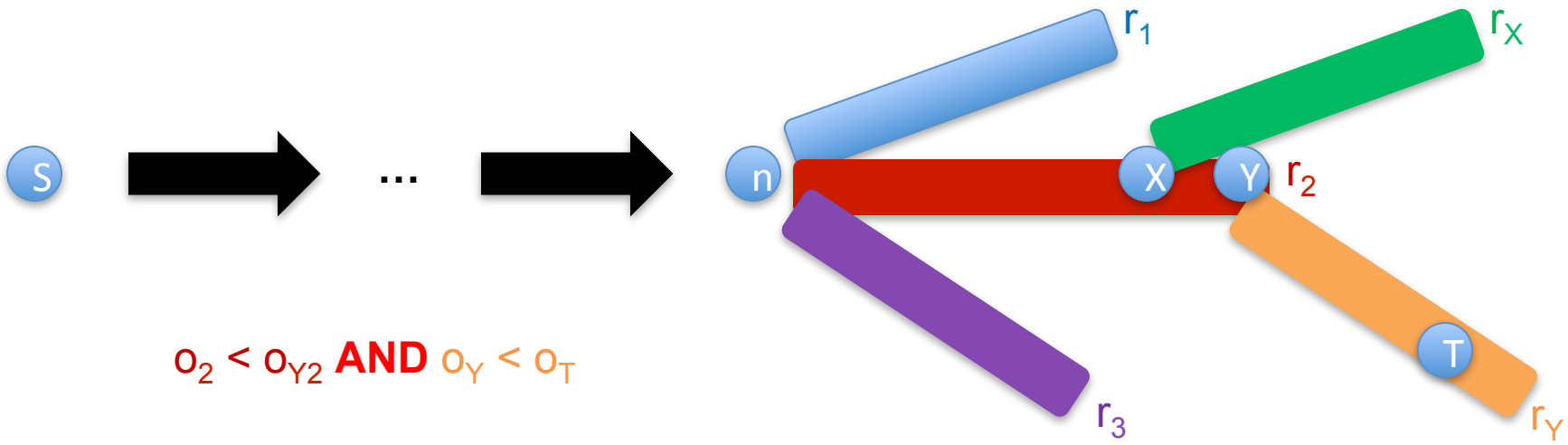
- pfsP terminates search
  - When: current node n is contained in a route before target t
  - How: joins *routes*[n] with *routes*[t] & stops after finding a common route entry




*routes*[n] = {<$r_1$:$o_1$>,<$r_2$:$o_2$>,<$r_3$:$o_3$>} **JOIN** *routes*[T] = {…,<$r_2$:$o_T$>,…}

# Evaluating PATH queries cont.

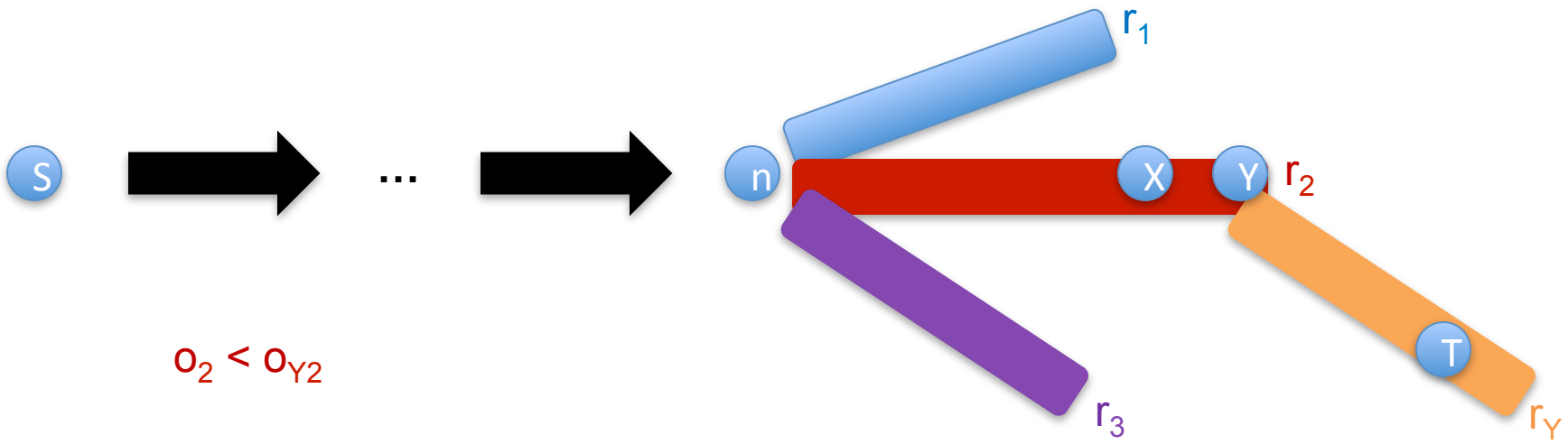
- pfsH terminates search
  - When: current node n is contained in a route that is connected with a route containing target t
  - How: for each route r containing n, joins *edges*[r] with *routes*[t] & stops after finding a common route entry




*edges*[$r_2$] = {<$r_X$,X:$o_{X2}$:$o_X$>,<$r_Y$,Y:$o_{Y2}$:$o_Y$>} **JOIN** *routes*[T] = {…,<$r_Y$:$o_T$>,…}

# Evaluating PATH queries cont.

- pfsL
  - Constructs list $\mathcal{T}$ of the links before target
  - Visits only the links in a route
  - Terminates search
    - When: current node n is contained in a route before a link of $\mathcal{T}$
    - How: for each route r containing n, joins *links*[r] with list $\mathcal{T}$ & stops after finding a common link entry




*links*[$r_2$] = {<X:$o_X$>,<Y:$o_{Y2}$>} **JOIN** $\mathcal{T}$ = {…,<Y,($r_Y$:$o_Y$:$o_T$)>,…}

# Evaluating FLSP queries

- Basic idea
  - Traverse nodes similar to uniform-cost search
  - Compute lower bound of the answer, a candidate answer
    - Using P-Index => method spP, L-Index => method spL
    - Triggers early termination condition
    - Prunes search space
  - At each iteration:
    - Pop current node n
    - Check termination condition against candidate answer
    - Update candidate answer through n
    - Access routes containing n, *routes*[n] from P-Index
      - For each route, push nodes after n in search stack iff expanding them would give answer better than candidate

# Handling frequent updates

- P-Index, H-Index, L-Index as inverted files on disk
  - Updates -> adding new routes
  - Not consider each new route separately
  - Batch updates, consider set of new routes
- Basic idea:
  - Build memory resident indices for new routes
  - Merge disk-based indices with memory resident ones

# Future work

- Three directions:
  - ① Address other kind of updates
    - Insert nodes in routes – as dynamic pickup and delivery problem is solved
    - Update time intervals of nodes
  - ② Evaluate other types of queries
    - Trip planning or optimal sequenced like queries
      - Each node is an instance of a class in set C, e.g. {Museum,Stadium,Restaurant}
      - Find a path passing through a Museum, then a Stadium and finally a Restaurant
  - ③ Combine query evaluation with keyword search
    - Starting and ending node given as set of keywords
    - Find a path passing through a Restaurant relevant to "sea food, lobster"


GONE
SURFING!