EFFICIENT ANSWERING OF SET CONTAINMENT QUERIES FOR SKEWED ITEM DISTRIBUTIONS

Manolis Terrovitis (IMIS, RC ATHENA)
Panagiotis Bouros(NTUA)
Panos Vassiliadis (UoI)
Timos Sellis (NTUA & IMIS, RC ATHENA)
Nikos Mamoulis (HKU)

THE PROBLEM

- We have huge collections of transactional data
  - Retail store transaction logs
  - Web logs
  - Biomedical databases etc.

- We address the efficient evaluation of boolean containment queries
  - In which transactions were products ‘a’ and ‘b’ sold together?
  - Which users visited only the main page or the download page of our site?

- The problem of indexing set values in secondary storage remains a challenge still
  - Inverted files are the state-of-the-art but longs lists can slow them significantly
  - Skewed distributions cause long lists
  - Most of the recent efforts are concerned with more complicated queries or main memory solutions
  - The problem still exists

- *We propose the Ordered-Inverted file (OIF) index*

QUERIES - SUBSET

tid	products	*tid*	products
1	{*f*,*a*}	9	{*a*,*e*}
2	{*a*,*d*,*c*}	10	{*g*,*c*,*a*}
3	{*c*,*b*,*a*}	11	{*b*,*a*,*e*}
4	{*f*,*a*,*c*}	12	{*b*,*d*,*c*}
5	{*c*,*g*}	13	{*c*,*f*,*a*,*d*,*b*}
6	{*a*,*b*,*g*,*c*,*d*,*e*}	14	{*b*,*d*}
7	{*a*,*d*,*b*}	15	{*e*}
8	{*a*,*e*,*b*}	16	{*b*,*f*,*a*}

- ***Find all transactions that contain 'a', 'b' and 'd' (subset)***

QUERIES - EQUALITY

tid	products	*tid*	products
1	{*f*,*a*}	9	{*a*,*e*}
2	{*a*,*d*,*c*}	10	{*g*,*c*,*a*}
3	{*c*,*b*,*a*}	11	{*b*,*a*,*e*}
4	{*f*,*a*,*c*}	12	{*b*,*d*,*c*}
5	{*c*,*g*}	13	{*c*,*f*,*a*,*d*,*b*}
6	{*a*,*b*,*g*,*c*,*d*,*e*}	14	{*b*,*d*}
7	{*a*,*d*,*b*}	15	{*e*}
8	{*a*,*e*,*b*}	16	{*b*,*f*,*a*}

- *Find all transactions that contain 'a', 'b' and 'd' (subset)*
- ***Find all transactions that contain exactly 'a', 'b' and 'd' (equality)***

QUERIES - SUPERSET

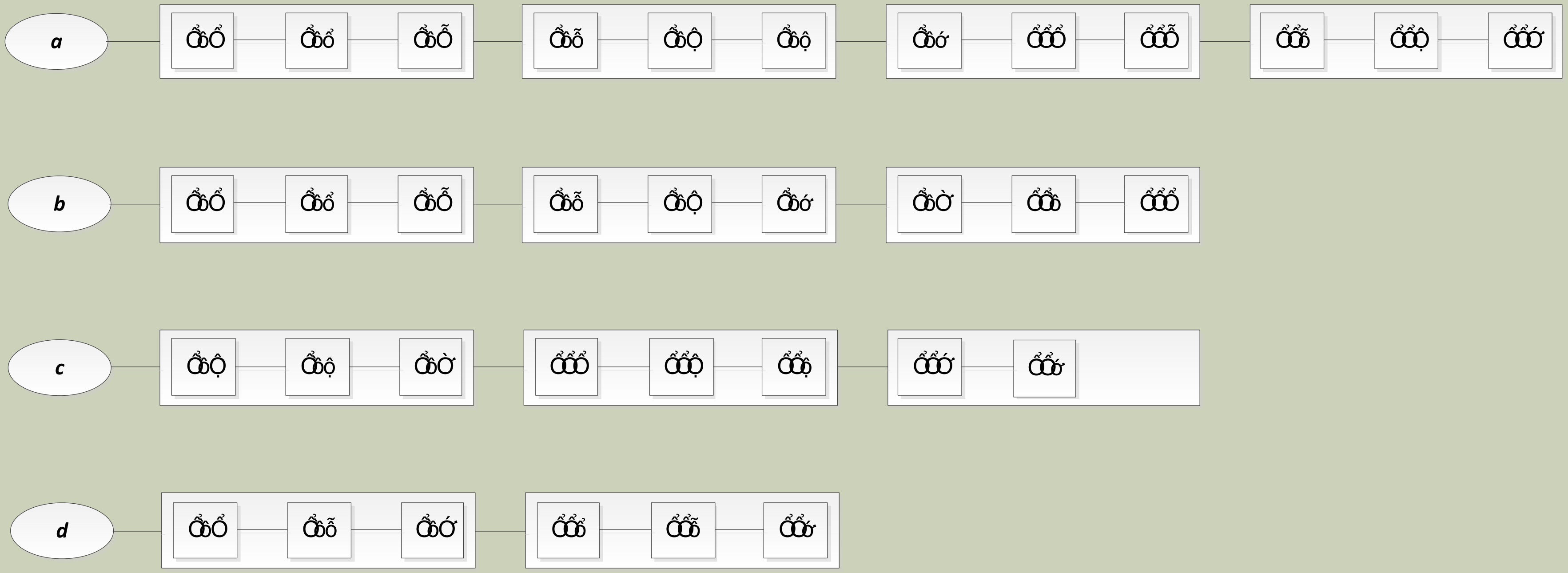
tid	**products**	*tid*	**products**
1	{f,a}	9	{a,e}
2	{a,d,c}	10	{g,c,a}
3	{c,b,a}	11	{b,a,e}
4	{f,a,c}	12	{b,d,c}
5	{c,g}	13	{c,f,a,d,b}
6	{a,b,g,c,d,e}	14	{b,d}
7	{a,d,b}	15	{e}
8	{a,e,b}	16	{b,f,a}

- *Find all transactions that contain 'a', 'b' and 'd' (subset)*
- *Find all transactions that contain exactly 'a', 'b' and 'd' (equality)*
- *Find all transactions that contain only items from 'a', 'b' and 'd' (superset)*

ORIGINAL DATA

id	Items	id	Items	id	Items
101	{g, b, a, d}	107	{d, h}	113	{a}
102	{a, e, b}	108	{b, a, f}	114	{a, d}
103	{f, e, a, b}	109	{b, c}	115	{j, c, a}
104	{d, b, a}	110	{j, b, g}	116	{i, c}
105	{a, b, f, c}	111	{a, c, b }	117	{a, c, h}
106	{c, a}	112	{i, d}	118	{d, c}

SIMPLE INVERTED FILE



Terrovitis et. al., EDBT'11

MOTIVATION

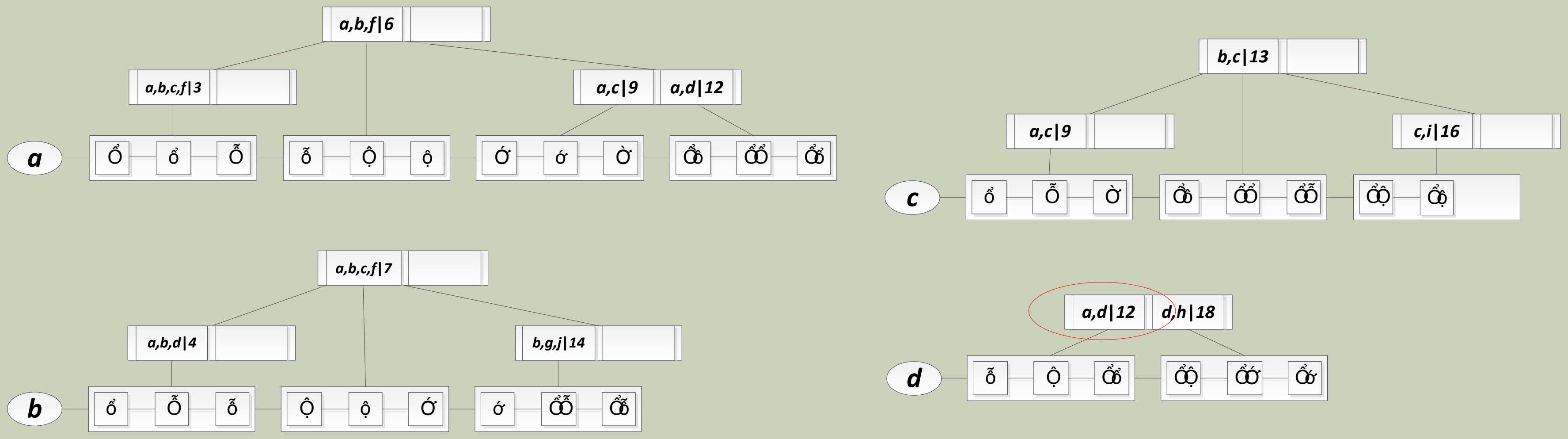
- Lists grow large
  - When the (database size) / (domain size) ratio grows big
  - When the item distribution is skewed
- Using indexing on lists (e.g., skip lists, b-tree) does not offer by itself a big advantage
  - Only if the selectivities are extremely small
- Most frequent items are the key
  - Most frequent items cause most problems
  - Most frequent items are most frequently involved in queries
- Basic Idea
  - Introduce a global order to records
  - Records that contain common frequent items are placed close to each other
  - Postings in lists follow the same order

ORDERED DATA

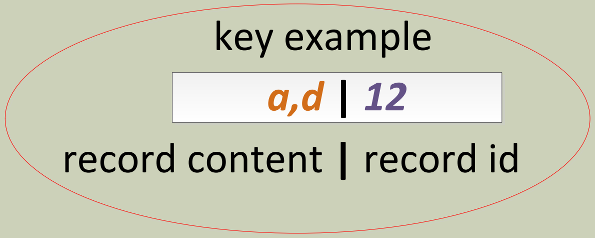
id	Items	Id	Items	id	Items
1	{a}	7	{a, b, f, e}	13	{b, c}
2	{a, b, c }	8	{a, b, e}	14	{b, g, j}
3	{a, b, c, f}	9	{a, c}	15	{c, d}
4	{a, b, d}	10	{a, c, h}	16	{c, i}
5	{a, b, d, g}	11	{a, c, j}	17	{d, i}
6	{a, b, f}	12	{a, d}	18	{d, h}

- A global order for records
- Records sorted internally by frequency (most frequent first)
- Dataset sorted lexicographically (items are compared based on their support)
- New ids reflecting the new order are assigned (storage might not reflect the new order)

ORDERED INVERTED FILE



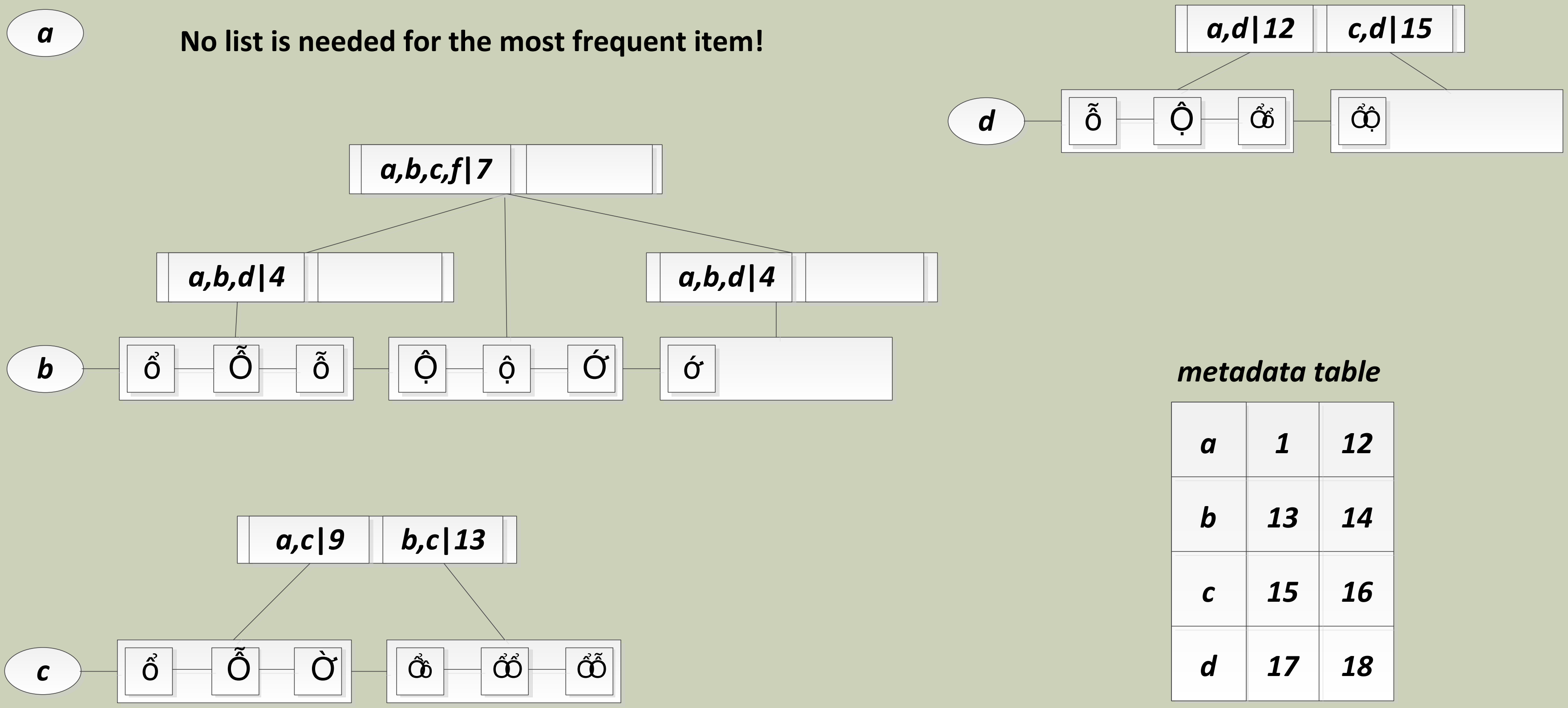
- Lists are created over the new ids
- Indexed by a sparse B-tree
- One key per page



Terrovitis et. al., EDBT'11

ORDERED INVERTED FILE
METADATA TABLE

No list is needed for the most frequent item!



metadata table

a	*1*	*12*
b	*13*	*14*
c	*15*	*16*
d	*17*	*18*

- In the list of item x we do not need to keep explicit postings for records whose most frequent item is x
- We save 1/(average record size) of the total postings

QUERY EVALUATION

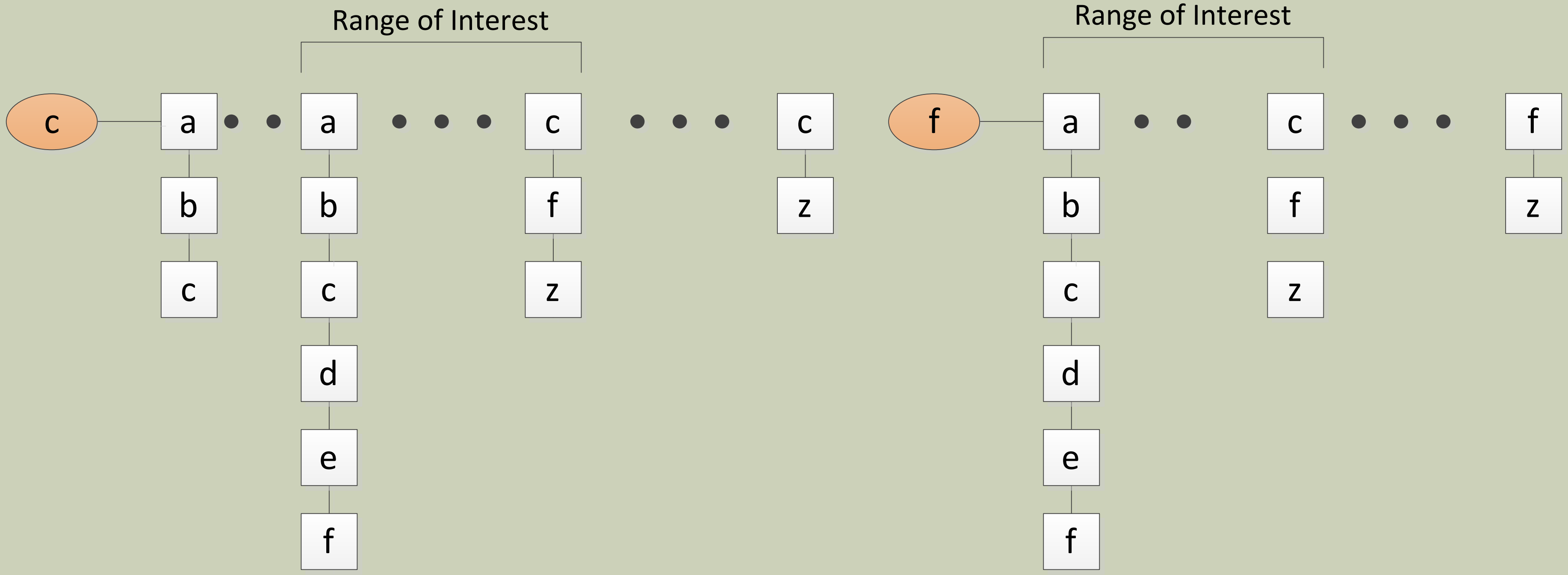
- The evaluation of containment on OIF is similar to the evaluation on IF
- The difference is that instead of intersecting the whole lists, we intersect only a part of them

1. Find the bounds that contain possible answers in each list
   - Range of Interest (RoI) of each query
   - The Range of Interest are calculated using only the query set
2. Use the B-tree to access the block that contains the lower bound
3. Intersect only the part of the lists that is between the bounds

- Observations:
  - We use the metadata for verifying some results
  - Equality query becomes a value retrieval query on the B-tree

SUBSET QUERIES

QS= {*c*, *f*}

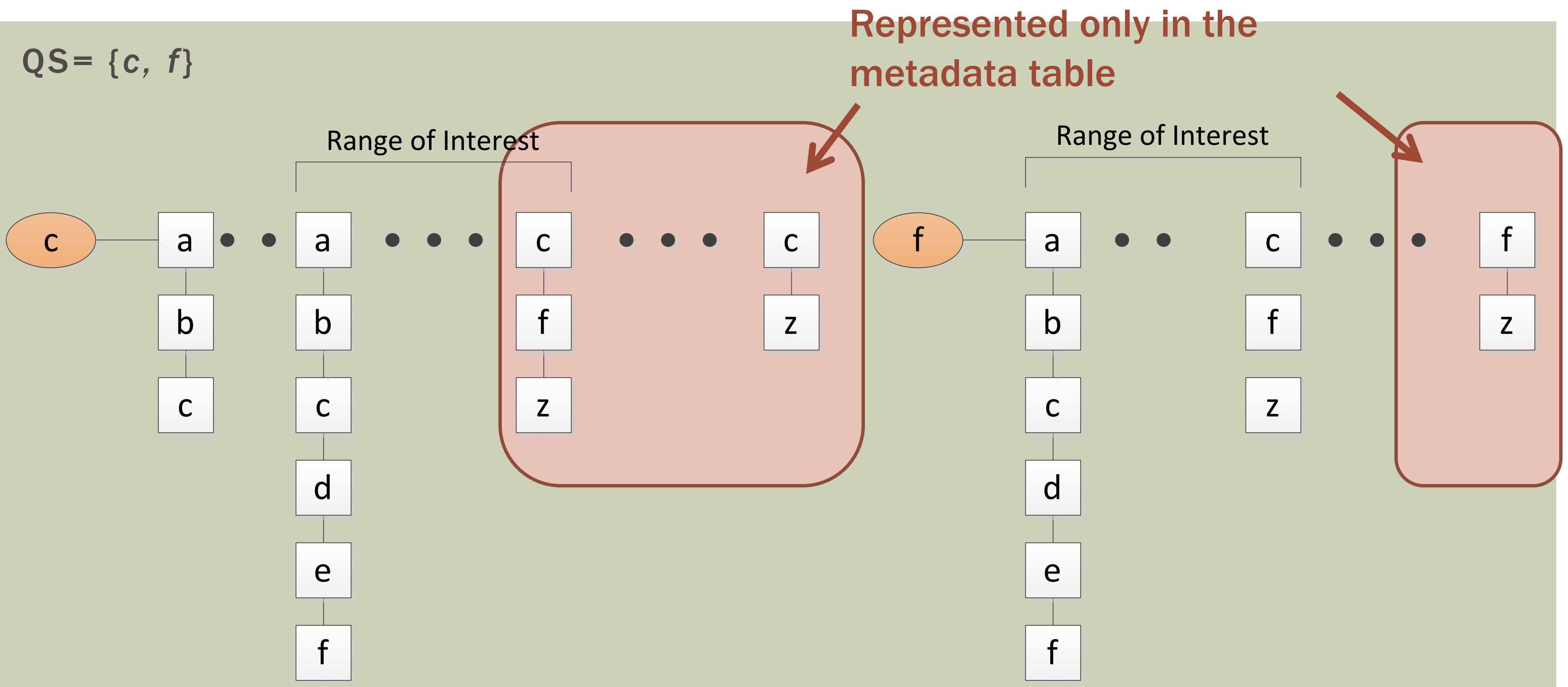


In $QS=o_{q1},\dots,o_{qn}$

Lower bound = $a,b,..,o_{qn}$ upper bound=$o_{q1},\dots,o_{qn},z$

SUBSET QUERIES

QS= {*c*, *f*}

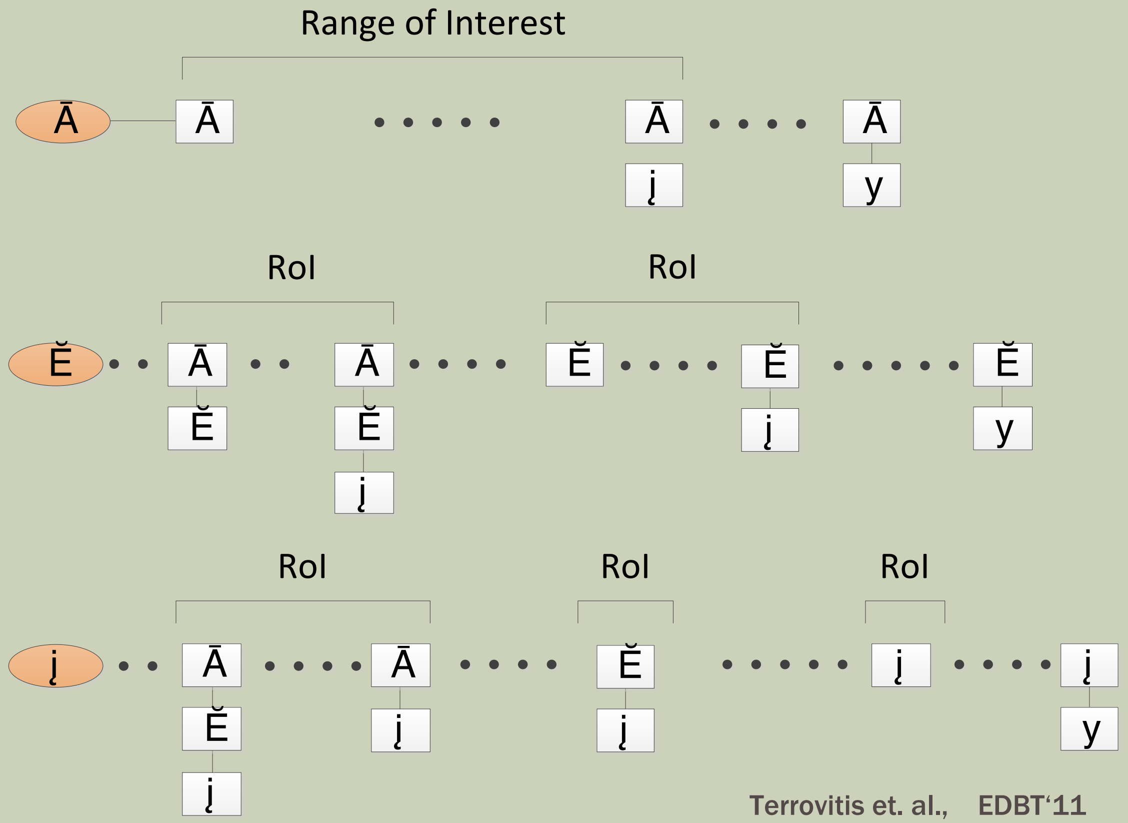


In QS=$o_{q1},\dots,o_{qn}$

Lower bound = $a,b,..,o_{qn}$ upper bound=$o_{q1},\dots,o_{qn},z$

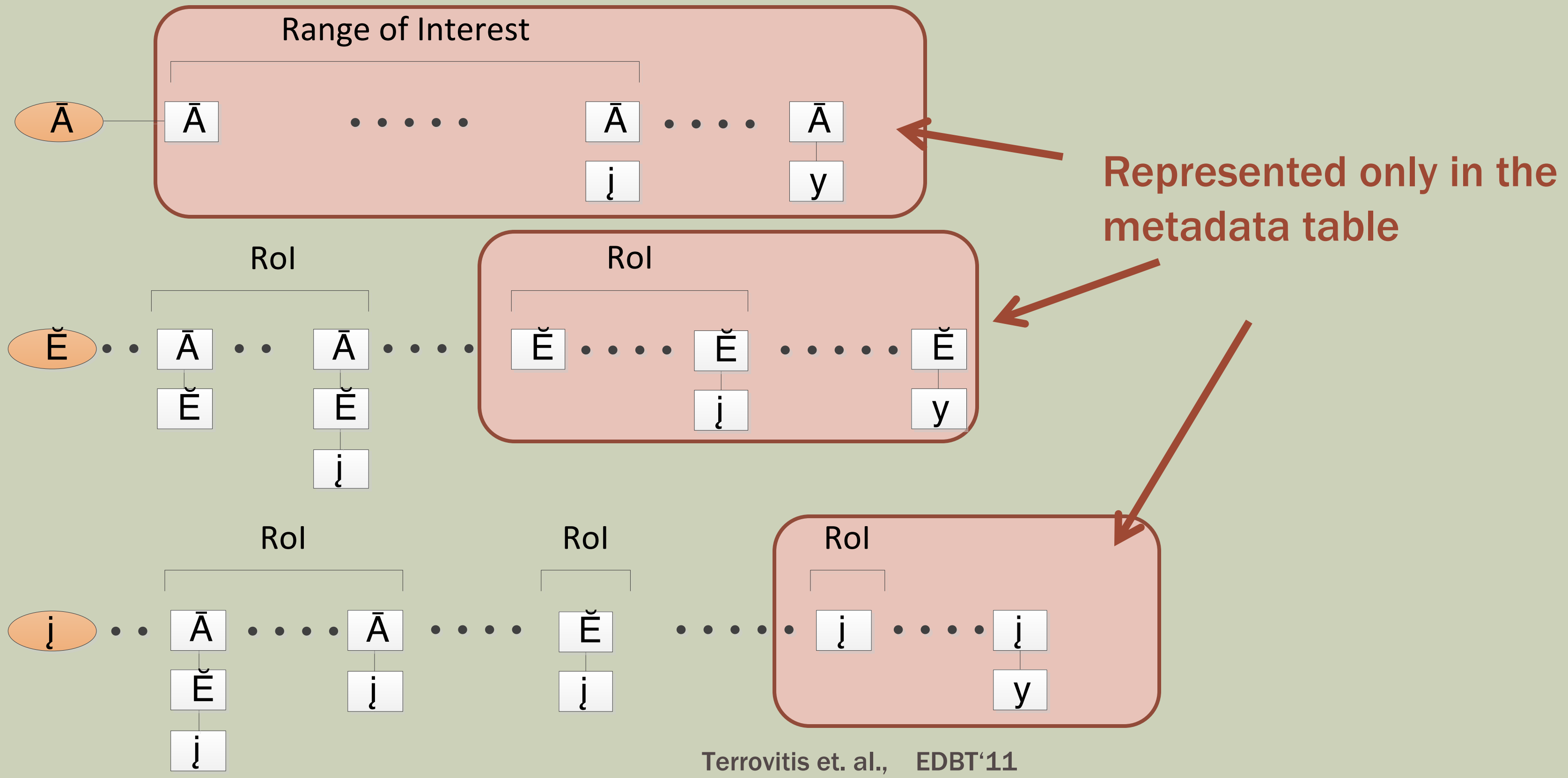
SUPERSET QUERIES

- $QS = \{a,c,f\}$
- Its $2^{|QS|}$-1 equality queries
- We group them by the most frequent item



SUPERSET QUERIES

- $QS = \{a,c,f\}$
- Its $2^{|QS|}$ equality queries
- We group them by the most frequent item



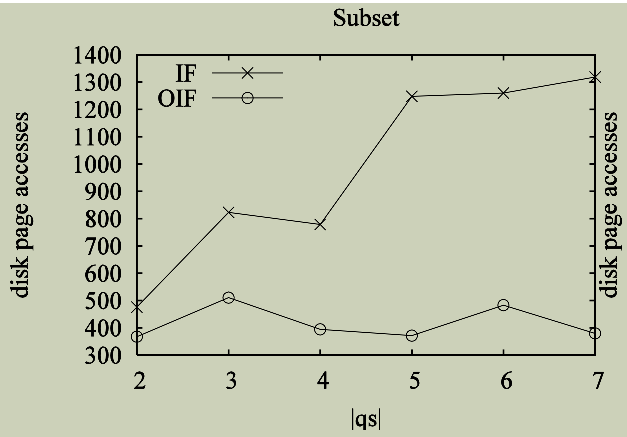
UPDATES

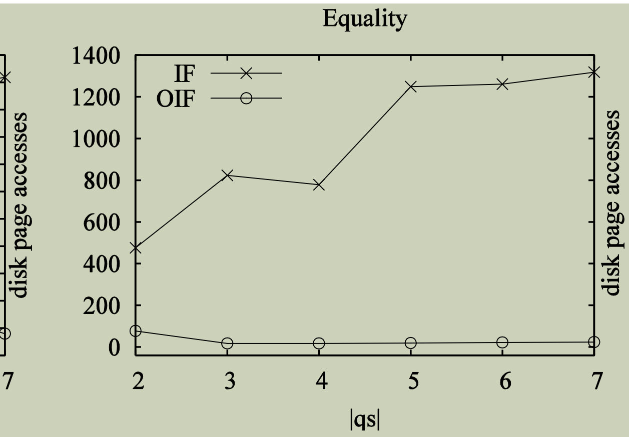
- Updates are usually insertions
- Strategies for updating IF are basically bulk update strategies
  - Incoming postings are kept in a in-memory smaller index
  - Queries have to be evaluated using both indices
  - When buffer becomes full indices are merged
  - ..or merge follows the retrievals of lists in queries
- OIF can be updated in the same way
  - Incoming postings are kept in a in-memory smaller index
  - Before merging the new indices the records have to be sorted
  - Changes in frequencies of items can be ignored if they are small or estimated by sampling

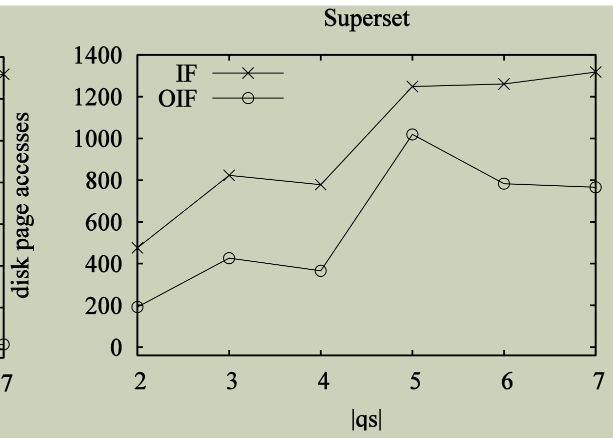
EXPERIMENTS

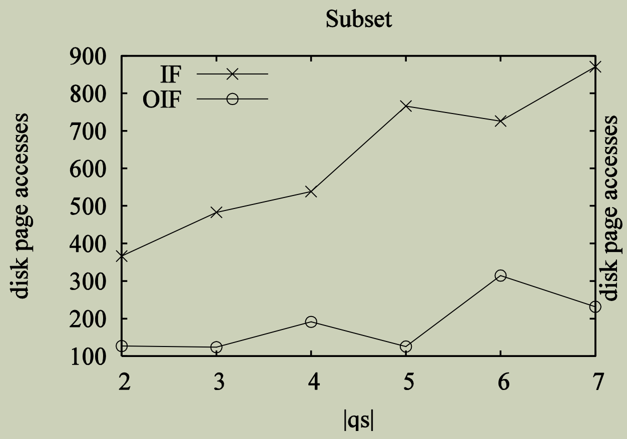
- BerkeleyDB used as storage engine both for OIF and IF
- IO was traced in terms of cache misses
- We traced real execution time and CPU time
- All queries that had at least one answer by using existing records
- Datasets:
  - ms-web: 320k records, 297 items (web log)
  - Ms-nbc: 990k records, 17 items
  - Synthetic datasets default values: 10M records, 5000 items, zipf order = 0.8

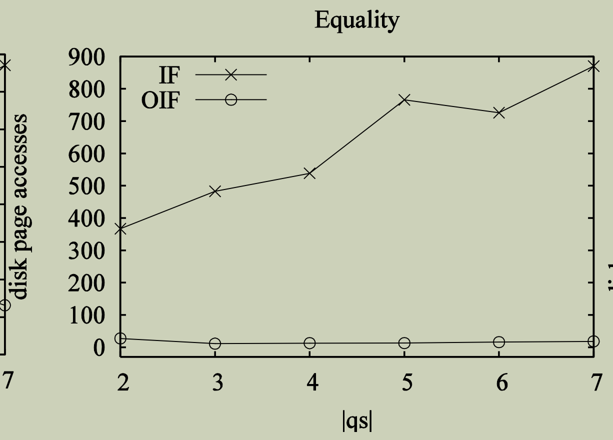
EXPERIMENTS ON REAL WEB LOGS: MS-WEB, MS-NBC

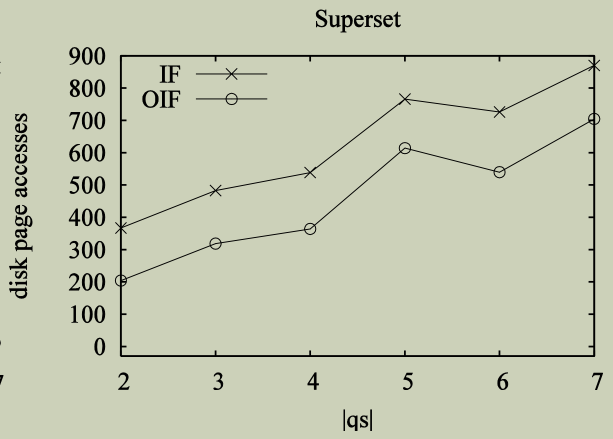






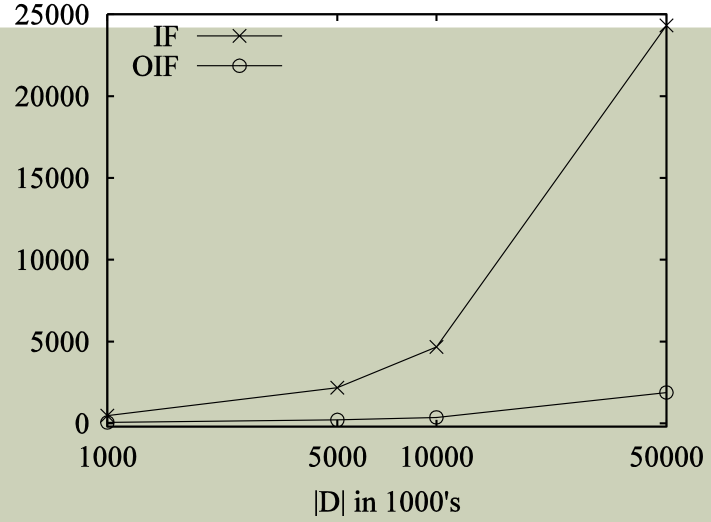


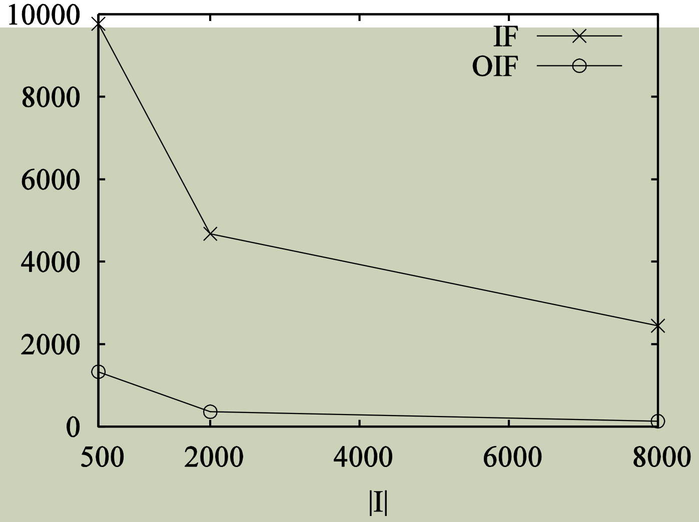


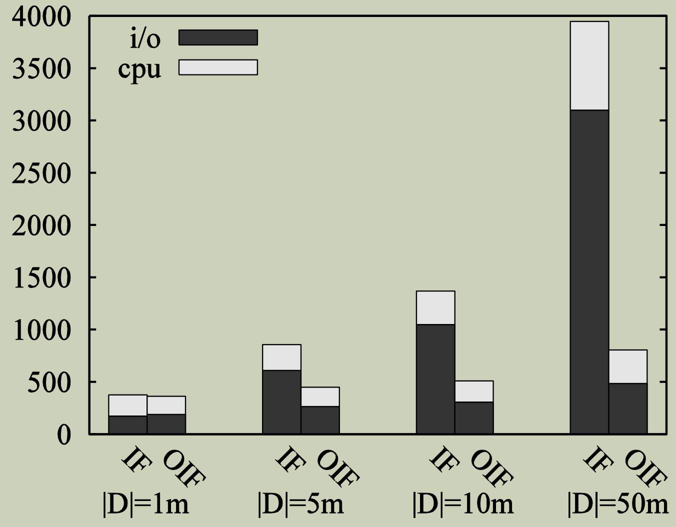


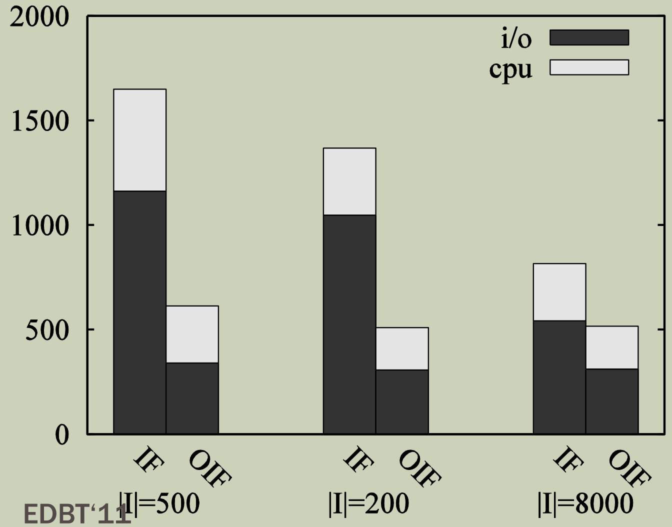
Terrovitis et. al., EDBT'11

EXPERIMENTS ON SYNTHETIC DATASETS (SUBSET)

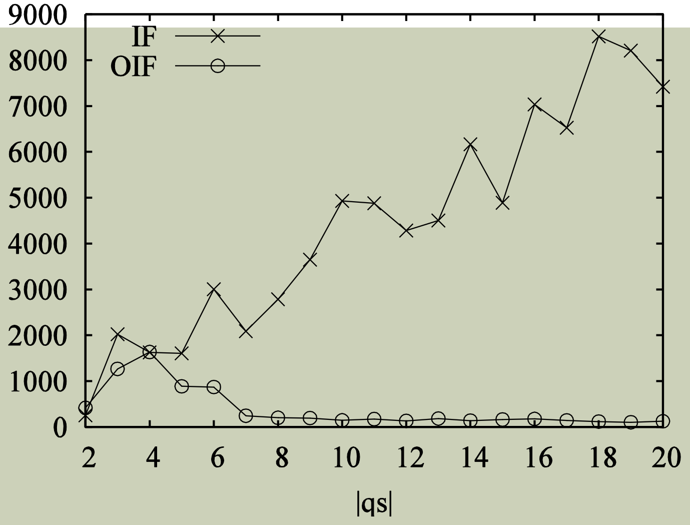


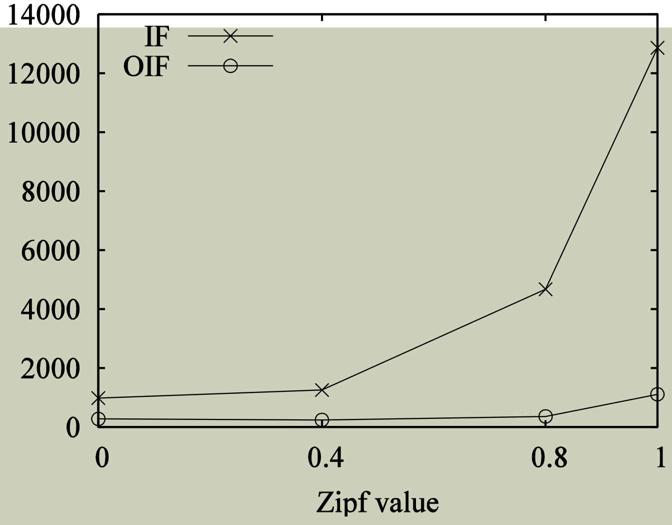


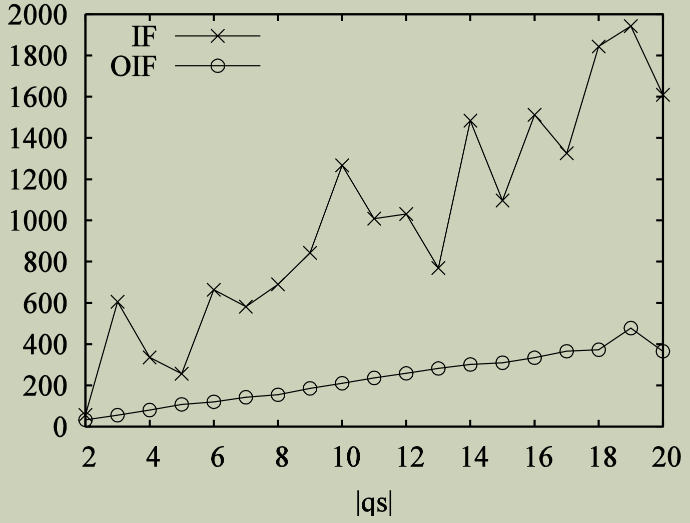


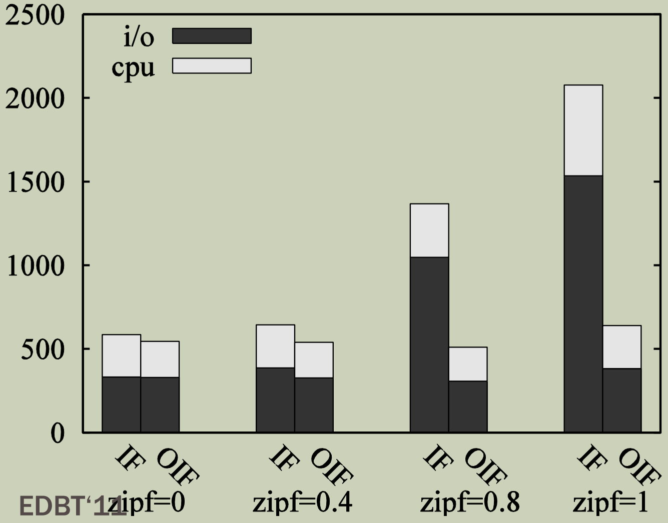


EXPERIMENTS ON SYNTHETIC DATASETS (SUBSET)

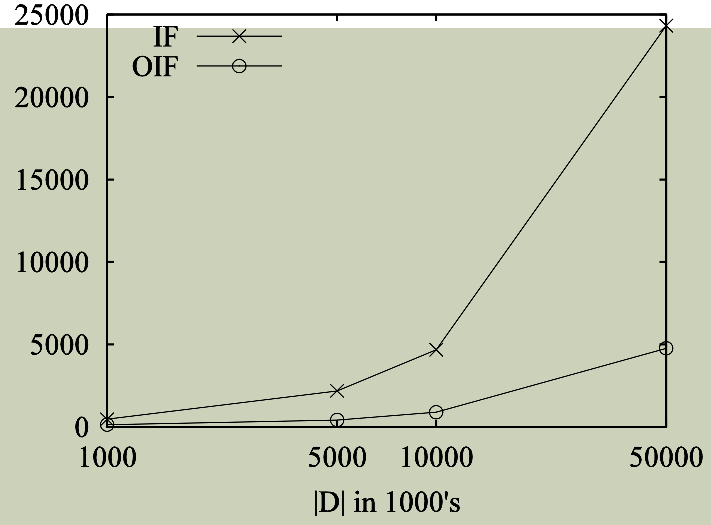


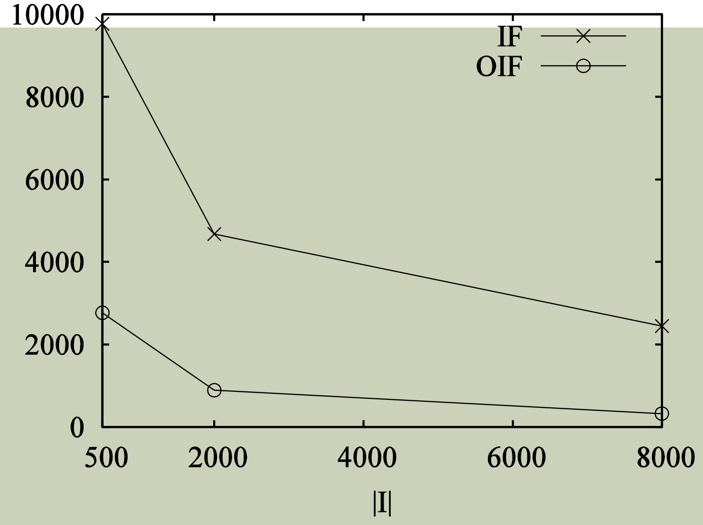


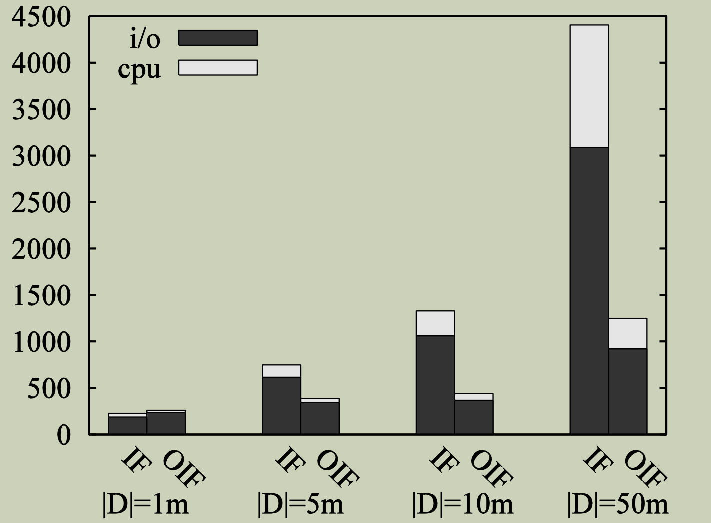


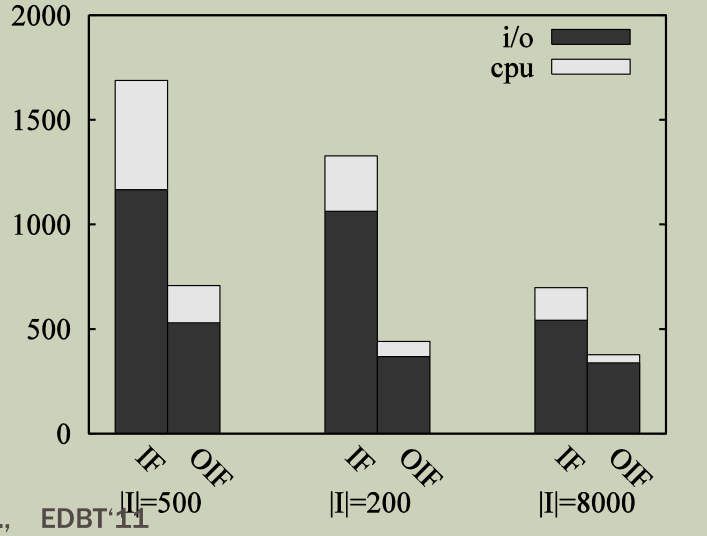


EXPERIMENTS ON SYNTHETIC DATASETS (SUPERSET)

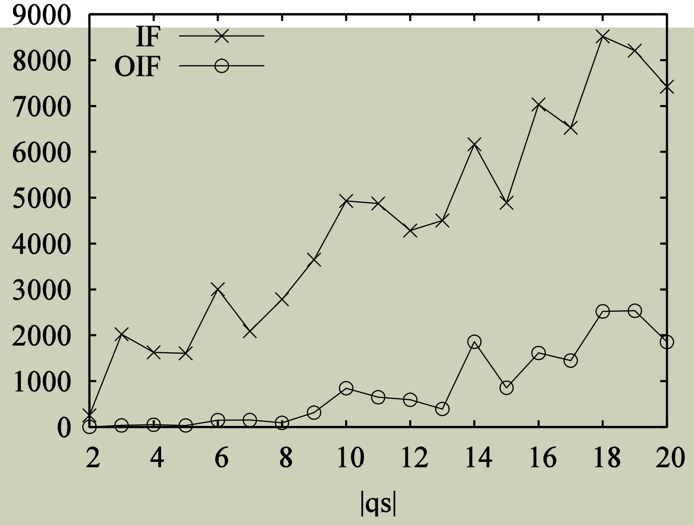


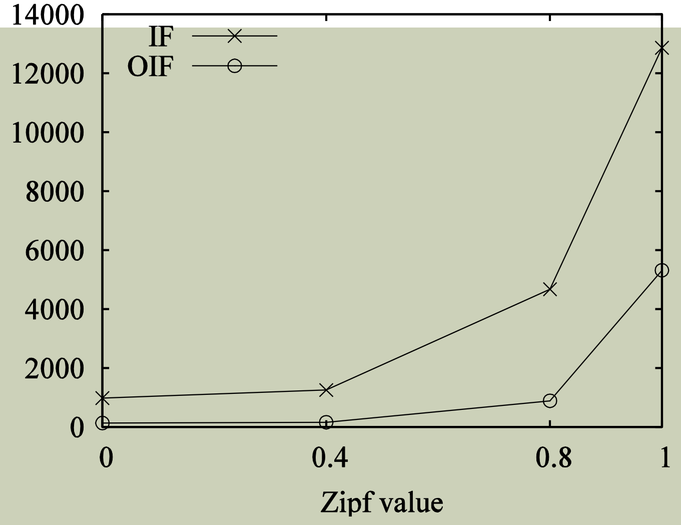


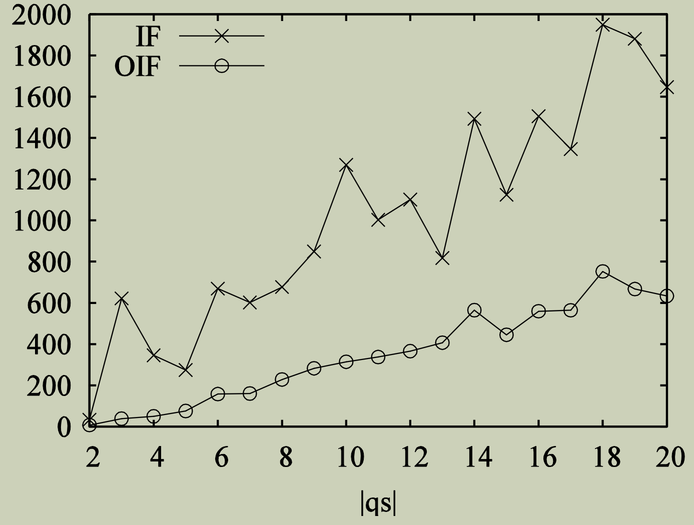


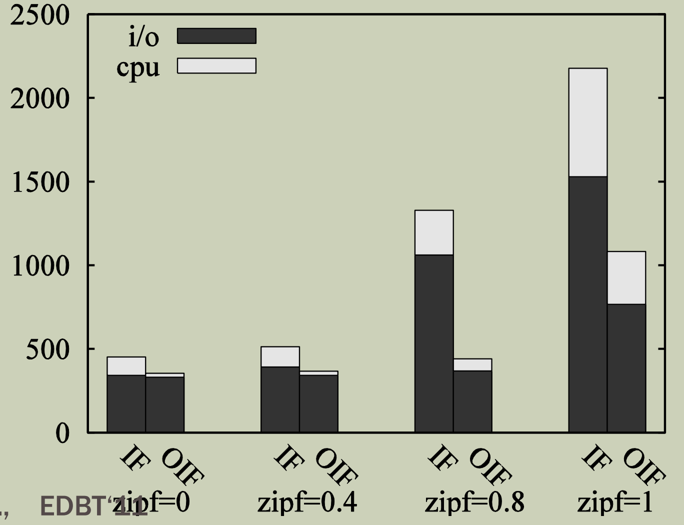


EXPERIMENTS ON SYNTHETIC DATASETS (SUPERSET)









PERFORMANCE SUMMARY

- When the ratio of database size/domain grows or when the data are skewed OIF has substantially superior performance w.r.t IF
- OIF has increased space requirements w.r.t IF
- Updates are more expensive
  - IF has an advantage when updates dominate the workload

QUESTIONS?

- Thank you!