Karlsruhe Institute of Technology

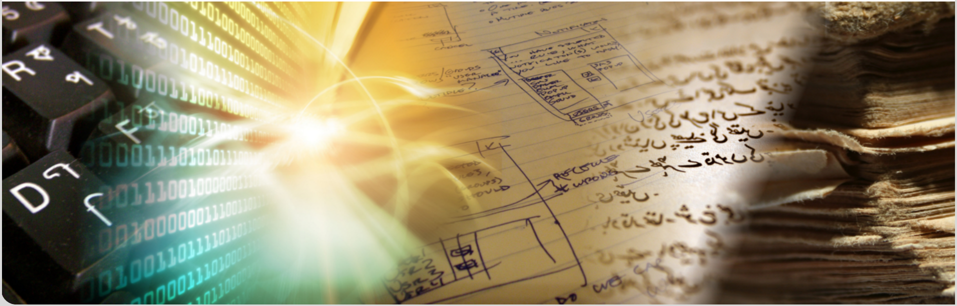
Fakultät für Informatik

Bachelor’s Thesis
Flexible User-Friendly Trip Planning Queries

Adviser: Saeed Taghizadeh

Violina Zhekova | May 9, 2019

INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION (IPD), CHAIR PROF. BÖHM



Outline

1. Motivation
2. Related Work
3. Example
4. Problem Definition
5. Introducing the Operators
   - Equality operator
   - Inequality operator
   - Or operator
   - Order operator
6. Evaluation
   - Experiments
   - Equality operator
   - Inequality operator
   - Or operator
   - Order operator
7. Conclusion
8. Discussion

Motivation

- *Sequenced Route Queries* (SRQ) - finding routes passing through multiple *Points of Interest* (PoIs)
- Advances in *Location Based Services* (LBS) and *Geographic Information System* (GIS) applications (e.g. logistics and supply chain management)
- Aim: Designing a language to enable the user to express his query requirements in a flexible manner

Related work

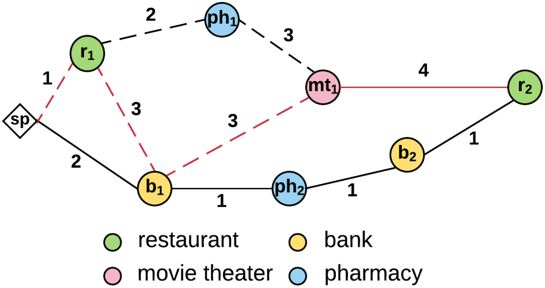
- Vector vs. metric spaces
- *Trip Planning Queries* (TPQ) [1]
- The *Optimal Sequenced Route* (OSR) Query [2]
- The Skylyne concept applied to SRQ [3]
- Considering multiple factors of a route – rating of PoIs, distance and category weights, dynamic factors (e.g. traffic information) [4]
- SRQ issued by users moving along a route [5]
- *Multi-rule Partial Sequenced Route* (MRPSR) Query [6]

Example

- **Category sequence**: (restaurant, bank, movie theater, restaurant)
  Condition: equal restaurants
- *Optimal Sequenced Route* (OSR): (r_1, b_1, mt_1, r_2), length: 11 (shown with red lines)
- Optimal route with equal restaurants: (r_1, b_1, mt_1, r_1), length: 12 (shown with dashed lines)

Example

- **Category sequence**: (restaurant, bank, movie theater, restaurant)
  Condition: equal restaurants
- *Optimal Sequenced Route* (OSR): (r_1, b_1, mt_1, r_2), length: 11 (shown with red lines)
- Optimal route with equal restaurants: (r_1, b_1, mt_1, r_1), length: 12 (shown with dashed lines)



Problem definition

- **Problem**: Need for flexibility in route finding queries
- **Solution**: Developing a query language operators to fulfill the essential user's requirements:
  - Relationships among the PoIs
  - Order and priority of the PoIs
  - Expressing multiple travel variations
- Proposing four essential operators: ”equality” operator, ”inequality” operator, ”or” operator, ”order” operator
- Making use of existing approaches (PNE (*Progressive Neighbor Exploration*) [2]) to transform the complex user query

Problem definition

- **Problem**: Need for flexibility in route finding queries
- **Solution**: Developing a query language operators to fulfill the essential user's requirements:
  - Relationships among the PoIs
  - Order and priority of the PoIs
  - Expressing multiple travel variations

- Proposing four essential operators: ”equality” operator, ”inequality” operator, ”or” operator, ”order” operator
- Making use of existing approaches (PNE (*Progressive Neighbor Exploration*) [2]) to transform the complex user query

”Equality” operator

- *Input*: A category sequence $M = (c_1, c_2, ..., c_l)$, a starting point *sp* in $\mathbb{R}^2$ and indices of the equal PoIs *i* and *j*, where $c_i = c_j$
- *Output*: Optimal route $R = (r_1, r_2, ..., r_l)$, where $r_i = r_j$

- **Proposed approach**: uses the Progressive Neighbor Explorator (PNE) as its base to upgrade on and extends it with a heuristic approach to shrink the search space
- **Baseline/trivial approach**: extends the PNE with forcing the PoIs r_i and r_j to be equal; does not use optimization techniques

Heuristic

Given a sequence of categories $M = (c_1, c_2, ..., c_l)$ and a PSR $R' = (r_1, r_2, ..., r_k)$ the ***heuristic*** for this route is defined as:

$$h(R') = \max_{i \in [k+1, l]} nearestNeighbor(r_k, C_{M_i}) \quad (1)$$

- Informal: The heuristic of a certain PSR is the maximum distance out of the distances to the nearest PoIs from the set of categories that are yet to be expanded.

The ***lower bound*** of a PSR R' represents the sum of its length and its heuristic:

$$LB(R') = length(R') + h(R') \quad (2)$$

- The proposed algorithm uses a heap, sorted by the lower bound of the routes

Heuristic

Given a sequence of categories $M = (c_1, c_2, ..., c_l)$ and a PSR $R' = (r_1, r_2, ..., r_k)$ the ***heuristic*** for this route is defined as:

$$h(R') = \max_{i \in [k+1, l]} nearestNeighbor(r_k, C_{M_i}) \tag{1}$$

- Informal: The heuristic of a certain PSR is the maximum distance out of the distances to the nearest PoIs from the set of categories that are yet to be expanded.

The ***lower bound*** of a PSR R' represents the sum of its length and its heuristic:

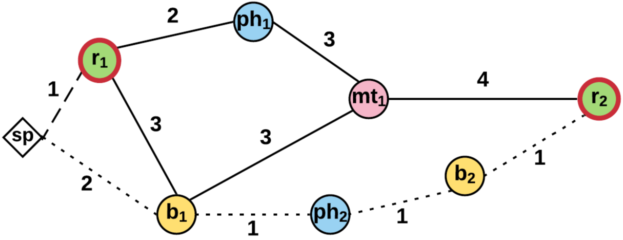
$$LB(R') = length(R') + h(R') \tag{2}$$

- The proposed algorithm uses a heap, sorted by the lower bound of the routes

Example: Step 1

$sp, M = (r, b, mt, r), EQUAL(0, 3)$

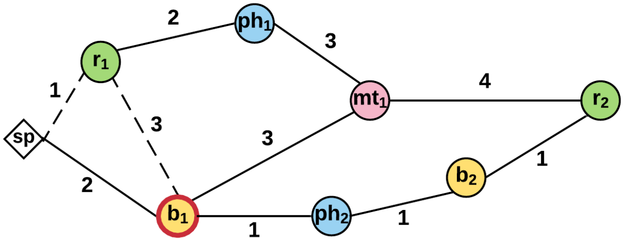
- Optimal route found with PNE: (r_1, b_1, mt_1, r_2)
- Dummy SR: (r_1, b_1, mt_1, r_1); Upper Bound: $length(dummySR) = 12$, Candidate SR: *dummySR*



Step	Heap contents (PSR R : $length(R)$, $heuristic(R)$)
1	$(r_1 : 1, 5)$, $(r_2 : 5, 4)$

Example: Step 2

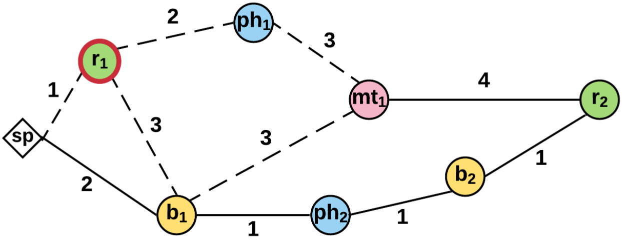
$sp, M = (r, b, mt, r), EQUAL(0, 3)$



Step	Heap contents (PSR R : $length(R), heuristic(R)$)
1	$(r_1 : 1, 5), (r_2 : 5, 4)$
2	$(r_1, b_1 : 4, 3), (r_2 : 5, 4)$

Example: Step 8

$sp, M = (r, b, mt, r), EQUAL(0, 3)$

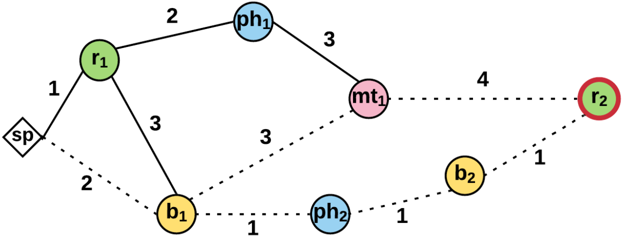


Candidate SR: $(r_1, b_1, mt_1, r_1 : 12, 0)$

Step	Heap contents (PSR R : $length(R)$, $heuristic(R)$)
7	$(r_1, b_1, mt_1 : 7, 5), (r_2, b_1, mt_1 : 11, 4), (r_2, b_2, mt_1 : 11, 4),$ $(r_1, b_2, mt_1 : 11, 5)$
8	$(r_2, b_1, mt_1 : 11, 4), (r_2, b_2, mt_1 : 11, 4), (r_1, b_2, mt_1 : 11, 5)$

Example: Step 9

$sp, M = (r, b, mt, r), EQUAL(0, 3)$

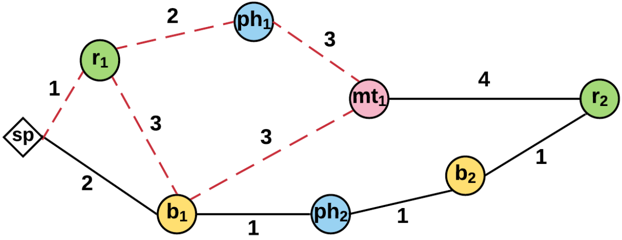


Candidate SR: ~~$(r_2, b_1, mt_1, r_2 : 15, 0)$~~

Step	Heap contents (PSR R : $length(R), heuristic(R)$)
8	$(r_2, b_1, mt_1 : 11, 4), (r_2, b_2, mt_1 : 11, 4), (r_1, b_2, mt_1 : 11, 5)$
9	$(r_2, b_2, mt_1 : 11, 4), (r_1, b_2, mt_1 : 11, 5)$

Example: Result

$sp, M = (r, b, mt, r), EQUAL(0, 3)$



Found SR: $(r_1, b_1, mt_1, r_1 : 12, 0)$

”Inequality” operator

- *Input*: A category sequence $M = (c_1, c_2, ..., c_l)$, a starting point *sp* in $\mathbb{R}^2$ and indices of the unequal PoIs *i* and *j*, where $c_i = c_j$
- *Output*: Optimal route $R = (r_1, r_2, ..., r_l)$, where $r_i \neq r_j$

- **Proposed approach**: generates routes based on the Progressive Neighbor Explorator (PNE) and inspects partial routes for satisfying the requirement of unequal PoIs and modifies them accordingly

”Or” operator

- **OR sequence:** An OR sequence $OR = (M_1, M_2, ..., M_m)$ represents the disjunction of category sequences, such as $M_1 = (c_1, c_2, ..., c_l)$.

- *Input*: A sequence of OR sequences $S = (OR_1, OR_2, ..., OR_n)$ and a starting point *sp* in $\mathbb{R}^2$
- *Output*: Optimal route $R = (r_1, r_2, ..., r_l)$

- **Proposed approach**: progressively inspects each option M_i from the OR sequences OR_i in $S = (OR_1, OR_2, ..., OR_n)$, compares them and continues with the best one, based on length, until it reaches a full sequenced route
- **Proposed approach**: runs the PNE algorithm on all possible combinations of the query to find the shortest route out of them

”Or” operator

- **OR sequence:** An OR sequence $OR = (M_1, M_2, ..., M_m)$ represents the disjunction of category sequences, such as $M_1 = (c_1, c_2, ..., c_l)$.

- *Input*: A sequence of OR sequences $S = (OR_1, OR_2, ..., OR_n)$ and a starting point *sp* in $\mathbb{R}^2$
- *Output*: Optimal route $R = (r_1, r_2, ..., r_l)$

- **Proposed approach**: progressively inspects each option M_i from the OR sequences OR_i in $S = (OR_1, OR_2, ..., OR_n)$, compares them and continues with the best one, based on length, until it reaches a full sequenced route
- **Proposed approach**: runs the PNE algorithm on all possible combinations of the query to find the shortest route out of them

”Order” operator

- **ORDER sequence:** An order sequence $ORDER = (i_1, i_2, ..., i_k)$, is a sequence of indices in a category sequence $M = (c_1, c_2, ..., c_l)$, which indicate that the categories at the given indices should remain in the fixed positions in this category sequence.
- *Input*: A sequence of categories $M = (c_1, c_2, ..., c_l)$, a starting point sp in $\mathbb{R}^2$ and an ORDER sequence $ORDER = (i_1, i_2, ..., i_k)$
- *Output*: Optimal route $R = (r_1, r_2, ..., r_l)$

- **Proposed approach**: inspects progressively each category option for the indices out of the NOTORDERED sequence, compares them and continues with the best one, based on length, until it reaches a full sequenced route.
- **Proposed approach**: runs the PNE algorithm on all possible permutations of the query to find the shortest route out of them

”Order” operator

- **ORDER sequence:** An order sequence $ORDER = (i_1, i_2, ..., i_k)$, is a sequence of indices in a category sequence $M = (c_1, c_2, ..., c_l)$, which indicate that the categories at the given indices should remain in the fixed positions in this category sequence.
- *Input*: A sequence of categories $M = (c_1, c_2, ..., c_l)$, a starting point *sp* in $\mathbb{R}^2$ and an ORDER sequence $ORDER = (i_1, i_2, ..., i_k)$
- *Output*: Optimal route $R = (r_1, r_2, ..., r_l)$

- **Proposed approach**: inspects progressively each category option for the indices out of the NOTORDERED sequence, compares them and continues with the best one, based on length, until it reaches a full sequenced route.
- **Proposed approach**: runs the PNE algorithm on all possible permutations of the query to find the shortest route out of them

Experiments

- Road network of Berlin, with 428769 crossroad nodes, 504229 road edges, 5548 PoIs and 7 category types
- 1000 queries with randomly selected starting points

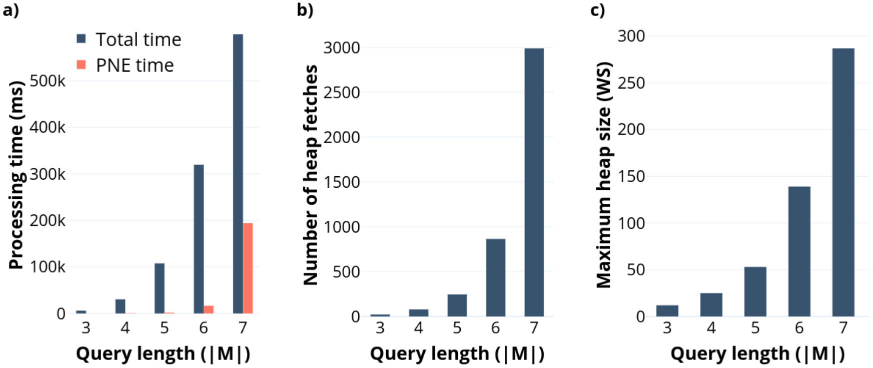
Categories	*Size*	*Frequency*
Restaurants	2081	High
Coffee shops	1002	
Pubs and bars	958	
Atms/Banks	597	Middle
Pharmacies	589	
Gas stations	180	Low
Movie theaters	141	

- **Quantitative evaluation criteria**: run time, number of heap fetches, maximum heap size (work space)
- **Qualitative evaluation**: compare against baseline approaches

”Equality” operator - query length

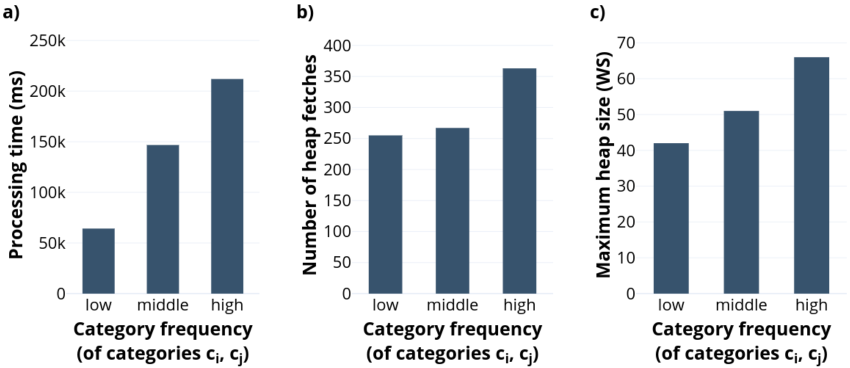
Karlsruhe Institute of Technology

- **Evaluation parameters**: (1) query length, (2) frequency of the categories c_i, c_j in the category sequence and (3) distance between the categories c_i, c_j



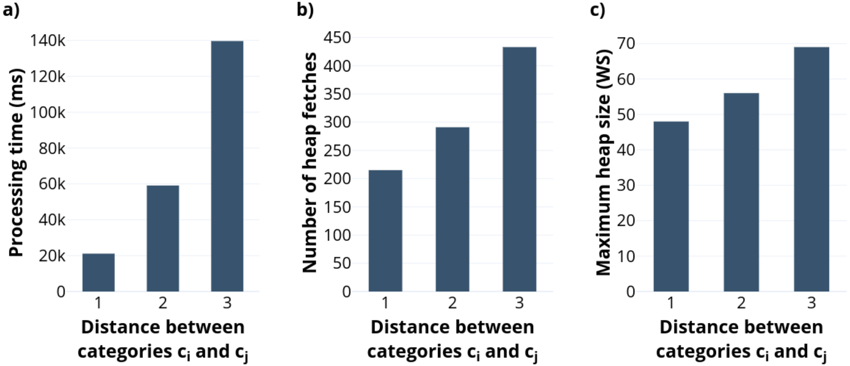
”Equality” operator - category frequency

- **Evaluation parameters**: (1) query length, (2) frequency of the categories c_i, c_j in the category sequence and (3) distance between the categories c_i, c_j

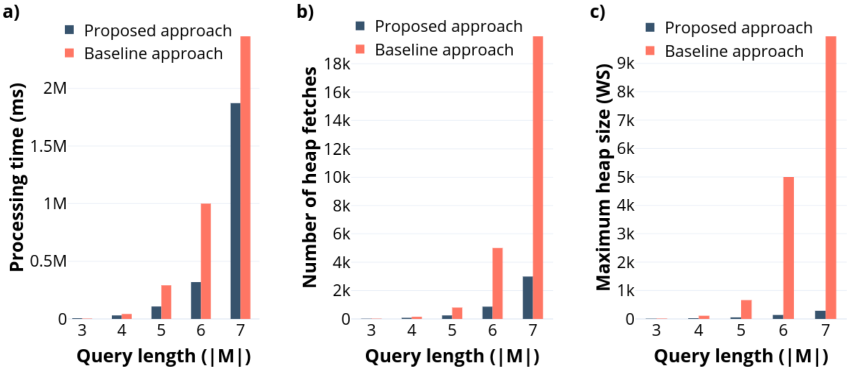


”Equality” operator - category distance

- **Evaluation parameters**: (1) query length, (2) frequency of the categories c_i, c_j in the category sequence and (3) distance between the categories c_i, c_j

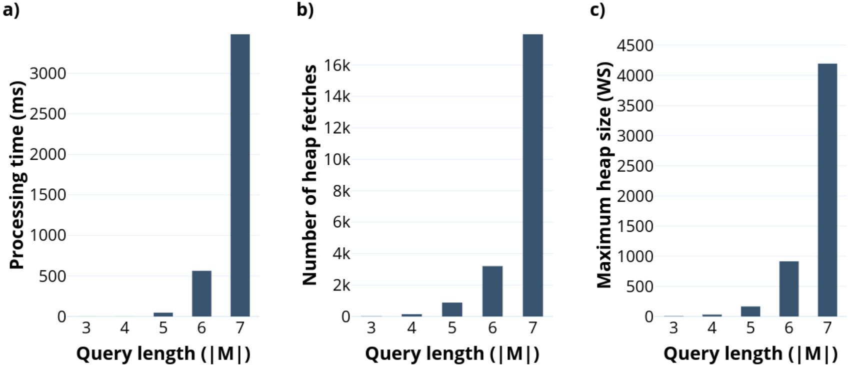


”Equality” operator - baseline approach



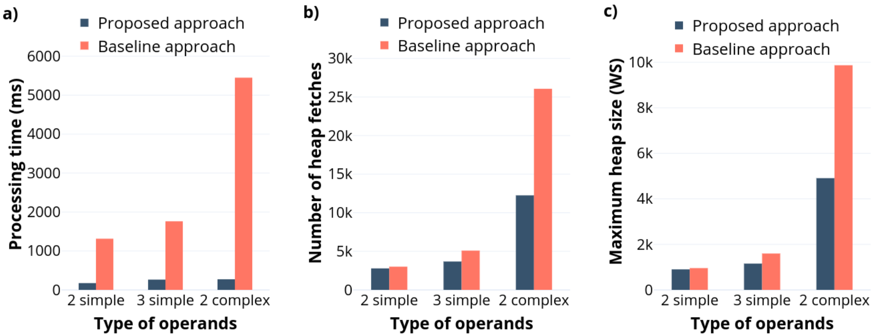
”Inequality” operator - query length

- **Evaluation parameters**: (1) query length, (2) frequency of the categories c_i, c_j in the category sequence and (3) distance between the categories c_i, c_j



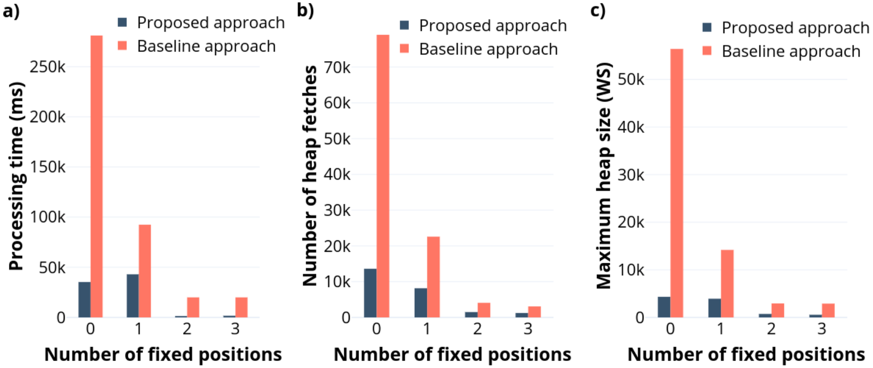
”Or” operator

- **Evaluation parameter**: type and number of operands
  - simple OR operand: $|M| = 1$
  - complex OR operand: $|M| > 1$



”Order” operator

- **Evaluation parameter**: number of fixed positions in *M*



Conclusion

- Proposed four essential route query language operators
- Developed algorithms that deliver an optimal result in metric spaces
- Evaluated the approaches qualitatively and quantitatively and proved the efficiency of the algorithms

Discussion and Future Work

- Easily modified: using a different quantifying parameter such as travel duration, finding *k* optimal routes
- Additional operators are possible: ”hop”, ”and”, ”not”, ”necessity”
- A different approach to implementing the operators
  - Skyline concept [3]
- Extending the proposed algorithms to also support dynamic road networks, in which traffic information is provided (e.g., travel time, traffic congestion, etc.). [7]

The End

Thank you for the attention!

Literatur I

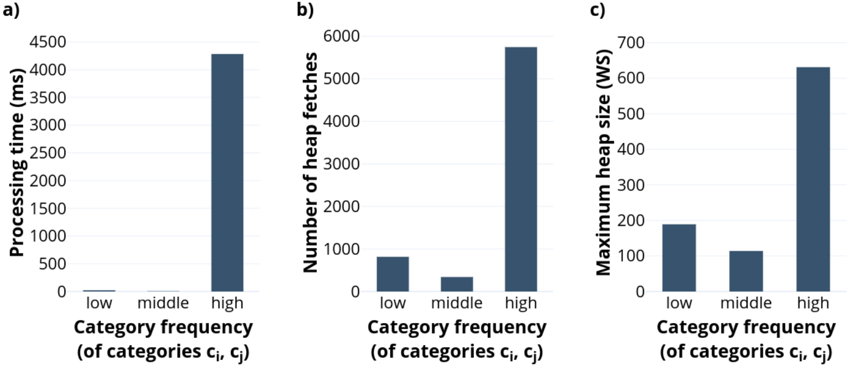
- Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios and Shang-Hua Teng. *On trip planning queries in spatial databases*. In Bauzer Medeiros C., Egenhofer M.J., Bertino E. (eds) Advances in Spatial and Temporal Databases (SSTD), pages 273–290, 2005.
- Mehdi Sharifzadeh, Mohammad Kolahdouzan and Cyrus Shahabi. *The optimal sequenced route query*. The VLDB Journal, 17:765–787, 2008.
- Yuya Sasaki, Yoshiharu Ishikawa, Yasuhiro Fujiwara and Makoto Onizuka. *Sequenced route query with semantic hierarchy*. In EDBT, pages 37–48, 2018.
- Jian Dai, Chengfei Liu, Jiajie Xu4 and Zhiming Ding. *On personalized and sequenced route planning*. World Wide Web, 19:679–705, 2016.

Literatur II

- Jochen Eisner and Stefan Funke. *Sequenced route queries: Getting things done on the way back home*. In SIGSPATIAL '12 Proceedings of the 20th International Conference on Advances in Geographic Information Systems, pages 502–505, 2012.
- Haiquan Chen, Wei-Shinn Ku, Min-Te Sun and Roger Zimmermann. *The partial sequenced route query with traveling rules in road networks*. Geoinformatica, 15:541–569, 2011.
- Nirmesh Malviya, Samuel Madden and Arnab Bhattacharya. A continuous query system for dynamic route planning. *A continuous query system for dynamic route planning*. In ICDE '11 Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, pages 792–803, 2011.

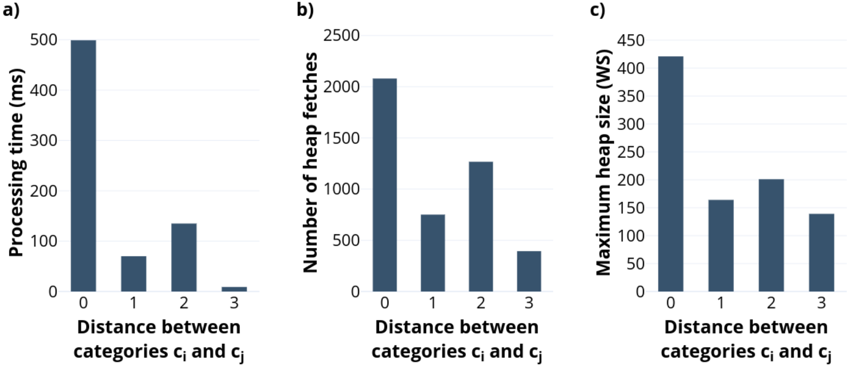
Inequality operator - category frequency

- **Evaluation parameters**: (1) query length, (2) frequency of the categories c_i, c_j in the category sequence and (3) distance between the categories c_i, c_j



Inequality operator - category distance

- **Evaluation parameters**: (1) query length, (2) frequency of the categories c_i, c_j in the category sequence and (3) distance between the categories c_i, c_j



PNE Algorithm

Algorithm 1: PNE

```
fetch a PSR from the heap;
switch $s = size(PSR)$ do
    case $s == l$ do
        PSR is the optimal route;
        return PSR;
    case $s \neq l$ do
        a)
            NN($r_{|PSR|}, C_{M_{|PSR|+1}}$);
            update PSR and perform trimming in case it is a candidate SR ;
            put PSR back on the heap ;
        b)
            kNN($r_{|PSR|-1}, C_{M_{|PSR|}}$);
            generate a new PSR and place it on the heap;
```

Notations

Symbol	*Meaning*
C_i	a point set for a category in $\mathbb{R}^2$
$\lvert C_i \rvert$	cardinality of the set C_i
n	number of point sets C_i
$dist(.,.)$	distance function in $\mathbb{R}^2$
M	category sequence, $= (c_1, c_2, ..., c_l)$
$\lvert M \rvert$	l, size of sequence M = number of items in M
c_i	ith member of M
R	route, $= (r_1, r_2, ..., r_r)$
$\lvert R \rvert$	r, size of route R = number of points in R
r_i	ith point in R
$length(R)$	length of R
$length(sp, R)$	length of $R_{sp} = (sp, r_1, r_2, ..., r_r), = length(R_{sp})$
$Q(sp, M)$	sequenced route query
$NN(r_q, c_p)$	nearest neighbor from the category point set C_p to a route point r_q
$KNN(r_q, c'_p)$	next nearest neighbor r'_p from the category point set C_p to a route point r_q

Modified PNE

Algorithm 2: modifiedPNE()

```
foreach $r_1$ in $C_{M_1}$ do Checking the upper bound for every $r_1$ neighbor of sp in the category set $C_{M_1}$
    build a new PSR with $r_1$;
    if $LB(PSR) <= UB$ then
        place the new PSR ($r_1$) on the heap;
while heap is not empty do
    current ← fetch a PSR from the heap;
    switch $s = size(current)$ do
        case $s <= j - 1$ do Finding PSRs before $r_j$
            caseBefore();
        case $s = j$ do Finding PSR containing $r_j$
            caseContaining();
        case $s = j + 1$ do Finding PSR after/containing $r_j$
            caseAfterOrContaining();
        case $s >= j + 2$ do Finding PSRs after $r_j$
            caseAfter();
return foundSR
```

NEO - Trimming

Procedure trim(*PSR*)

```
if $size(PSR) = l$ then
    if $PSR[i] \neq PSR[j]$ then
        // Optimization:  length check
        if $length(PSR) <= length(candidate)$ then
            update $candidate$;
            place $PSR$ on the $heap$;
    else
        // In case $j$ is the last index in the route, we
           find the kth neighbor of the previous PoI to
           the last one
        if $size(PSR) = j + 1$ then
            kNN($r_{|PSR|-1}, C_{M_{|PSR|}}$);
            generate a new $PSR$;
            trim($PSR$);
else
    // Optimization:  length check
    if $length(PSR) <= length(candidate)$ then
        place $PSR$ on the $heap$;
```