Approximate Join Processing Over Data Streams

Abhinandan Das Johannes Gehrke
Mirek Riedewald
Cornell University

SAKIRE ARSLAN

USC - 2003

Outline

- Data Stream Join Processing
- Sliding Window Join
- Approximate Join
- Error Measures
- Join Algorithms using the Proposed Error Measure
  - Static algorithm
  - Offline algorithm with Fast CPU
  - Online algorithm with Fast CPU
- Experiments and Results

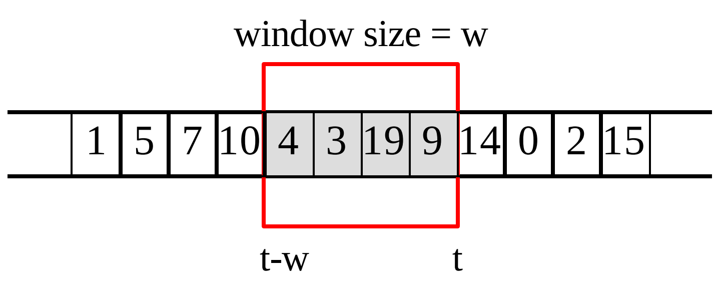
Data Stream Join Processing

- The data elements in the stream arrive online.
- The system has no control over the order in which data elements arrive to be processed.
- Once an element from a data stream has been processed it is discarded or archived
- Data streams are potentially unbounded in size.
- Performing join operation on unbounded streams has high resource requirements (both CPU and memory)

Sliding Window Join

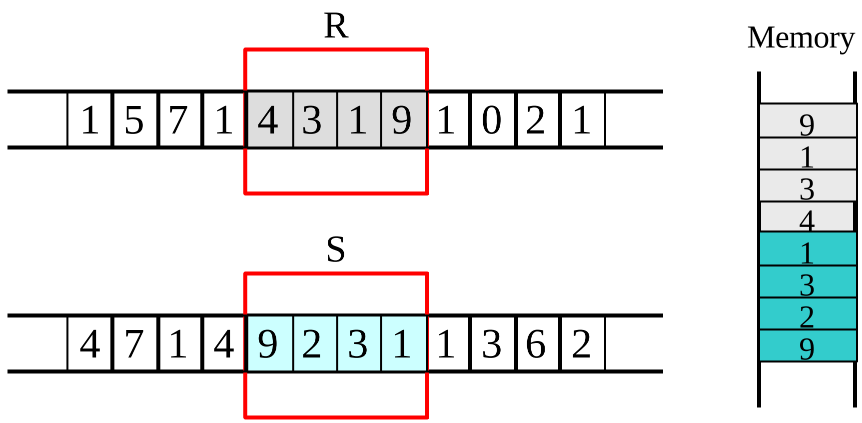
- Restrict the set of tupples that participate in the join to a bounded size window
- Window boundaries can be defined based on:
  - Time units
  - Number of tupples
  - Landmarks

– In proposed model: The window is defined in terms of time units, and at each time unit a new tupple arrives



Sliding Window Join (cont.)

- A sliding window join of window size *w*:
  - Has to store *2w* tupples
  - Has to process incoming tupples as fast as they arrive



- <u>Problem</u>: Limited resources (storage and CPU)
- <u>Solution</u>: Approximating the output

Approximating Query Answers

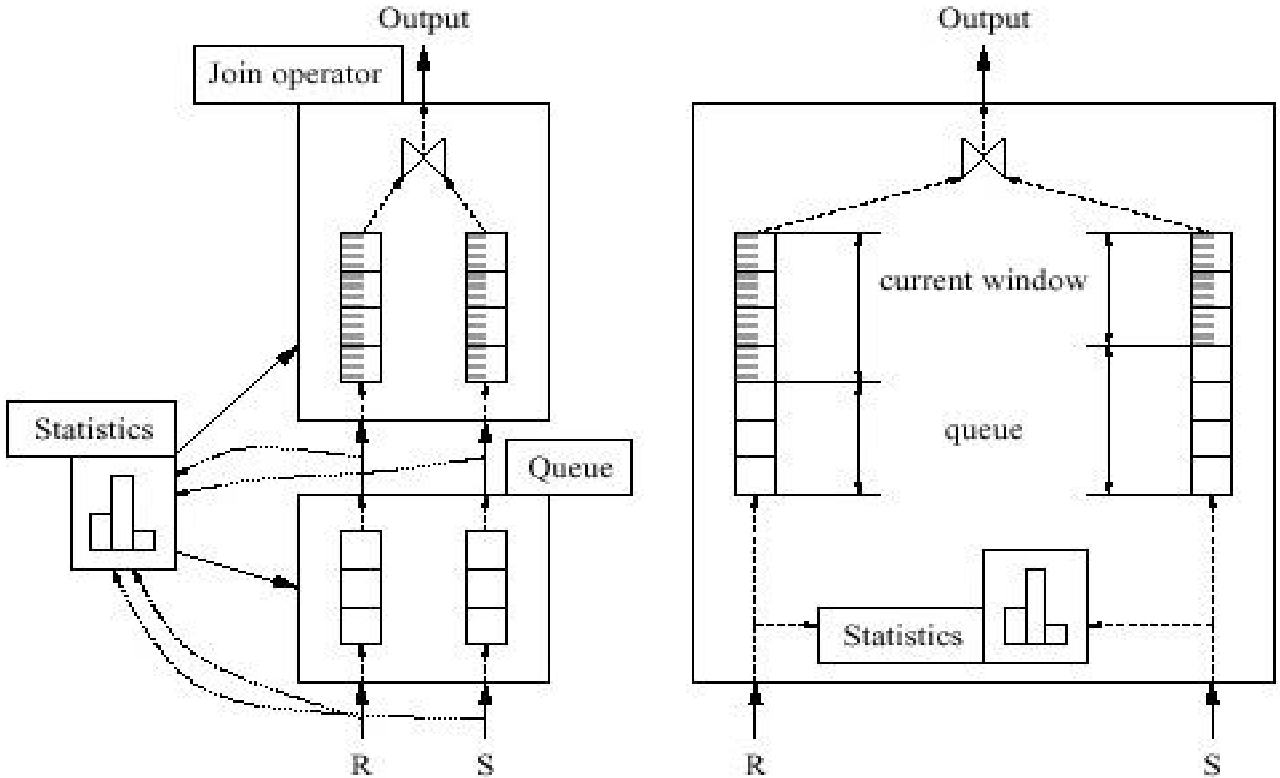
- Load Shedding : Dropping tupples before they naturally expire
  - Drop the tupples randomly
  - Assign priorities to tupples and remove the lowest priority

- **Proposed Solution**: Semantic Load Shedding

  Which tupples should be dropped when –in order to minimize the error of the output

Join Processing Models

- Modular vs. Integrated



Modular join model   Integrated join model

Join Processing Models (cont.)

- If CPU is fast:
  - Incoming tupples can be processed at least as quickly as they arrive
  - Modular and integrated models are equivalent
  - Approximation is due to memory restriction
  - Optimization Goal: Decide which tupples to drop in the join memory so that approximation error is minimized
- If CPU is slow:
  - Tupples arrive faster then they can be processed
  - Approximation is due to both memory and CPU processing constraints.
  - Optimization Goal: Select the tupples to drop in the join memory and **the queue** so that approximation error is minimized

Error Measures to Evaluate Approximation

- The output of the join operation is set a of tupples.
- For sets X & Y:
  - Symmetric Difference Measure is defined as

    $|(X\text{-}Y) \cup (Y\text{-}X)|$
- **Proposed Error Measure: MAX-subset measure**
  - MAX-subset measure represents the number of missing tupples in the approximate result set
  - It is a special case of Symmetric Difference Measure where one of the sets is a subset of the other

USC

Error Measures to Evaluate Approximation (cont.)

- **MAX-subset measure**

  X = the approximate result set

  Y = the exact result set

  $X \subseteq Y$

  symmetric difference (X,Y) = |Y-X|

  MAX-subset measure(X,Y) =|Y-X|

- If the set X maximized the error will be minimized (similarly similarity will be maximized)

USC

Error Measures to Evaluate Approximation (cont.)

Some of the set-theoretic error/similarity measures are:

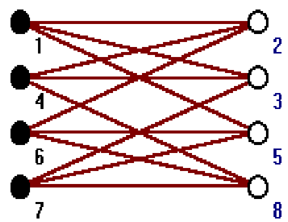
1. Matching Coefficient: $|X \cap Y|$
2. Dice Coefficient: $2 * |X \cap Y| / |X| + |Y|$
3. Jaccard Coefficient: $|X \cap Y| / |X \cup Y|$
4. Cosine Coefficient: $|X \cap Y| / |X \cup Y|^{1/2}$
5. Earth Mover's Distance
6. Matchand Compare

Join Algorithms

- Algorithm for the **Static Case**
- **Offline** window join algorithm with a **Fast CPU**
- **Online** window join algorithm with a **Fast CPU**

Bipartite Graphs

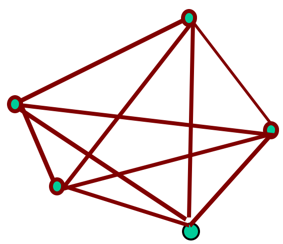
- A **bipartite graph** is a graph G whose vertex set V can be partitioned into two non empty sets V_1 and V_2 in such a way that every edge of G joins a vertex in V1 to a vertex in V2.



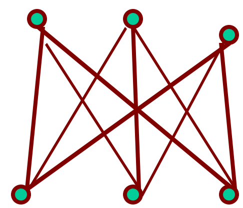
$V_1 = \{1,4,6,7\}$
$V_2 = \{2,3,5,8\}$

- **Kuratowski's theorem:** a graph is planar if and only if it does not contain a subgraph which is an expansion of K_5 (the full graph on 5 vertices) or $K_{3,3}$ (six vertices, three of which connect to each of the other three)
- **Kuratowski components** are the graphs that follow Kuratowski's theorem

K_5

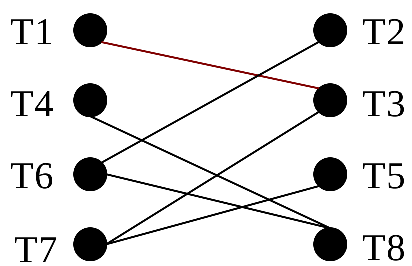


$K_{3,3}$



Static Case

- Input relations (A and B) are not data streams
- Goal is to find a set of k tupples to be dropped from the input relations such that the size of the k-truncated join result is maximized
- k-truncated join approximation problem is modeled as a graph problem:
  - The exact result set is a bipartite graph $G(V_A,V_B,E)$
    partition V_A represents tupples from A , partition V_B represents tupples from B, E represents the tupples in the join result

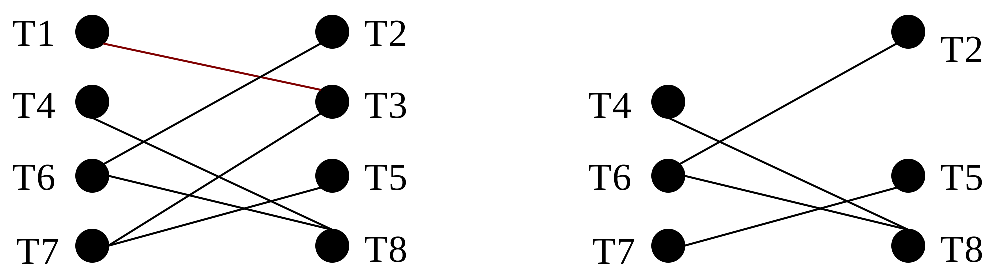


$V_A = \{T1,T4,T6,T7\}$

$V_B = \{T2,T3,T5,T8\}$

Static Case (cont.)

- G is a union of mutually disjoint fully connected bipartite components (called Kuratowski components, K(m,n) – where *m* and *n* are number of nodes from V_A and V_B)
- When we delete a node all edges incident on the node get deleted



- **New goal is**: To find a set of k nodes in the bipartite join-graph whose deletion results in the deletion of the fewest number of edges
- OR to find a set of *k* nodes to be retained, such that the subgraph has highest number of edges

Static Case (cont.)

Optimal Dynamic Programming Solution

- **Input :** A bipartite graph consisting *c* Kuratowski components $K(m_1,n_1)$, $K(m_2,n_2)$,… $K(m_c,n_c)$ and an integer *k*. $K(m_i,n_i)$, denotes i^{th} Kuratowski component
- For a component K(m,n) ***p£ m+n*** is the number of retained nodes
  - **m’** = nodes retained from m (m’ ≤ m) **n’**= nodes retained from n (n’ ≤ n)
  - **p = m’ + n’**
  - We want to maximize **m’ * n’** (the number of edges)
  - To maximize m’*n’ , **|m-n|** should be minimized.
    - If p is even m’ = n’ = p/2 and m’*n’ = $(p/2)^2$
    - if p is odd m’=(p+1)/2, n’=(p-1)/2 and m’*n’ = $(p^2-1)/4$ (m’ > n’)
  - Therefore, the max number of edges that can be retained for K(m,n) with retaining p nodes is

$$C_{m,n}(p) = \begin{cases} (p/2)^2 & \text{if } p \leq 2n,\ p \text{ even} \\ (p^2-1)/4 & \text{if } p \leq 2n,\ p \text{ odd} \\ n(p-n) & \text{else.} \end{cases}$$

Static Case (cont.)

- The max number of edges retained from all i Kuratowski components is:

  j is the number of nodes retained

  i=1
  $$T(1,j) = \begin{cases} C_{m_1,n_1}(j) & \text{if } 0 \le j \le m_1 + n_1 \\ -\infty & \text{if } j > m_1 + n_1 \end{cases}$$

  i > 1
  $$T(i,j) = \max \begin{cases} T(i-1,j), \\ T(i-1,j-1) + C_{m_i,n_i}(1), \\ T(i-1,j-2) + C_{m_i,n_i}(2), \\ \vdots \\ T(i-1,j-m_i-n_i) + C_{m_i,n_i}(m_i+n_i) \end{cases}$$

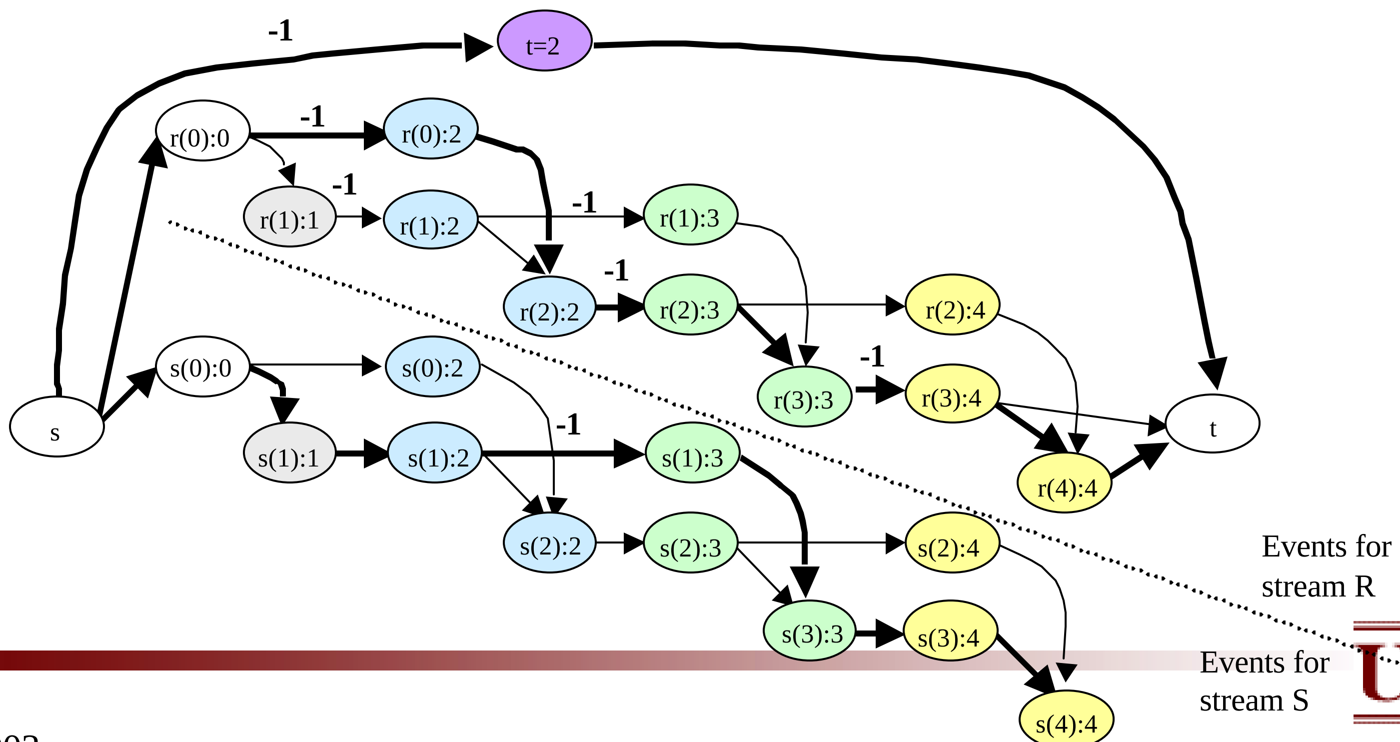
- **Final Output:** T(c,k)
- **Complexity:** $O(c.k^2)$
- If the the join operation has **m** input relations then static join load shedding algorithm will be NP-hard (m>2)

Offline, With a Fast CPU

- Input relations (R and S) are infinite data streams
- Based on sliding window join with a fast CPU and small memory
- All tupples that will arrive in future are already known to the algorithm
- Some tupples are dropped because of memory restriction
- Goal is to minimize the MAX-subset error in the approximation

Offline, With a Fast CPU (cont.)

- Approximation problem is modeled as a flow graph:
  - Nodes correspond to the tupples in memory
  - Node label *x(i) : j* means the tupple arrived at time *i* in stream X is in memory at time *j*
  - Arcs show all possible combinations of keeping or dropping tupples
  - Horizontal lines represents that a tupple survives in memory, non-horizontal line indicates , the tupple can be replaced by the newly arriving tupple
  - An arc has cost factor –1 if a result tupple produce in the transition. For all other arcs cost factor is 0
  - S is the source node and t is the sink node



Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams R=1,1,1,3,2 S=2,3,1,1,3
- Join memory M=2. Memory is shared between R and S equally
- w=3 , tupples are dropped after 3 time units
- Horizontal lines represents that a tupple survives in memory, non-horizontal line indicates , the tupple can be replaced by the newly arriving tupple

Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams R=1,1,1,3,2 S=2,3,1,1,3
- Join memory M=2. Memory is shared between R and S equally
- w=3 , tupples are dropped after 3 time units
- Horizontal lines represents that a tupple survives in memory, non-horizontal line indicates , the tupple can be replaced by the newly arriving tupple

R=1,1,1,3,2

S=2,3,1,1,3

t =0,1,2,3,4

Window contents:
r(0) : 1 s(0) : 2

Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams R=1,1,1,3,2   S=2,3,1,1,3
- Join memory M=2. Memory is shared between R and S equally
- w=3 , tupples are dropped after 3 time units
- Horizontal lines represents that a tupple survives in memory, non-horizontal line indicates , the tupple can be replaced by the newly arriving tupple

R=1,1 1,3,2

S=2,3,1,1,3

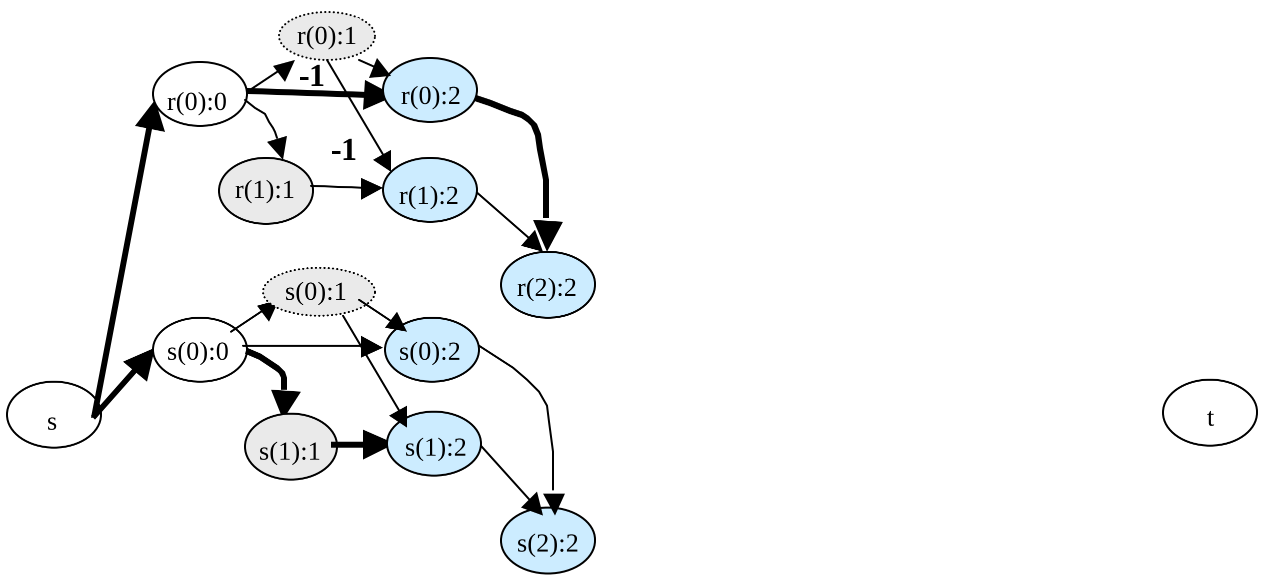
t =0,1,2,3,4

Window contents:
r(0) : 1 s(1) : 3
r(1) : 1 s(0) : 2
r(1) : 1 s(1) : 3

Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams R=1,1,1,3,2   S=2,3,1,1,3
- Join memory M=2. Memory is shared between R and S equally
- w=3 , tupples are dropped after 3 time units
- Horizontal lines represents that a tupple survives in memory, non-horizontal line indicates , the tupple can be replaced by the newly arriving tupple



R=1,1,1,3,2

S=2,3,1,1,3

t =0,1,2,3,4

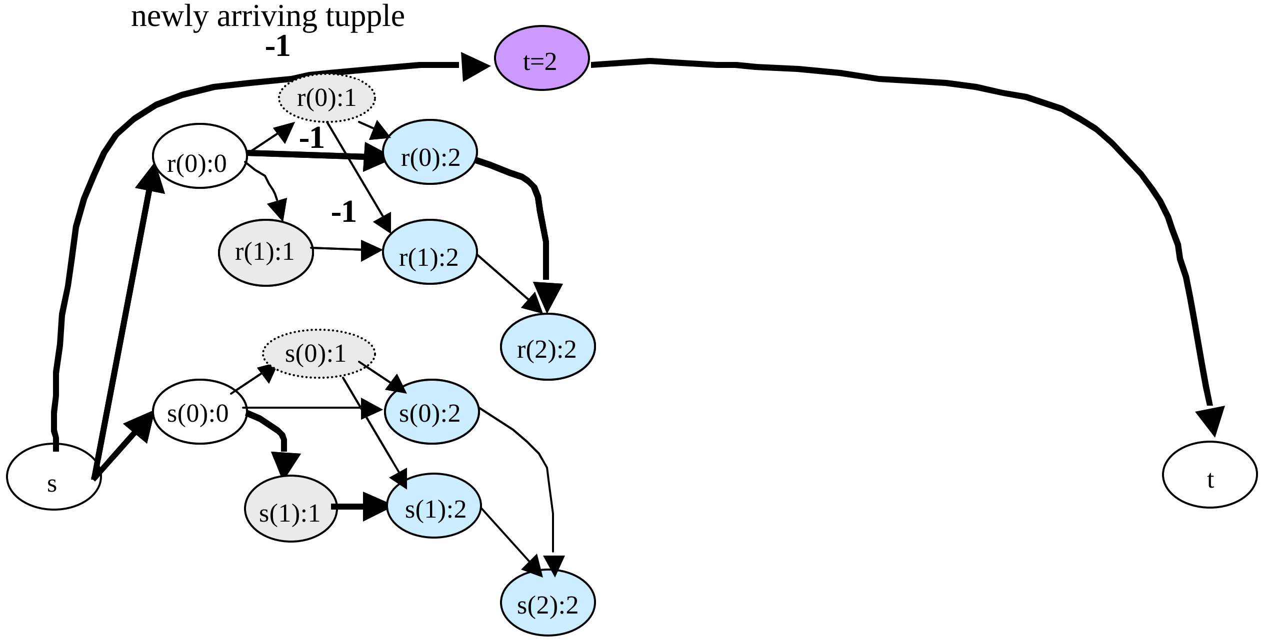
Window contents:

r(0) : 1 s(2) : 1
r(1) : 1 s(2) : 1
r(2) : 1 s(0) : 2
r(2) : 1 s(1) : 3
r(2) : 1 s(2) : 1

Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams R=1,1,1,3,2 S=2,3,1,1,3
- Join memory M=2. Memory is shared between R and S equally
- w=3 , tupples are dropped after 3 time units
- Horizontal lines represents that a tupple survives in memory, non-horizontal line indicates , the tupple can be replaced by the newly arriving tupple



R=1,1,1,3,2

S=2,3,1,1,3

t =0,1,2,3,4

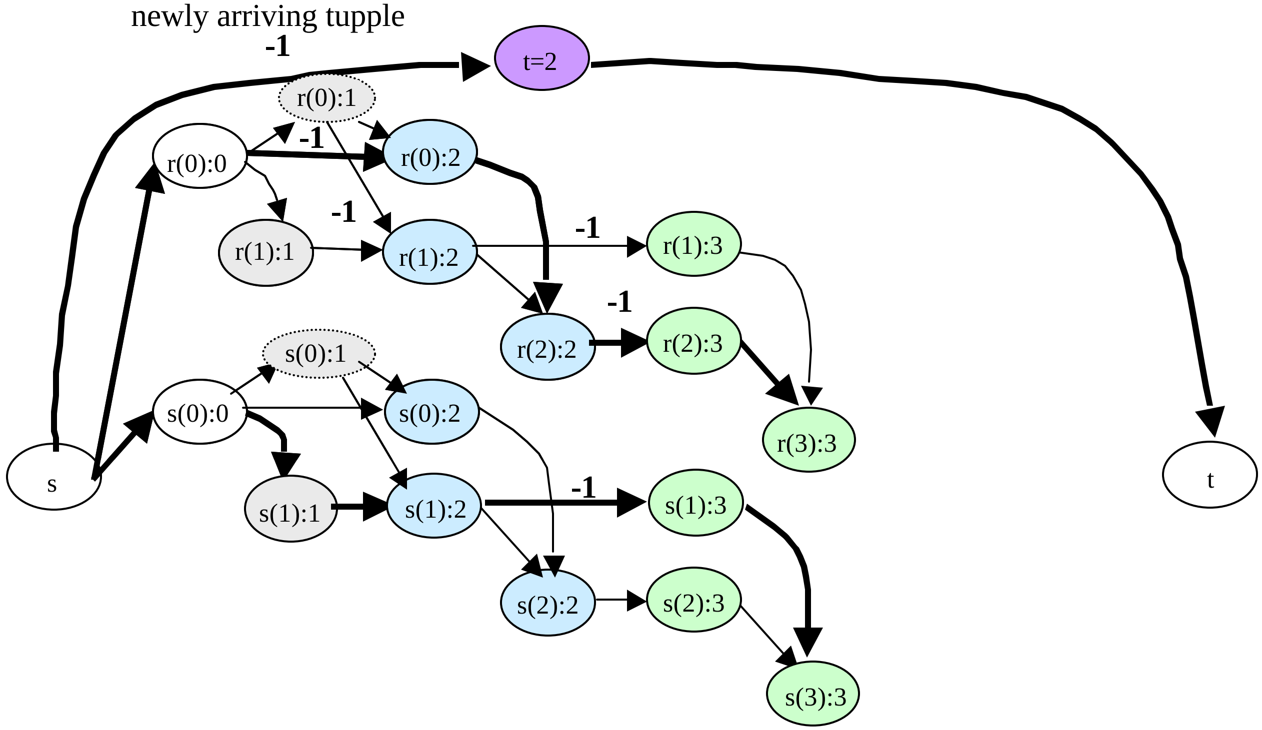
Window contents:

r(0) : 1 s(2) : 1
r(1) : 1 s(2) : 1
r(2) : 1 s(0) : 2
r(2) : 1 s(1) : 3
r(2) : 1 s(2) : 1

Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams R=1,1,1,3,2 S=2,3,1,1,3
- Join memory M=2. Memory is shared between R and S equally
- w=3 , tupples are dropped after 3 time units
- Horizontal lines represents that a tupple survives in memory, non-horizontal line indicates , the tupple can be replaced by the newly arriving tupple



R=1,1 1,3,2

S=2,3,1,1,3

t =0,1,2,3,4

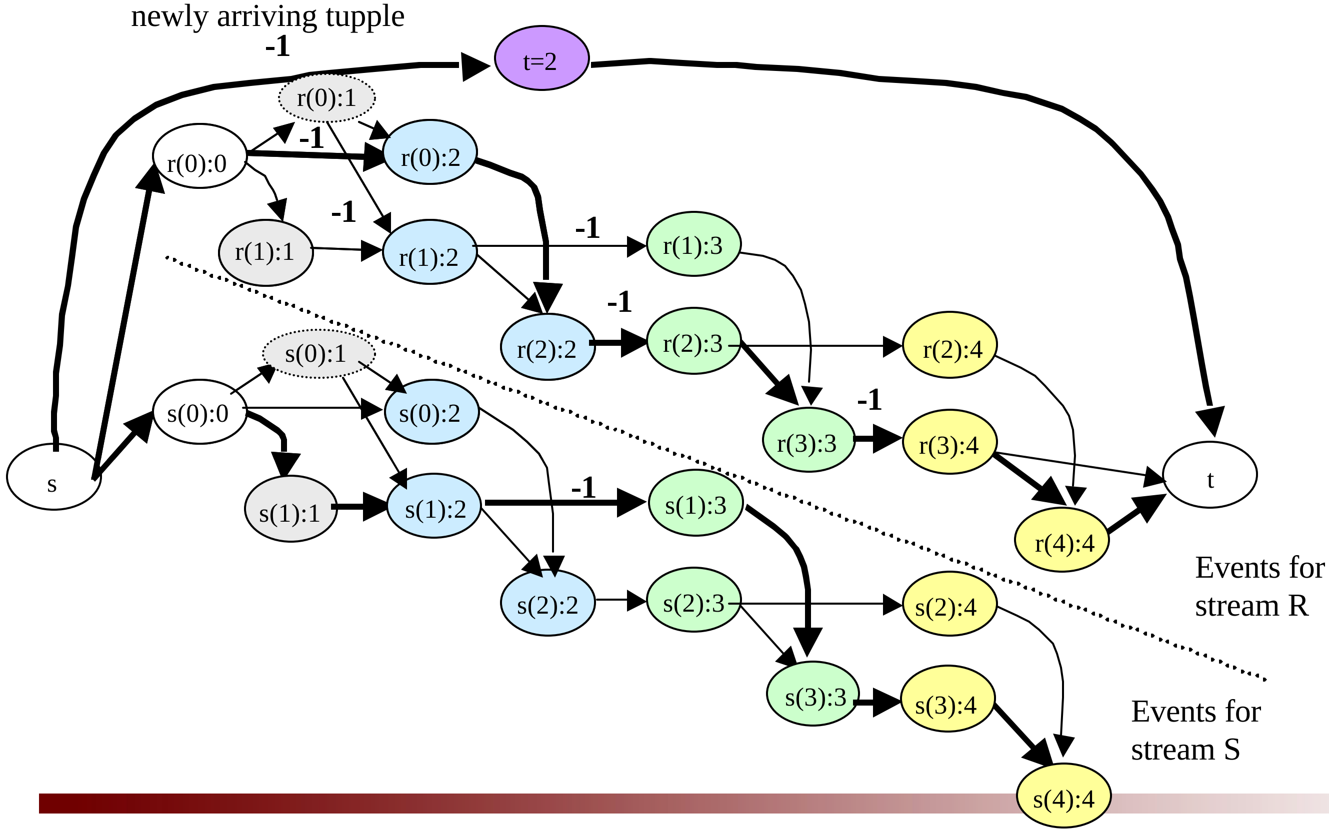
Window contents:

r(1) : 1 s(3) : 1
r(2) : 1 s(3) : 1
r(3) : 3 s(1) : 3
r(3) : 3 s(2) : 1
r(3) : 3 s(3) : 1

Offline, With a Fast CPU (cont.)

Graph Construction Example:

- Input streams R=1,1,1,3,2 S=2,3,1,1,3
- Join memory M=2. Memory is shared between R and S equally
- w=3 , tupples are dropped after 3 time units
- Horizontal lines represents that a tupple survives in memory, non-horizontal line indicates , the tupple can be replaced by the newly arriving tupple



R=1,1 1,3,2

S=2,3,1,1,3

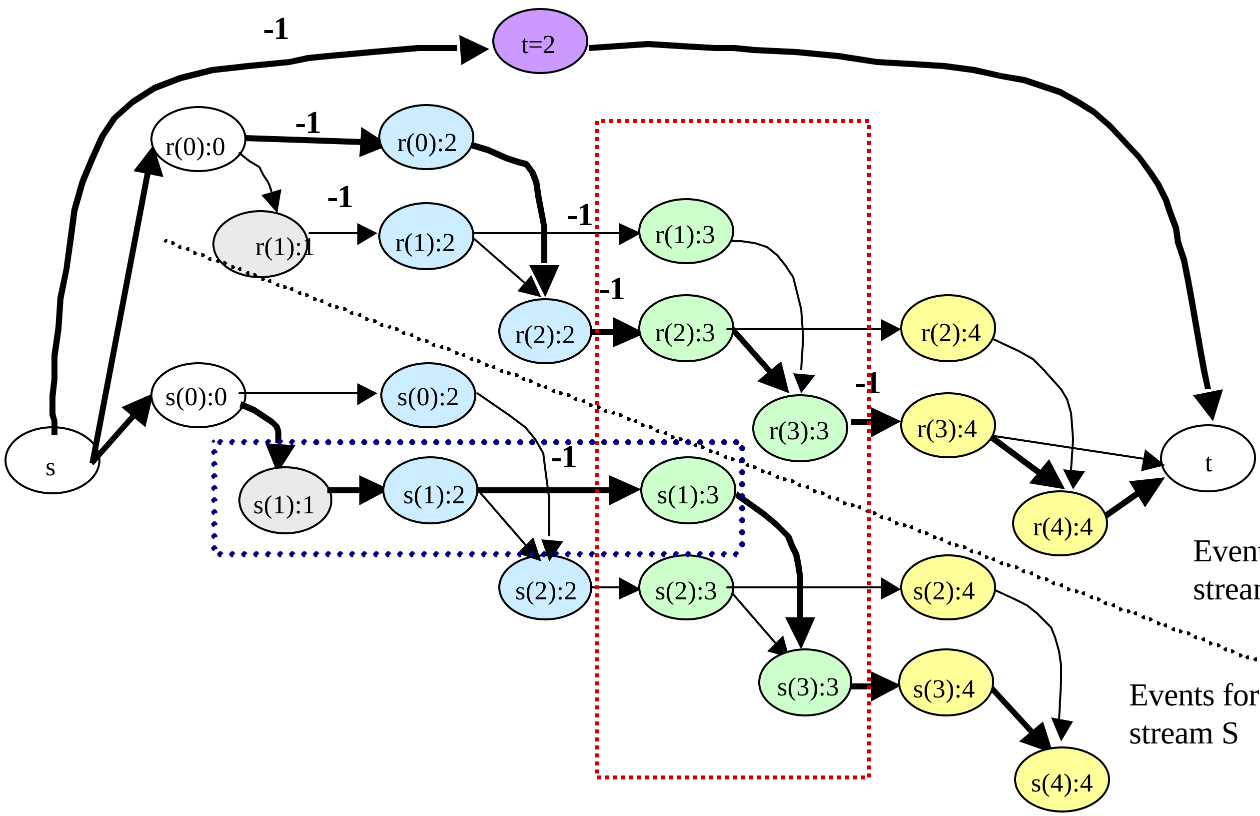
t =0,1,2,3,4

Window contents:

r(2) : 1 s(4) : 3
r(3) : 3 s(4) : 3
r(4) : 2 s(2) : 1
r(4) : 2 s(3) : 1
r(4) : 2 s(4) : 3

Offline, With a Fast CPU (cont.)

- The goal is to find the optimal flow which which produces most output tupples. In the graph optimal flow is the path with the min cost.



Optimal Solution:

<u>5 output tupples</u>

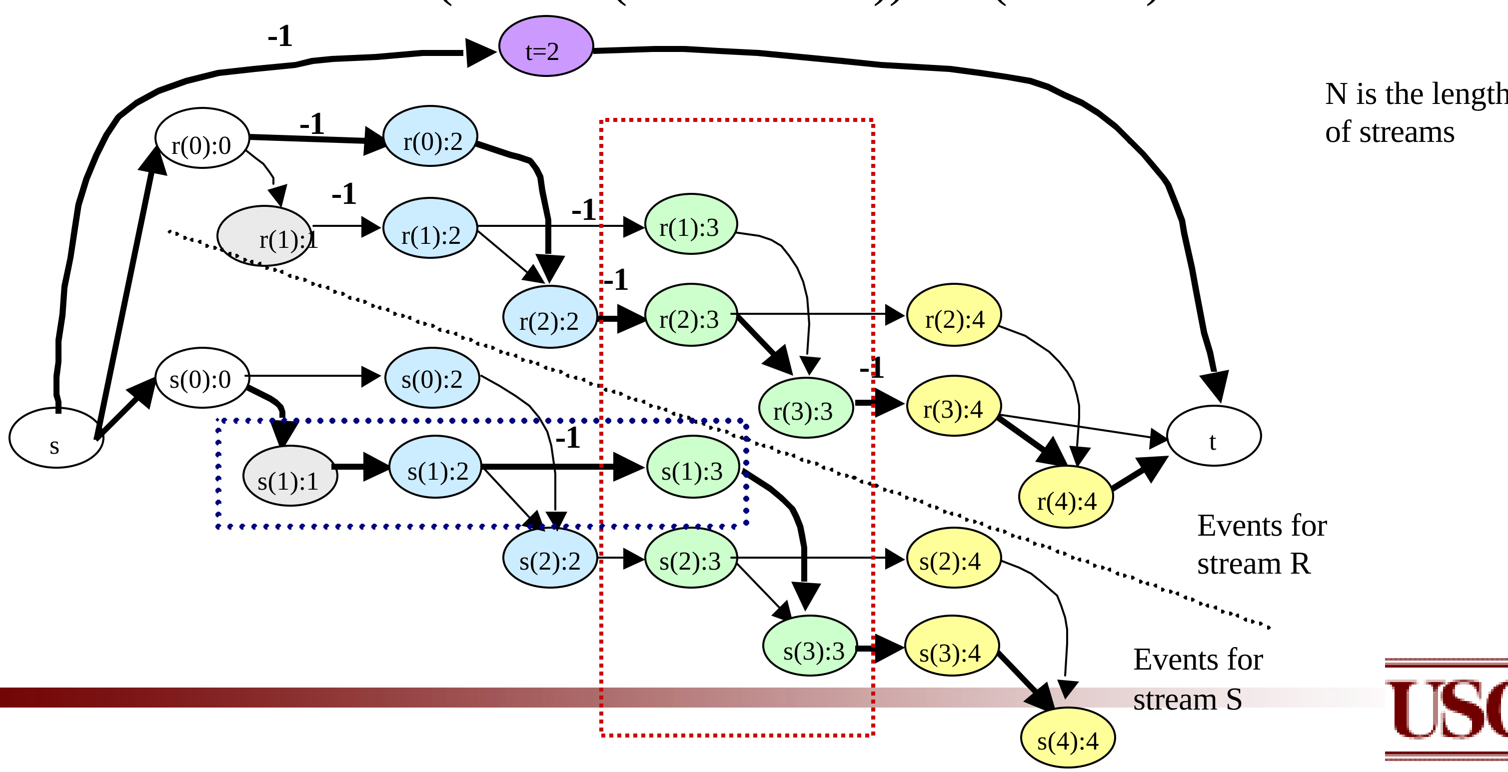
(r(0),s(2)) at time t=2
(r(2),s(2)) at time t=2
(r(2),s(3)) at time t=3
(r(3),s(1)) at time t=3
(r(3),s(4)) at time t=4

2 tupples are missed because of the approximation:

(r(1),s(2)) at time t=2
(r(1),s(3)) at time t=3

Offline, With a Fast CPU (cont.)

- Complexity for finding the minimum cost flow is $O(n^2 m \log n)$ where m is the number of arcs and n is the number of nodes
- Number of nodes and arcs can be bounded to reduce the complexity
  - There are at most $2wN + N + 2 = \theta(wN)$ nodes
  - There are at most (M+1+3.(numNodes-2)) = O(wN+M) arcs



Online, With a Fast CPU

- Online algorithm does not know which tupples will arrive in future
- Goal is to maximize the expected output size by assuming arrival probabilities for future tupples
- It estimates an arrival probability for each value in the domain of the join attribute.
- Two heuristics are defined to estimate prioroties:
- <u>PROB Heuristic</u>
  - A tupple's priority is equal to the arrival probability of it's join attribute in the other stream

    For example, for the tupple r(i) the priority is $p_S(r(i))$

Online, With a Fast CPU (cont.)

- <u>LIFE Heuristic</u>
  - It also estimates probabilities, but it favors age of the tupple to partner arrival probabilities

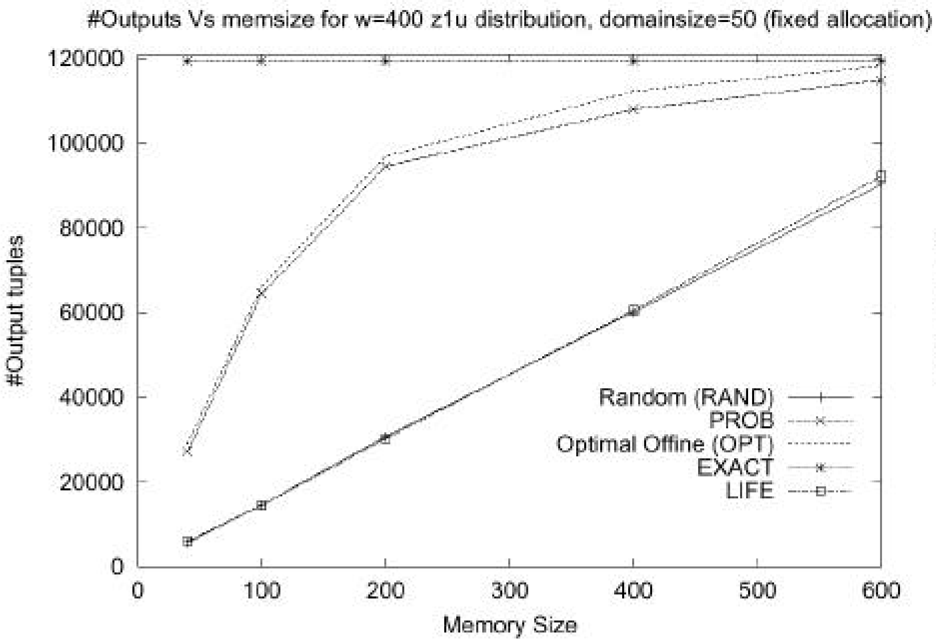
    For example, for the tupple r(i) with remaining lifetime t the priority is t*p_S(r(i))

- Example: For streams R and S ,
  - if p_S(3)=0.5, PROB priority for r(i)=3 is 0.5
  - and if remaining lifetime for r(i) is 3, LIFE priority is 1.5

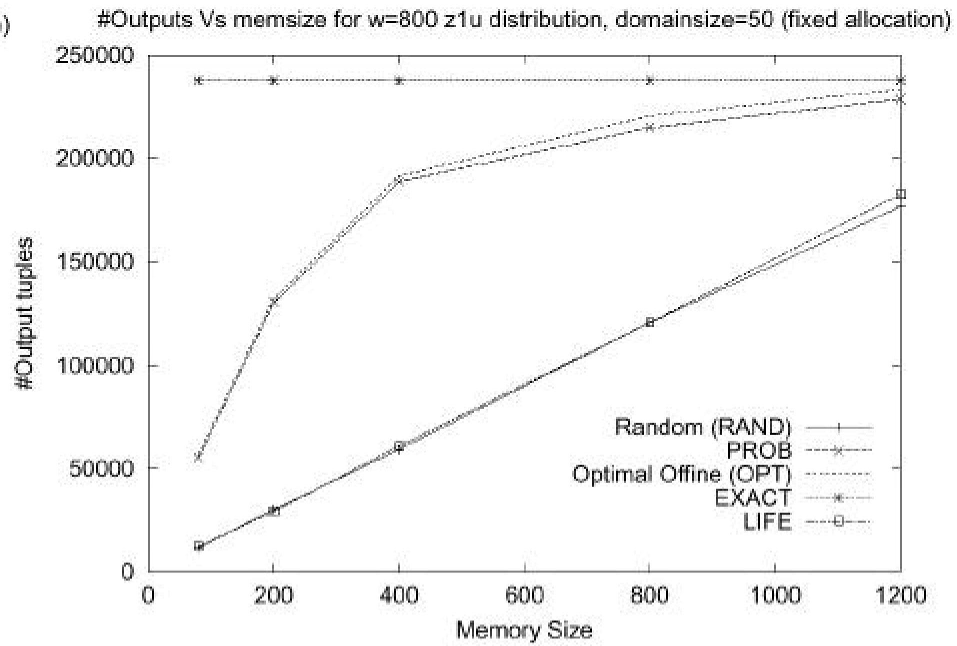
Experiments

- The performances of the following techniques are compared:
  - RAND : tupples are dropped randomly
  - OPT-offline : offline approach with fast CPU
  - PROB : online approach using PROB heuristic
  - LIFE : online approach using LIFE heuristic
  - EXACT : exact sliding window join with M=2*w*
- The length of the input streams are at most 5600 tupples.
- Experiments are done with both real datasets and synthetic dataset

Effect of Window Size



Window size 400



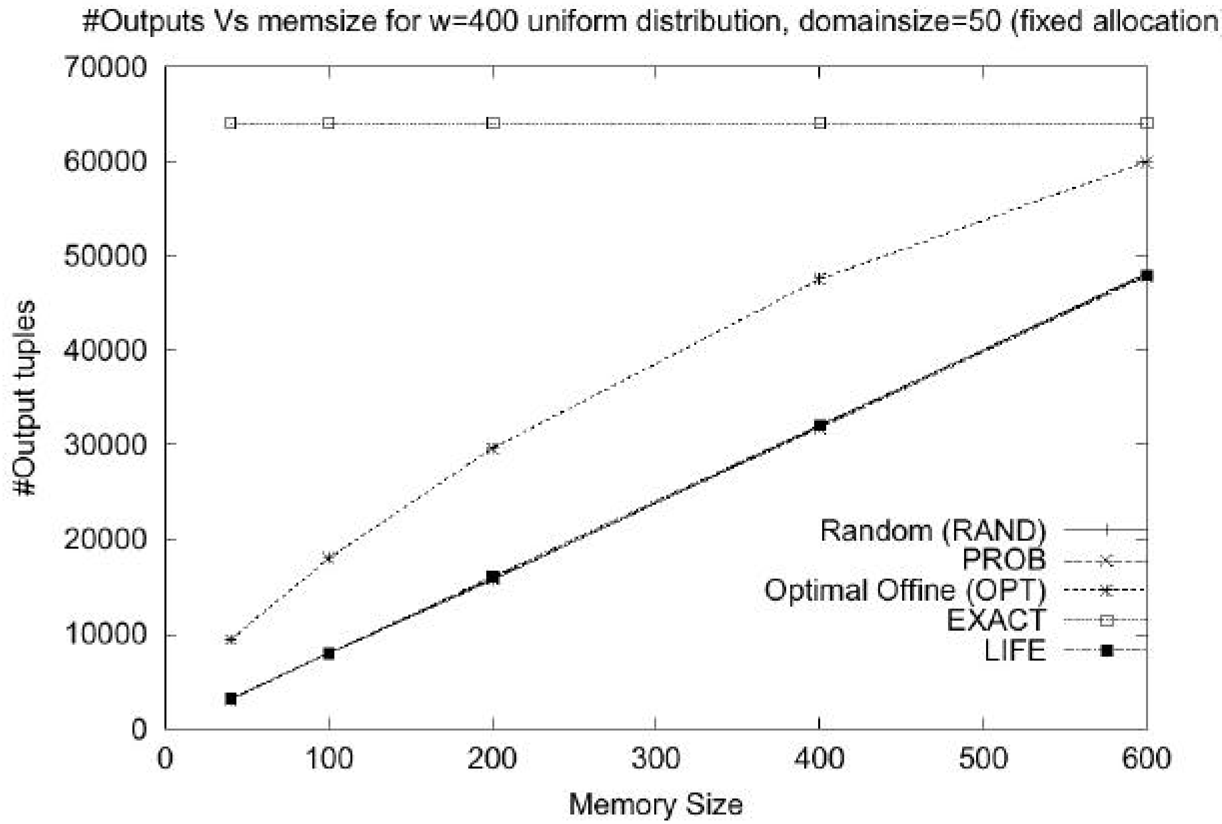
Window size 800

- The behavior of algorithms RAND, PROB, OPT and LIFE is similar for different window sizes

Effect of Data Pattern

1. Join Attribute Values are Uniformly Distributed
2. Join Attribute Values have Zipfian Distribution with varying degrees of skew

Effect of Having Uniform Data



- With uniformly distributed join attribute values, all online algorithms perform almost same, OPT-offline performs little improvement

Zipfian Distribution

- It is the distribution of occurrence probabilities which follow Zipf's law. Probabilities starts high and tapers off exponentially. Thus, a few items occur very often while many others occur rarely.

- Zipfian distribution is defined as:

$$P_n \approx a.n^{-\theta}$$

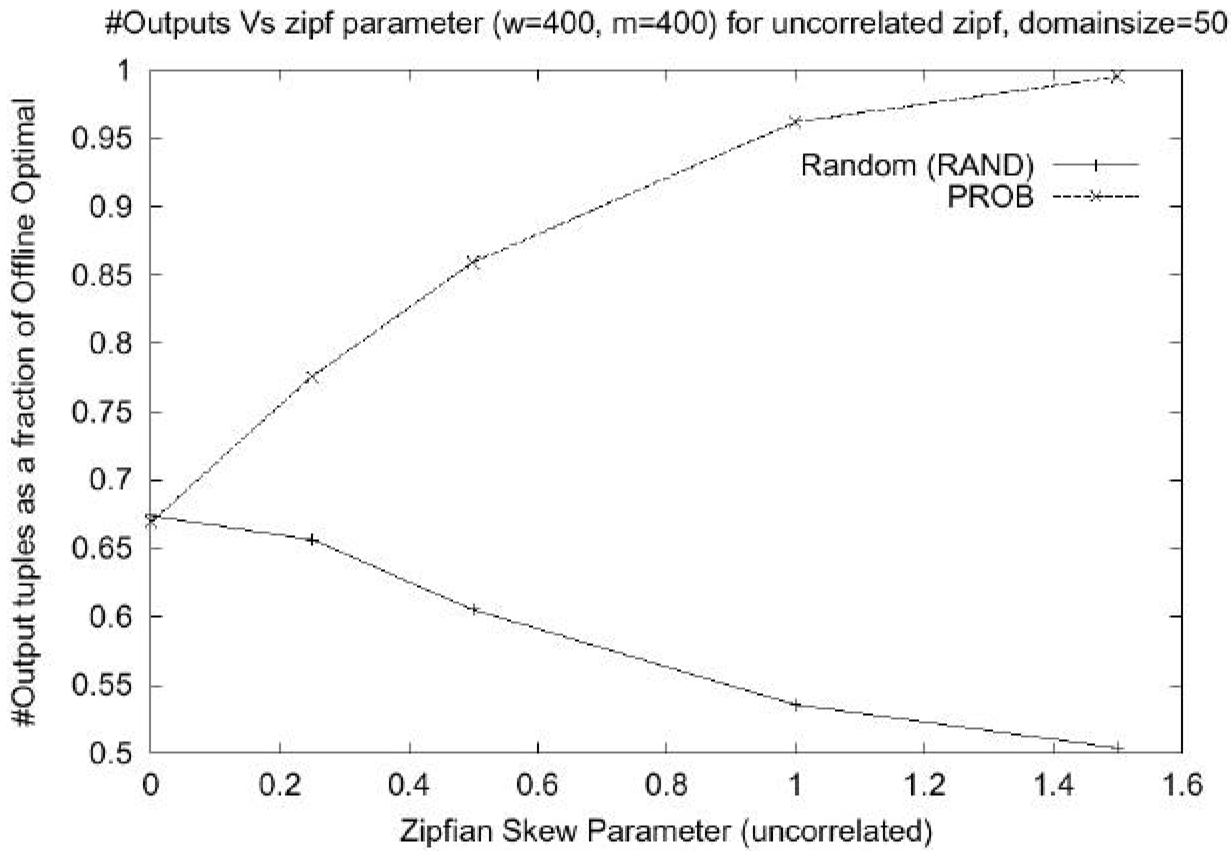
P_n : the frequency of occurrence of the n^{th} ranked item

a : a number close to 1

θ : skew parameter

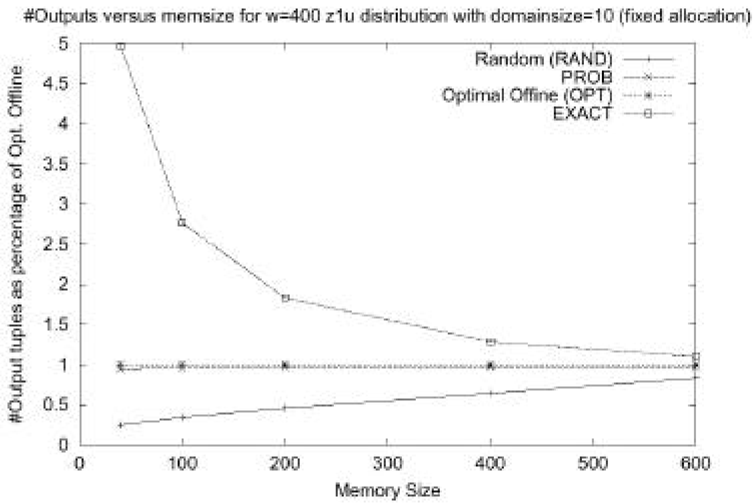
- If θ is big, probabilities drop quickly , else they drop slowly

Effect of Zipfian Skew Parameter

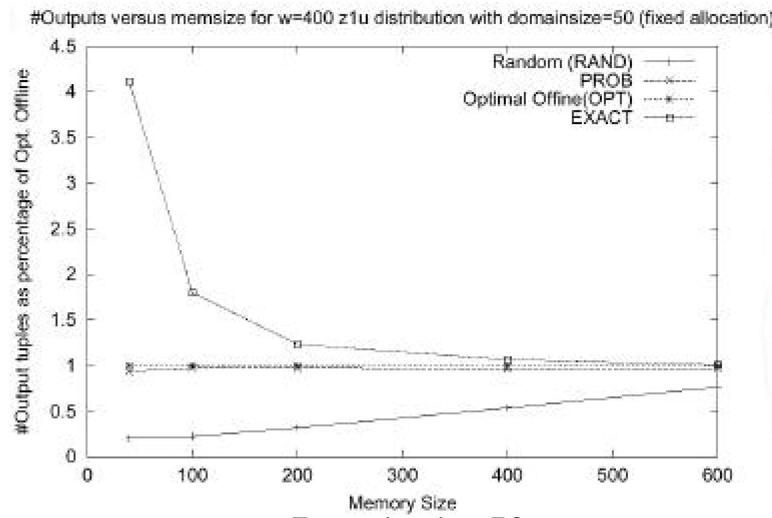


- PROB performs better than RAND as the skew increases

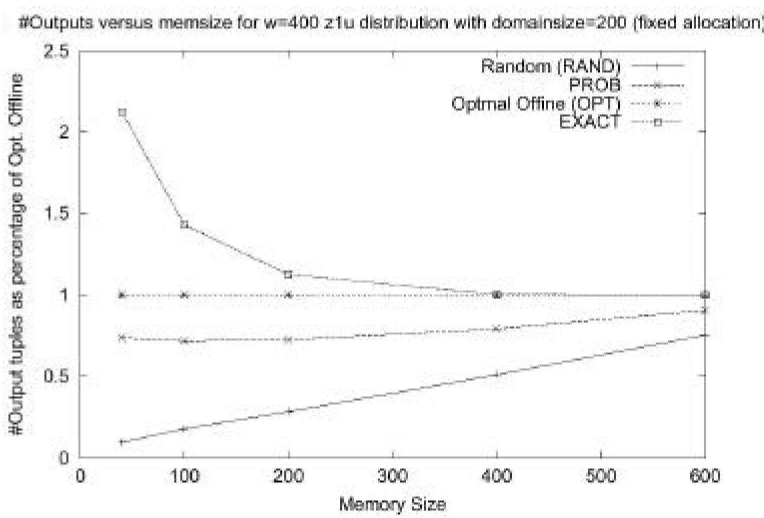
Effect of Domain Size



Domain size 10



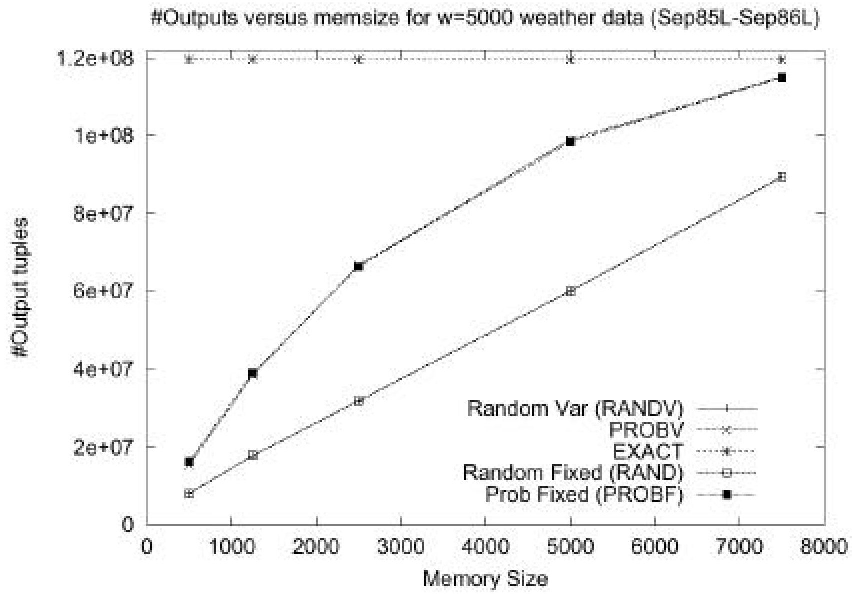
Domain size 50



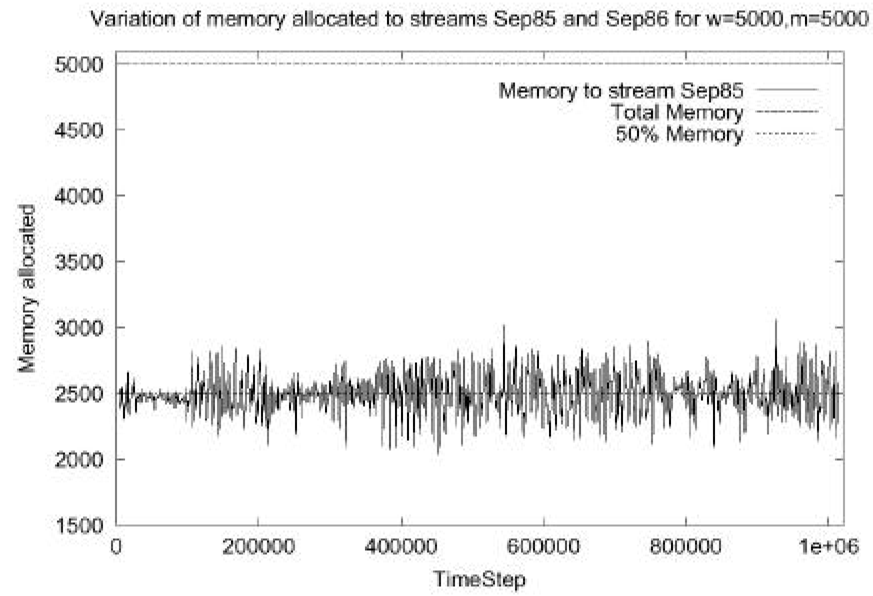
Domain size 200

- The performance of PROB and OPT-offline drops as the domain size increase. But, the performance of PROB gets worse than OPT-offline.

Experiments with Real Life Data



Weather data: Performance



Weather data: memory allocation

- The behavior of the algorithms is similar to synthetic dataset results

Related Work and Developments

- Previous work:
  - J. Kang, J. F. Naughton, and S. D. Viglas. Evaluating window joins over unbounded streams.
    - This paper also investigates algorithms for evaluating sliding window joins over unbounded streams. They consider the cases where :
      - data arrival rates of the input streams are different
      - processing speed is insufficient to keep with streams
      - memory is limited.

- Developments:

  The paper has 2 citations:
  - L. Golab, S. Garg and M. Tamer Ozsu. On Indexing Sliding Windows over On-line Data Streams.
    - Talks about sliding window indexing in main memory over online data streams
  - Ahmet Bulut and Ambuj K. Singh. Stardust: Fast Stream Indexing using Incremental Wavelet Approximations
    - They propose an approach for summarizing a set of data streams, and for constructing a composite index structure to answer similarity queries.

QUESTION

What is the use of
"Static Join Algorithm"
in this paper?

QUESTIONS ?

?

?

?