Efficiently Discovering Recent Frequent Items in Data Streams

Nishad Manerikar
Ferry Irawan Tantono
Themis Palpanas

University of Trento, Italy

Motivation

Examples of applications

- Financial: stocks that are traded the most in a stock market
- Networking: monitoring of frequency of packets travelling between specific nodes
- E-Commerce: click-stream analysis of users for online advertising

More pertinent problem: determining the **recent** frequent items in the stream

Outline

- Problem formulation
- TiTiCount approach
- TiTiCount+ improvement
- Experimental evaluation
- Related work
- Conclusions

Problem Definition: 1

Data stream S, consisting of a stream of integers

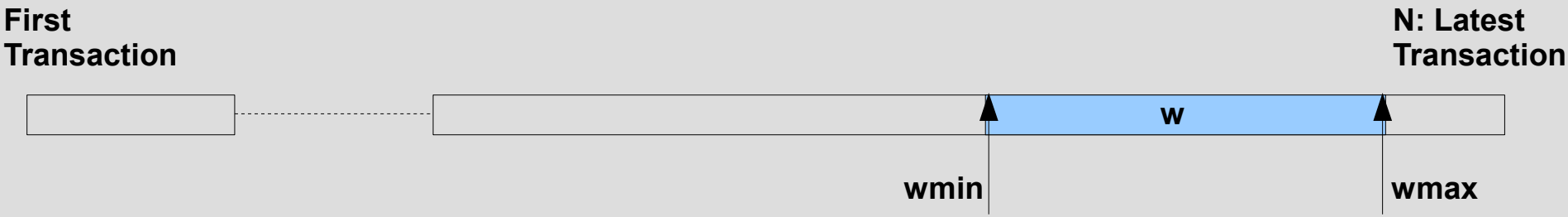
N – current length of the stream
φ – support: user-defined threshold in [0.0,1.0]

Frequent Item – an item whose frequency is at least φ.N

Problem Definition: 2

Our focus is on Recent Frequent Items

w = [wmin, wmax]: a window in the history of the stream such that, (N – wmax) << wmin



|w| = (wmax – wmin): window width

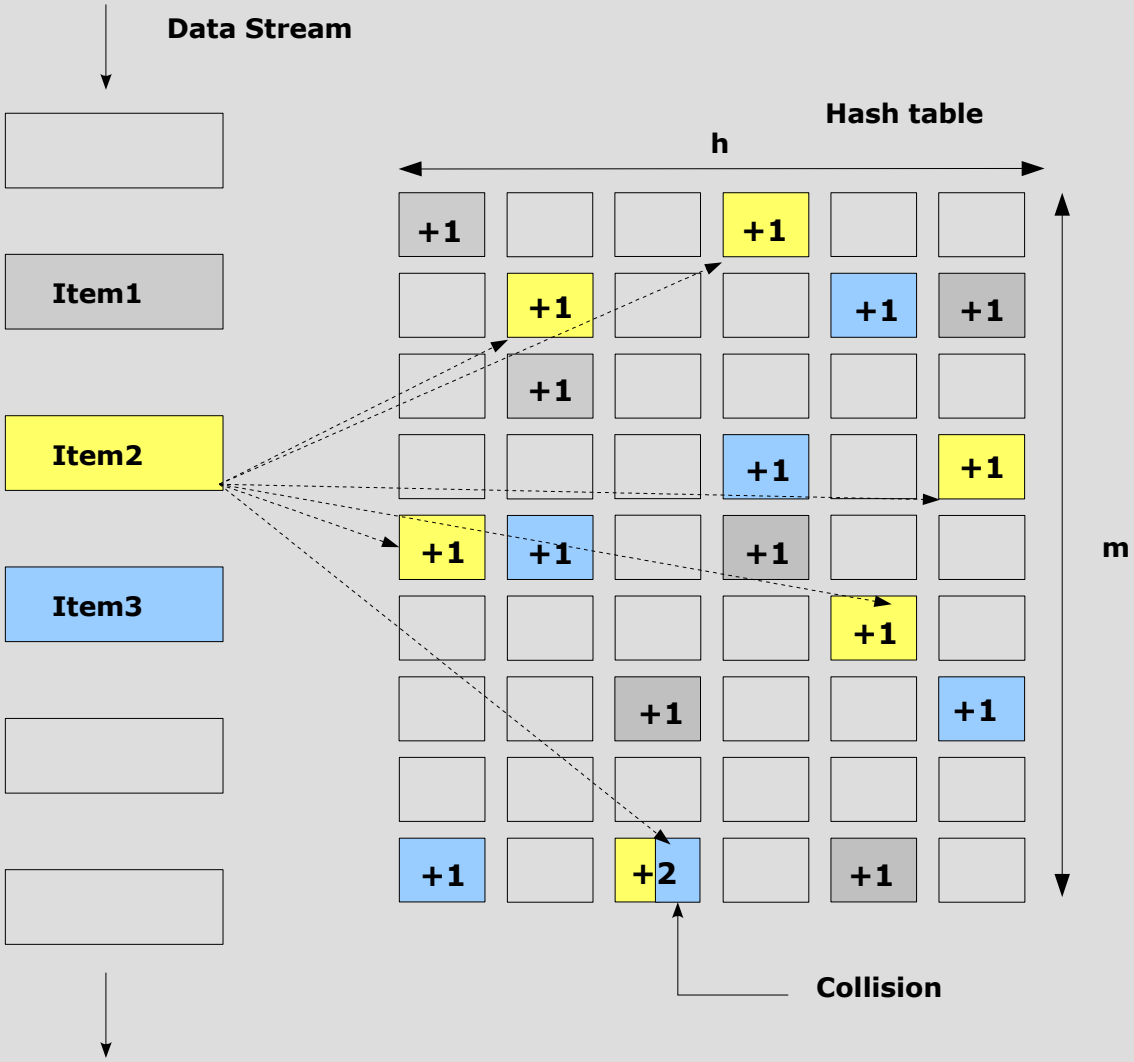
An item is frequent in w if it occurs at least φ.|w| times within w

Proposed Approach

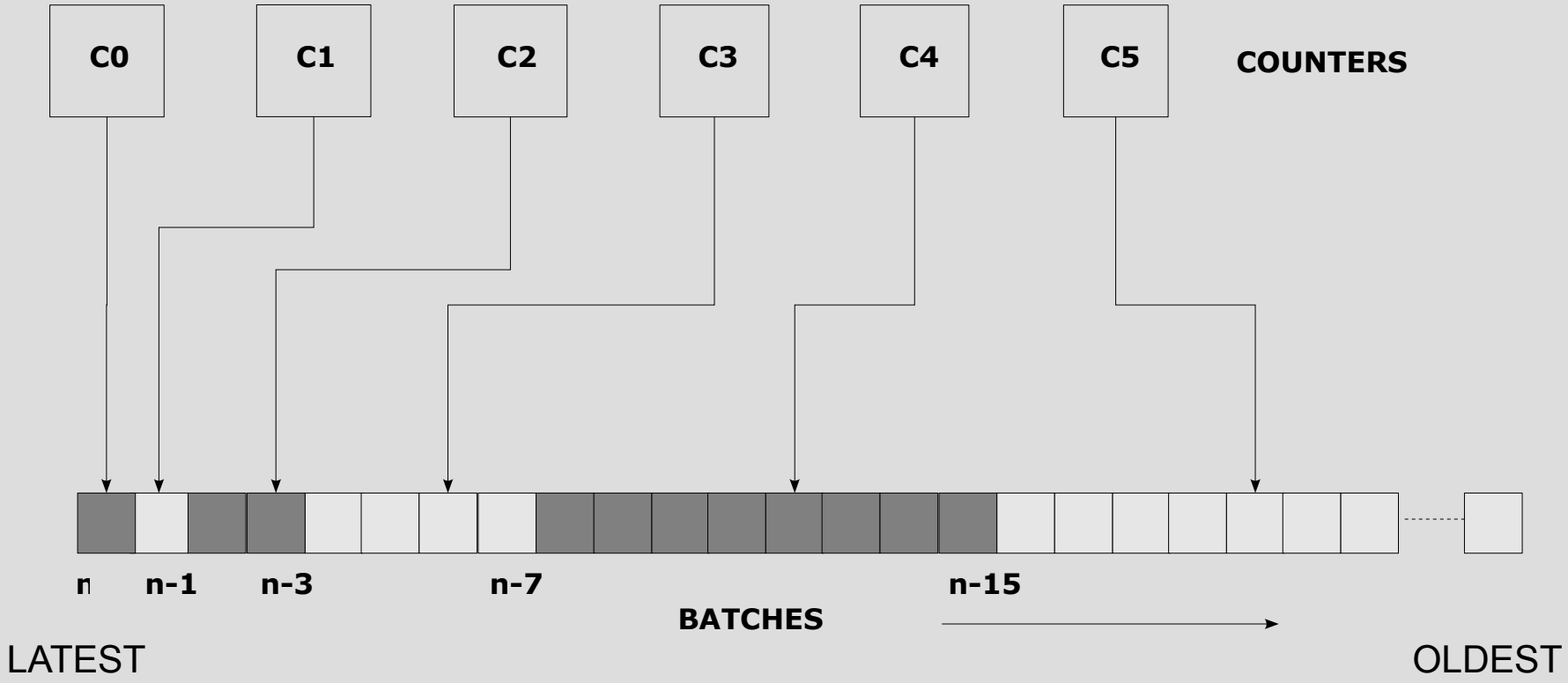
Stream Sketch + Tilted Time Windows

- Preliminary tests to evaluate existing algorithms for finding frequent items in streaming data
- We selected one of the existing sketch-based algorithms – hCount (Jin et al '03)
- Use of other sketch-based algorithms possible
- The hCount sketch is combined with an implementation of Tilted Time Windows (Gianella et al '02) to store information at different time granularities

hCount



Tilted Time Windows



Tilted Time Windows

Logarithmic Tilted Time Windows

e.g. With batch size 1000, we require just 11 windows to store 10^6 transactions

If linear windows were used, 1000 windows would be required

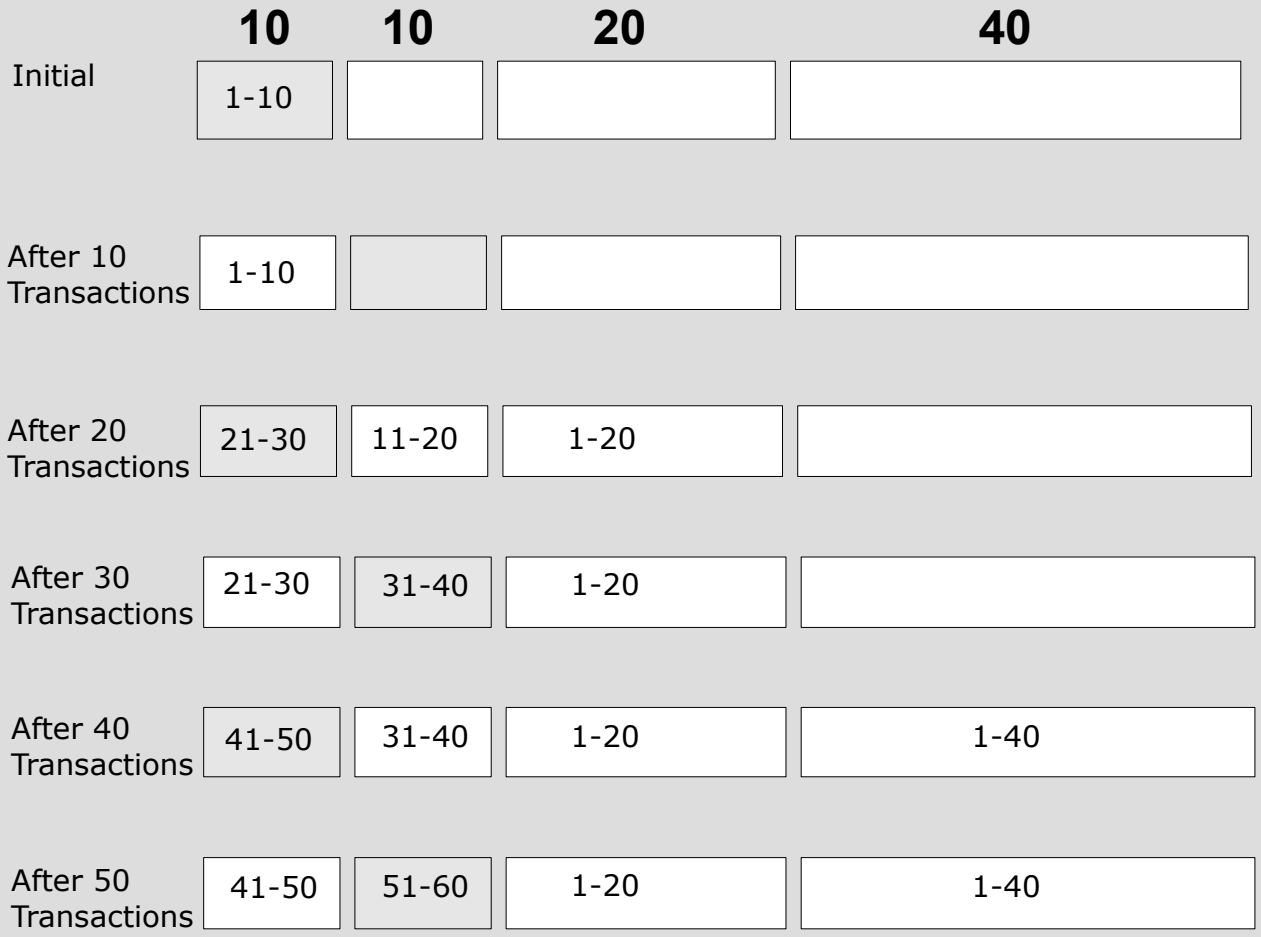
Algorithm Skeleton

```
Let  N := current transaction number
     B := batch size

UPDATE(new item Tn)
     Determine counters in hCount sketch corresponding to Tn
     Update each counter
     If (N mod B == 0) shift the tilted time windows

QUERY(item Q, wmin, wmax)
     Determine counters in hCount sketch corresponding to Q
     Determine windows which encompass [wmin, wmax]
     Determine and return frequency of Q
```

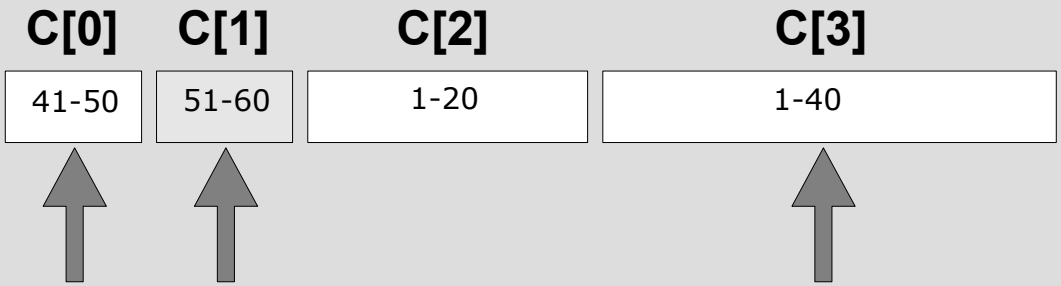
TiTiCount


10
10
20
40
Initial
1-10
After 10 Transactions
1-10
After 20 Transactions
21-30
11-20
1-20
After 30 Transactions
21-30
31-40
1-20
After 40 Transactions
41-50
31-40
1-20
1-40
After 50 Transactions
41-50
51-60
1-20
1-40

TiTiCount

Query answering

eg. Assume required window is [15, 55]

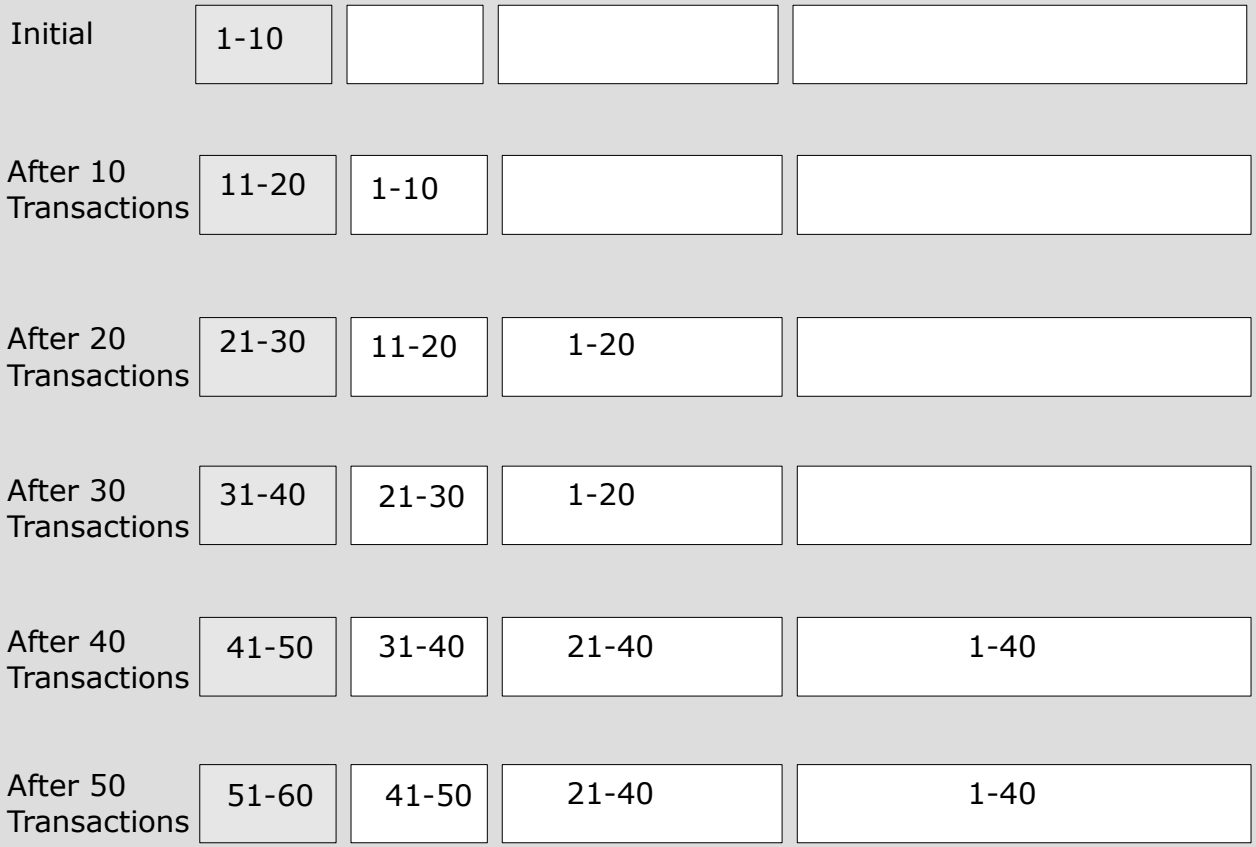


Frequency Count =

C[0] + weighted fractions of C[0] and C[3]

TiTiCount+

A different shifting mechanism to make use of redundant data while answering queries



TiTiCount+

With this method of shifting, we have the following desirable properties:

Lemma1: If two window intervals overlap, then the smaller window interval is completely contained in the larger one

Lemma2: All window intervals that overlap have one common boundary, and this common boundary is the most recent edge of these intervals.

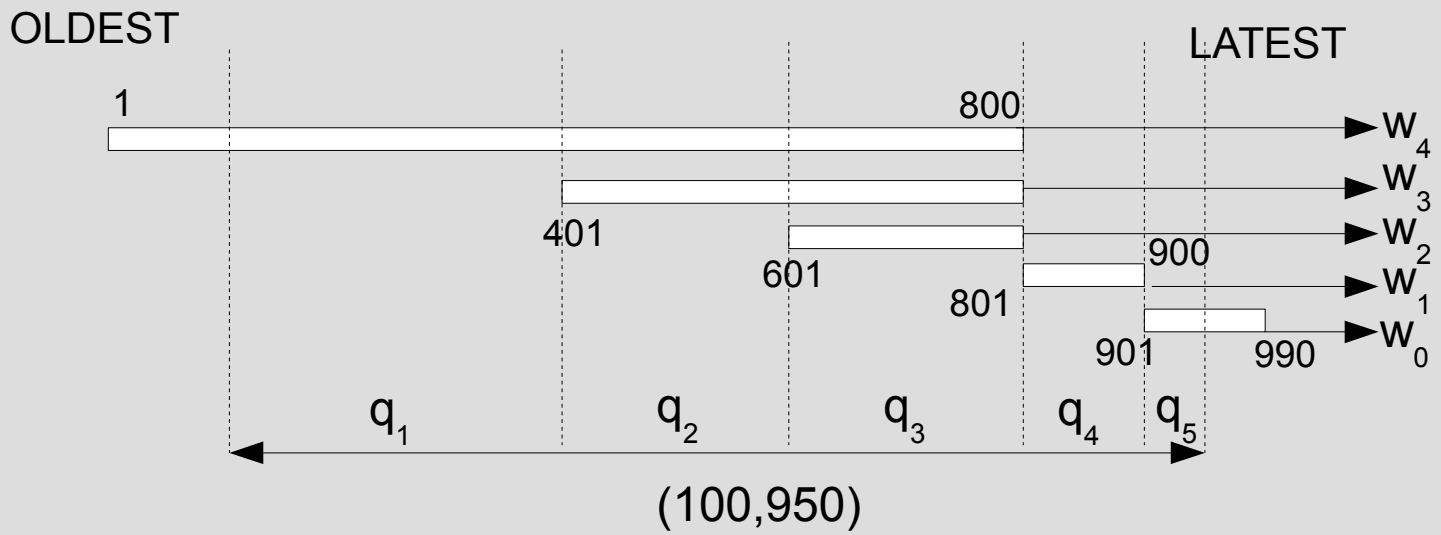
These properties are useful in answering queries...

TiTiCount+

Answering queries:

- Split into sub-queries
- Subtract contents of overlapping windows wherever applicable

Example: tilted windows after 990 transactions, batch size 100.
Query (100, 950)



Experimental Evaluation

Algorithms implemented in C, compiled using gcc under Linux Fedora Core 5; on a dual Intel Xeon 2.8 GHz machine.

Datasets used:

- Synthetic:
  - zipfian distributions
- Real:
  - kosarak
  - retail

Algorithms tested for Recall, Precision and processing Time

Test Cases

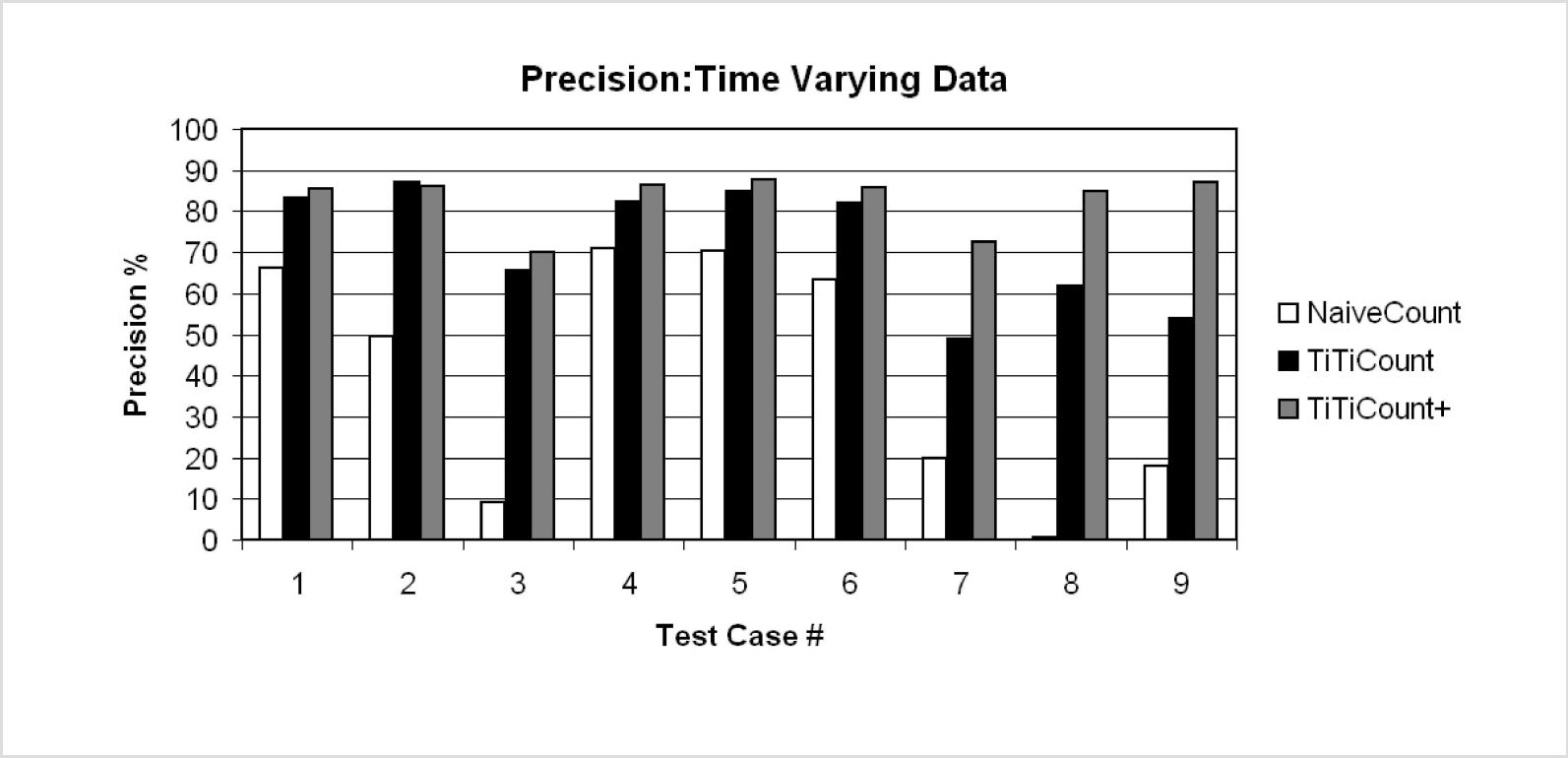
Performance over different categories of queries:

- Old/new transactions
- Small/large transaction intervals
- Varying/non-varying distributions

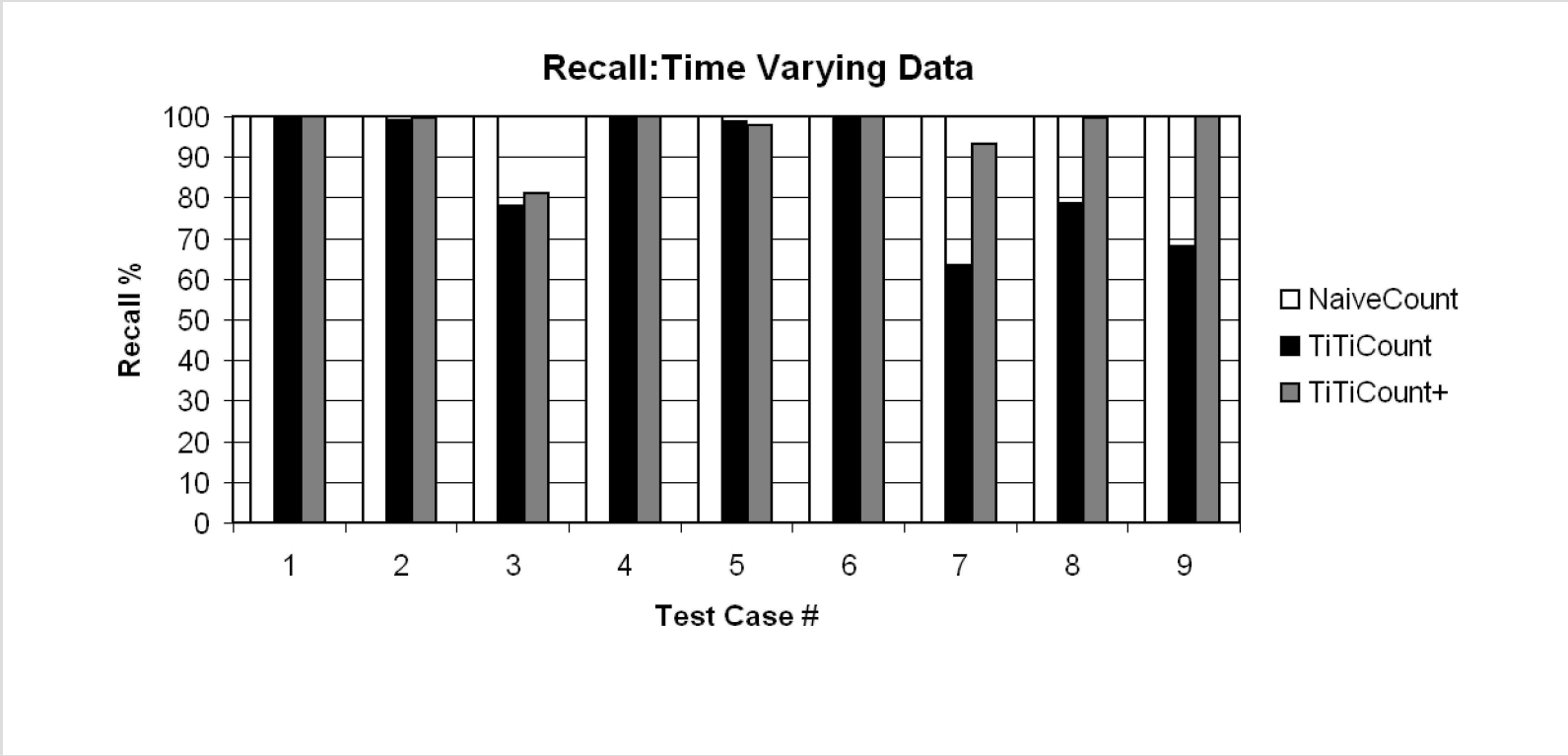
Synthetic data sets were used. Queries were posed at different points during the data stream.

N = 50000	N = 60000	N = 70000
(5000, 45000)	(5000, 55000)	(20000, 45000)
(35000, 45000)	(35000, 55000)	(40000, 55000)
(25000, 40000)	(5000, 50000)	(40000, 65000)

Test Cases - Results



Test Cases - Results



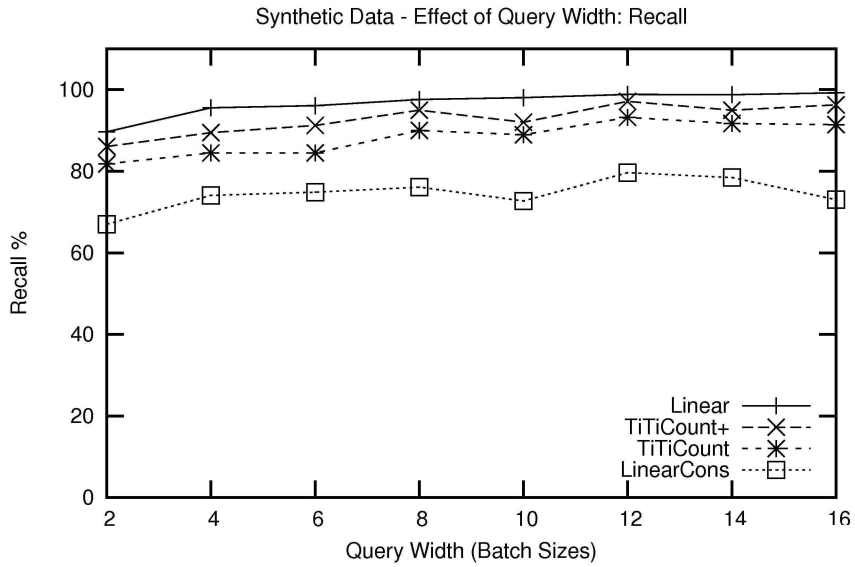
Query Width – Synthetic Data

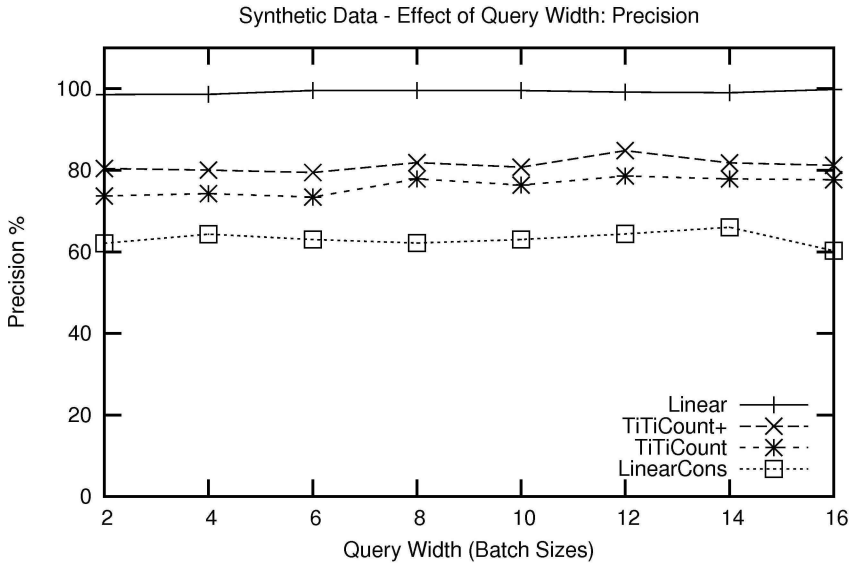
30 queries were randomly generated, after every 100,000 transactions

For comparison:

- Linear – Use linear windows, unbounded memory
- LinearCons – Use linear windows, bounded memory (compensated by increasing batch size)

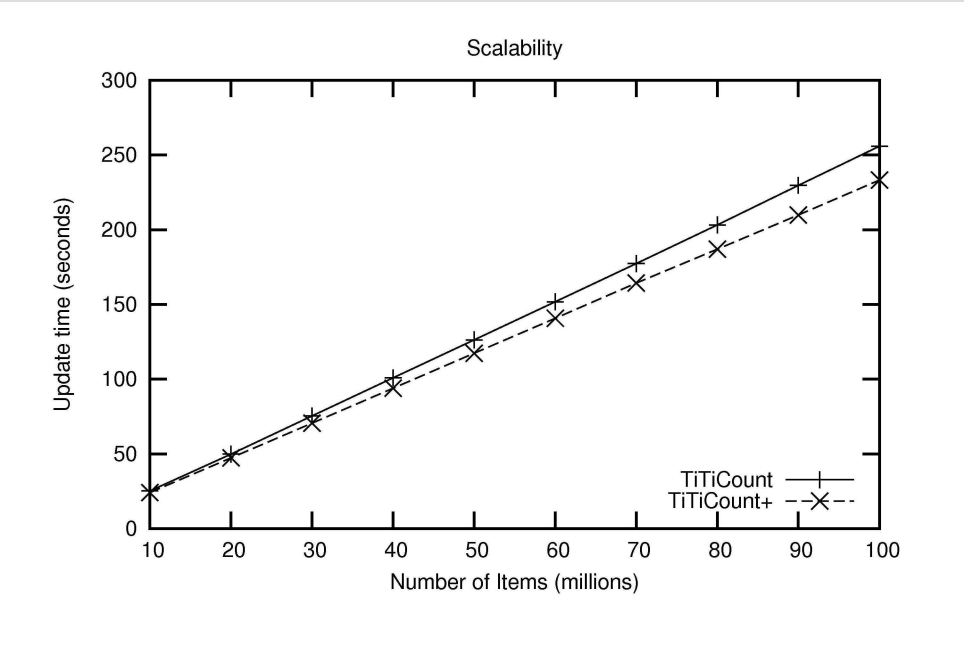
Query Width – Synthetic Data





Scalability

Synthetic data. Update time for upto 100 million items.



Related Work

Time-decaying approximations:

- Time series summarizations [WS01, BS03]
- Streaming data clustering [AHW+03]
- Data warehousing [CDH+02]

Frequent itemsets over streaming data [CY03]

Recent frequent itemsets:

- Sliding windows [LCW+05]
- Tilted time windows [GHP+02]

Summary and Conclusion

- Novel algorithm that addresses the problem of finding Recent Frequent Items in streaming data
- ad hoc queries – to find the frequency of an item in any given interval
- More than just frequent items – we have a sketch of the stream with temporal information
- Experimental validation of proposed approach using synthetic and real data
- Achieves high quality approximation using limited space and time resources

Thank You!

nd.manerikar@studenti.unitn.it