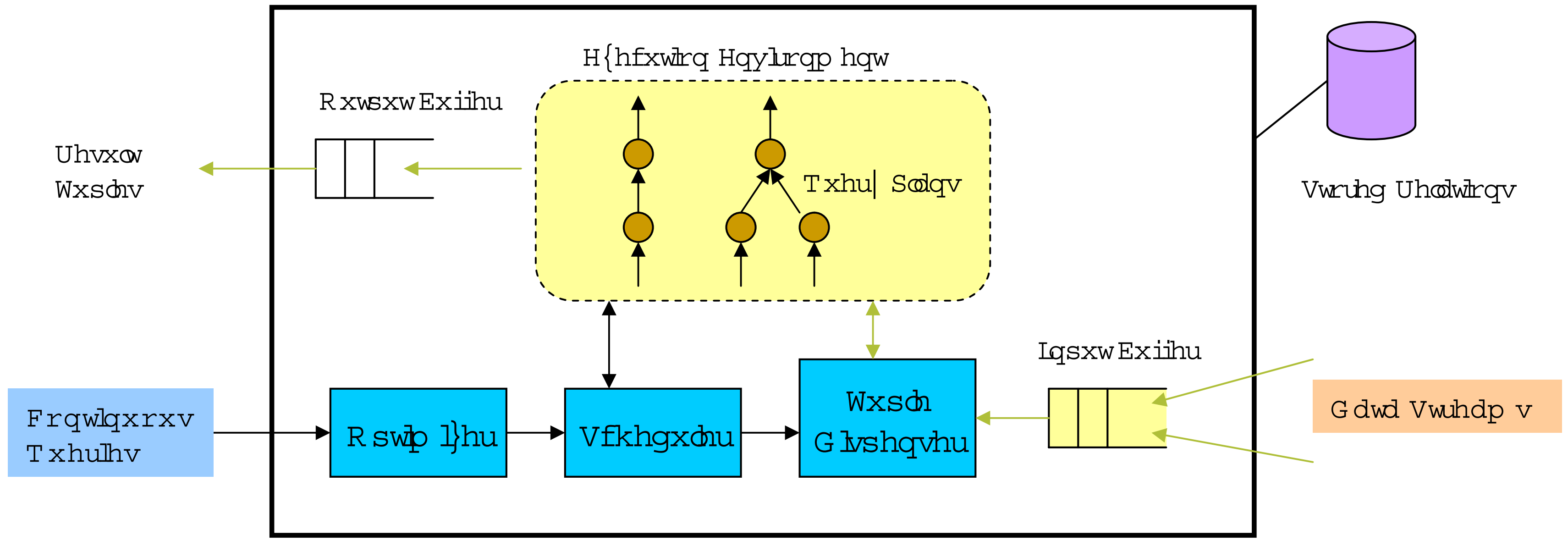
iJoin: Importance-aware Join Approximation over Data Streams

Dhananjay Kulkarni (Boston University)
Chinya V. Ravishankar (University of California – Riverside)

SSDBM 2008

G dwd Vwuhdp P dqdjhp hqw V|whp v



A continuous query submitted once, but executed multiple times as new data arrives

- Average temperature on floor 3, every 10 minutes
- IP addresses of all packets, going to destination 64.233.167.99 in last 1 hour

Data stream is a continuous, unbounded, time varying sequence of data tuples

- Data generated by sensors (temp, pressure)
- Network monitoring data
- XML Data

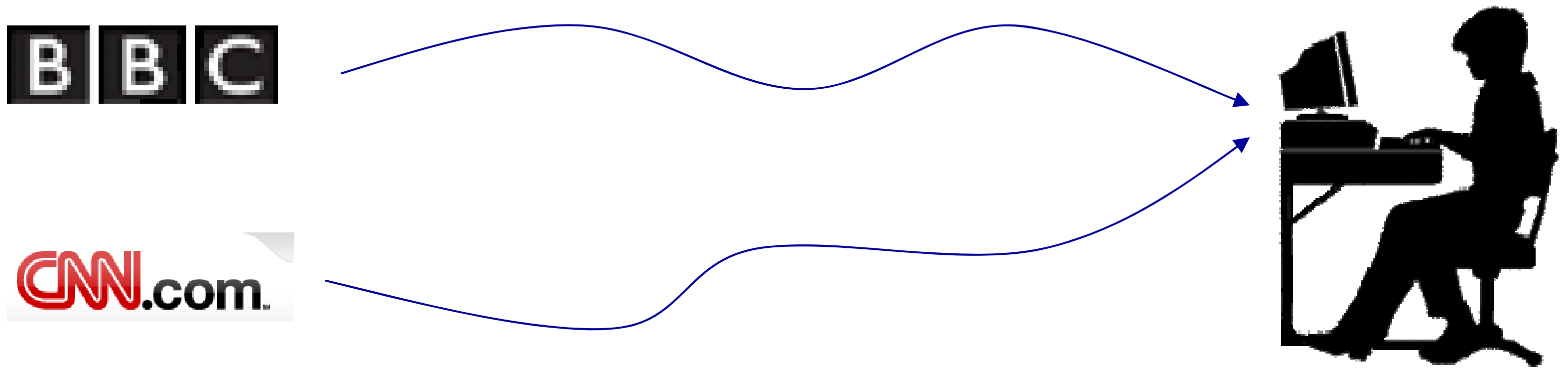
Uhodwlrqdo GEPV yv1 Gdwd Vwuhdpv

	Relational DBMS	Data Stream Management
Data Characteristics	Mostly static, disk resident	Continuous, unbounded, mostly processed in memory
Query Model	One-time query, based on request-response (pull)	Continuous query, based on pushing incoming data to queries
Input to Queries	Complete Relation/table	Tuples within the 'window'
Cost Model	Minimize disk I/O	Output rate, memory utilization
Catalog information (data distribution, size)	Remains fairly static, unless there are updates	Changes dynamically as new data arrives into the system
Overflow data	Written to disk	Dropped from processing or summarized in memory
Access methods	Index structures	Summaries, histograms
Examples	Inventory, census, payroll	Sensors, online bids, news feeds
Implementations	Oracle, sybase, etc ..etc	Streambase, coral8, and other university research projects (STREAM, Aurora)

Dssur{lpdwh0Mrlqv ryhuGdwd Vwuhdpv

- Joins are used to find correlations among data
- Applications
  - Find the news articles under the same category that appear in BBC and CNN news-feed
    - Equi-Join
  - Find all sensors that are reporting higher temperatures than other sensors in the area
    - Conditional Join

Qhzv Ihhg H{dpsoh



Data streams:

- html link
- keyword (e.g. California Fires, LA Lakers)
- category (e.g. sports, politics, weather)
- timestamp

Sliding Window Join Query

Find all news articles reported in the last 2 hours that have the same keyword

iGoogle does something like this?

Ydxh0edvhg Lpsruwdqfh

- The tuple importance is determined based on the value
  - User Preferences:
    - Politics news in the morning
    - Sports news in the evening

$$\begin{tikzcd}\boxed{\text{Importance attribute}} \arrow[r, "F_i"] & \boxed{\text{Importance domain}} \\ & \text{1 to n}\end{tikzcd}$$

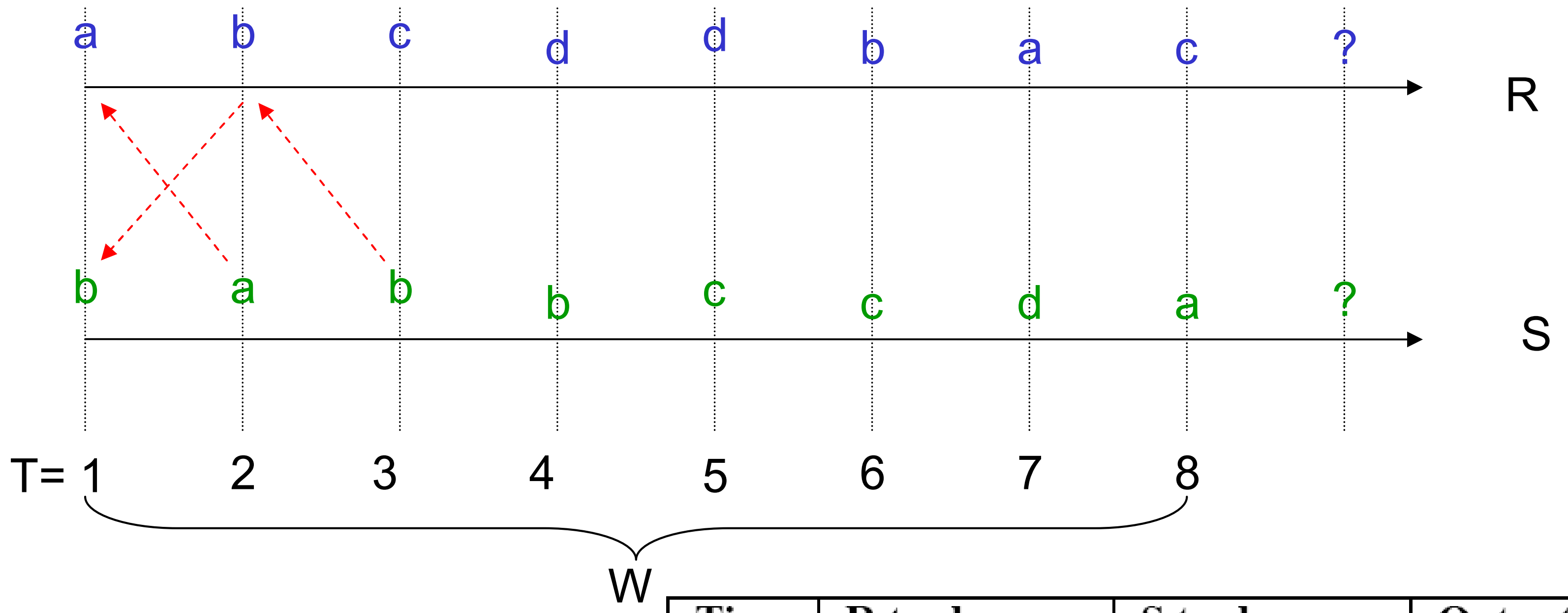
Mrlq lv d vwdwhixo rshudwru

- Consider
  - Streams R and S
  - Window W
  - Current time is T
- The sliding window join requires following R and S tuples to be in memory
  - r(i), such that T-W <= i <= T
  - s(j), such that T-W <= j <= T

H{dpsoh

- Streams arrive at 1 tuple per second
- Equi-join over the 2 streams R and S
- Importance Function:
  - imp(a) = 1
  - imp(b) = 2
  - imp(c) = 3
  - imp(d) = 4

H {dfwMrlq



Time	R tuples	S tuples	Output	Imp
2-3	a,**b**	b,**a**	b,a	3
3-4	a,b,c	b,a,**b**	b	2
4-5	a,b,c,d	b,a,b,**b**	b	2
5-6	a,b,c,d,d	b,a,b,b,**c**	c	3
6-7	a,b,c,d,d,**b**	b,a,b,b,c,**c**	b,b,b,c	9
7-8	a,b,c,d,d,b,**a**	b,a,b,b,c,c,**d**	a,d,d	5
8-9	a,b,c,d,d,b,a,**c**	b,a,b,b,c,c,d,**a**	c,c,a,a	8

- Memory is not limited
- Output Size = 16
- Total Importance = 36

Dssur{lpdwh Mrlq +UDQG,

Load-shedding Policy:

When M is full, randomly choose a tuple to drop.

Time	R tuples	S tuples	Output	Imp
2-3	a,**b**	b,**a**	b,a	3
3-4	a,c	a,b	-	0
4-5	a,d	b,b	-	0
5-6	d,d	b,c	-	0
6-7	d,**b**	b,c	b	2
7-8	b,a	b,d	-	0
8-9	b,c	b,a	-	0

- M=2
- Output Size = 3
- Total Importance = 5

Observation:

- R(c, 3) is dropped too early
- R(a,1) remains unproductive

Dssur{lpdwh0Mrlq +ILIR ,

Load-shedding Policy:

When M is full, drop the tuple at the front of the queue.

Time	R tuples	S tuples	Output	Imp
2-3	a ,**b**	b,**a**	b,a	3
3-4	b,c	a,**b**	b	2
4-5	c,d	b,b	-	0
5-6	d,d	b,c	-	0
6-7	d,b	c,c	-	0
7-8	b,a	c,d	-	0
8-9	a,c	d,**a**	a	1

- M=2
- Output Size = 4
- Total Importance = 6

Observation:

- Does not exploit any value correlation
- Suffers from early drops

Dssur{lpdwh0Mrlq +VL]H,

Load-shedding Policy:

When M is full, drop a tuple corresponding to a value that has produced least outputs.

Time	R tuples	S tuples	Output	Imp
2-3	a ,**b**	b,**a**	b,a	3
3-4	a,b	a,**b**	b	2
4-5	a,b	b,**b**	b	2
5-6	a,b	b,b	-	0
6-7	b,**b**	b,b	b,b	4
7-8	b,b	b,d	-	0
8-9	b,c	b,a	-	0

- M=2
- Output Size = 6
- Total Importance = 11

Observation:

- Does not exploit tuple-importance
- Suffers from early drops

Gudzedfnv ri Suhylrxv Vfkhphv

1. None exploit tuple-importance
2. Suffer from premature tuple drops
3. Suffer from unproductive tuple retention
4. Unfair to some tuples with respect to time spent in memory
   - tuple-lifetime

R xuDssur{lp dwlrq R emhfwlyhv

- Overcome the drawbacks + maximize the total importance of join output
- We do not have any foreknowledge of
  - Tuple arrival characteristics
  - Data distributions

Mrlq0rxwsxwtxdolw|

- We measure the join-quality in terms of the importance of tuples
  - o(i).imp = min { r(j).imp, s(i).imp }
  - We can also use: max, sum, product, etc.
- Output set: Ω
- Total Importance of query q
  - IMP(q) = ∑ o(i).imp, where o(i) ε Ω

Sureohp Vwdwhp hqw

- Given the available memory M and a sliding window join query <α, c, w>, compute the approximate join, such that the total importance of the output is maximized.
    - α = set of streams {S1, S2, S3 ..Sn}
    - c is the join condition
    - w is the window size

Ohw´v eh J UHHG\

Load-shedding Policy:

When M is full, drop the tuple with the lowest importance.

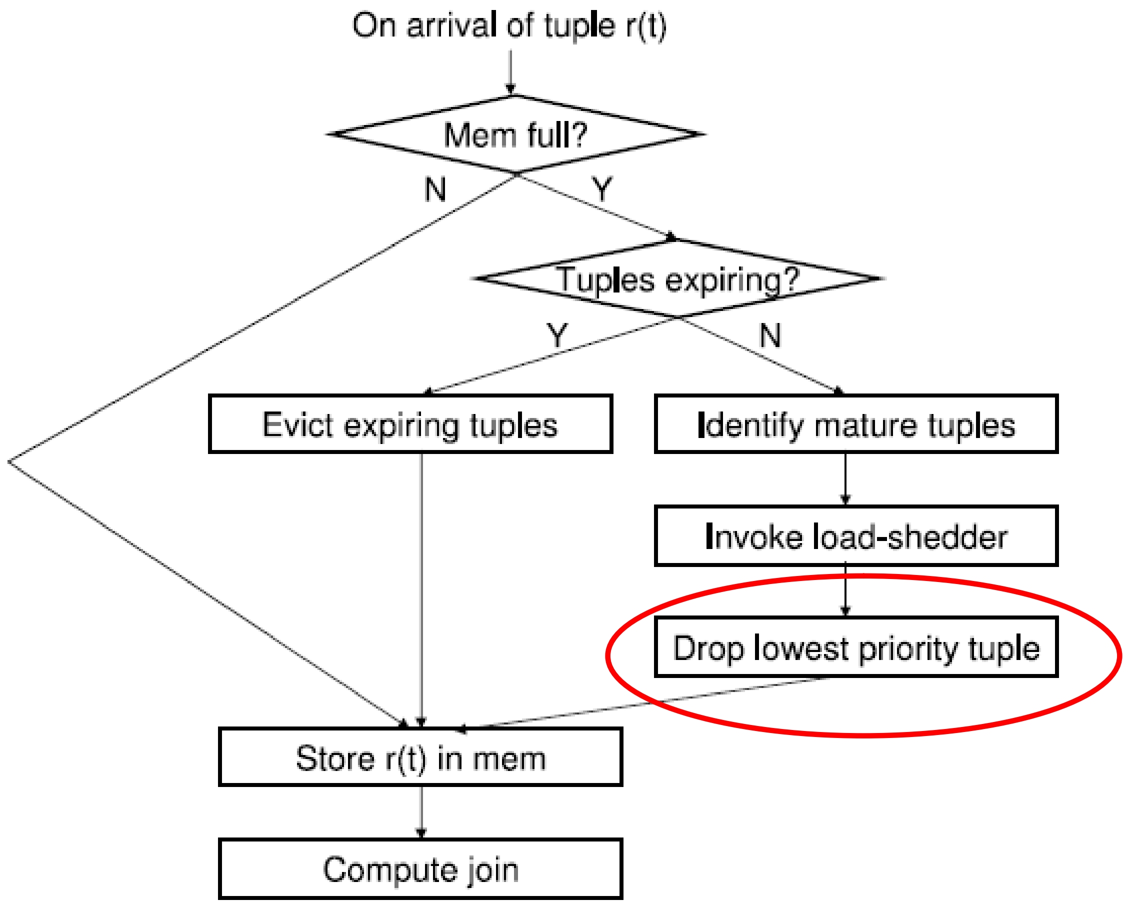
Time	R tuples	S tuples	Output	Imp
2-3	a,**b**	b,**a**	b,a	3
3-4	b,c	b,**b**	b	2
4-5	c,d	b,b	-	0
5-6	d,d	b,c	-	0
6-7	d,d	c,c	-	0
7-8	d,d	c,**d**	d,d	8
8-9	d,d	c,d	-	0

- M=2
- Output Size = 5
- Total Importance = 13

Observation:

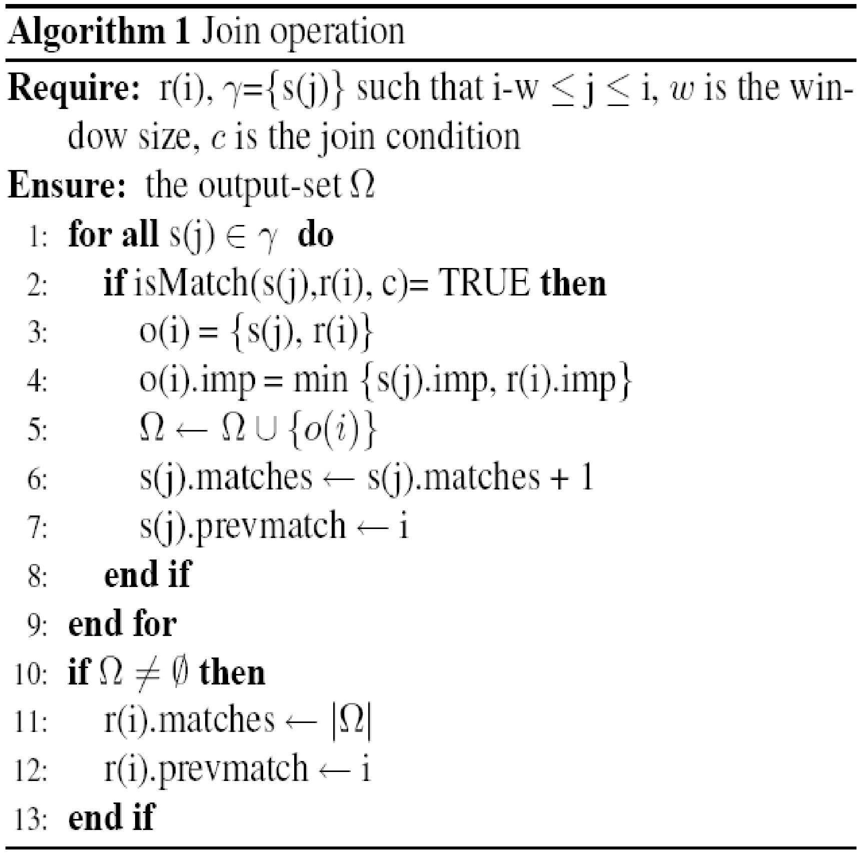
- Does not exploit any value correlation
- Suffers from premature drops
- Suffers from unproductive tuples
- Unfair!

MR IQ R yhuylhz



Wxsoh P hwd0G dwd

- We store the following with each tuple
  - t_a : arrival time
  - imp: tuple-importance
  - matches: number of join-output generated, so far.
  - prevmatch: timestamp of the most recent match
- This meta-data is used to compute tuple priority



Algorithm 1 Join operation

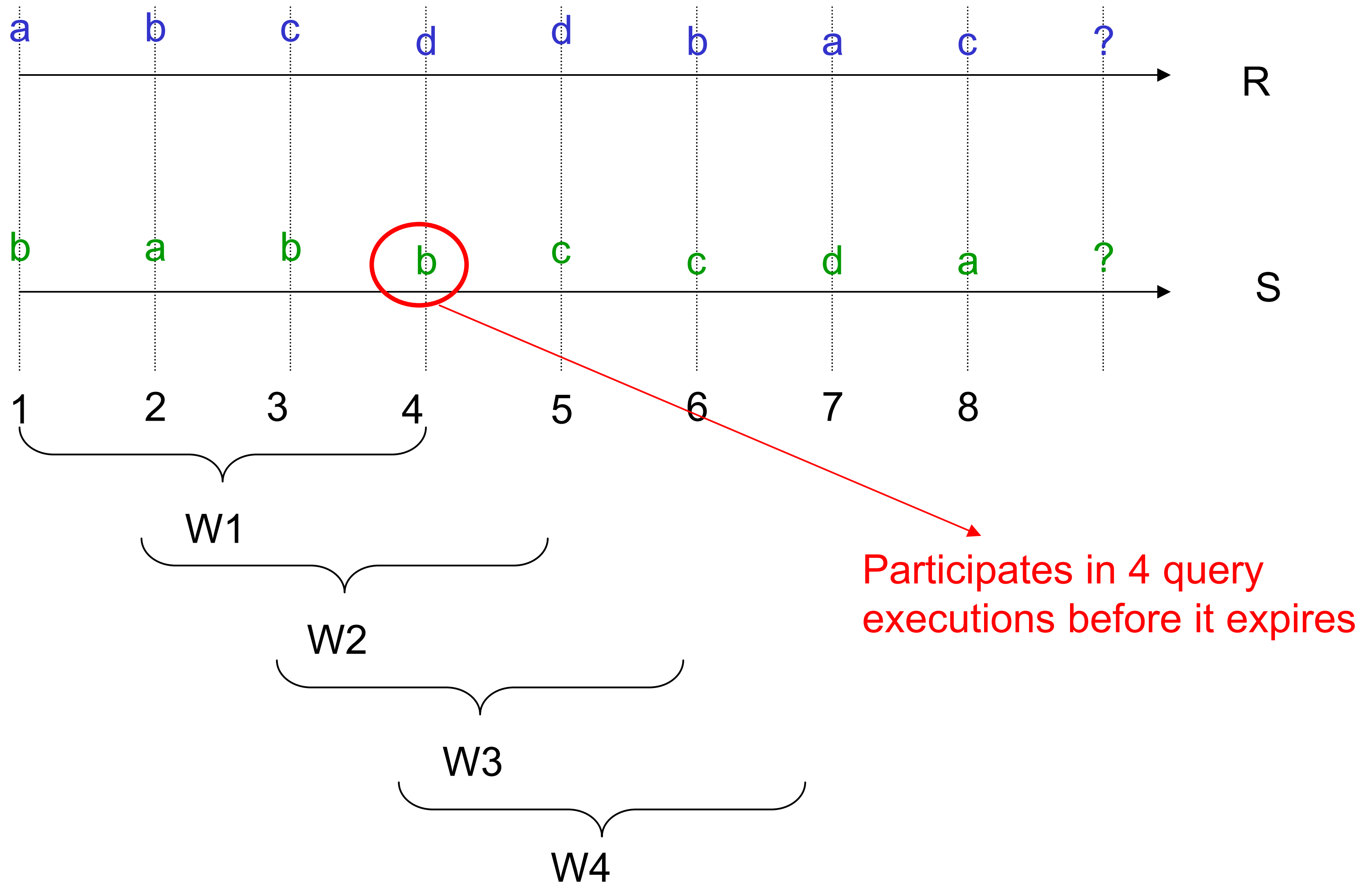
```
Require: r(i), $\gamma$={s(j)} such that i-w $\leq$ j $\leq$ i, $w$ is the window size, $c$ is the join condition
Ensure: the output-set $\Omega$
for all s(j) $\in \gamma$ do
    if isMatch(s(j),r(i), c)= TRUE then
        o(i) = {s(j), r(i)}
        o(i).imp = min {s(j).imp, r(i).imp}
        $\Omega \leftarrow \Omega \cup \{o(i)\}$
        s(j).matches $\leftarrow$ s(j).matches + 1
        s(j).prevmatch $\leftarrow$ i
    end if
end for
if $\Omega \neq \emptyset$ then
    r(i).matches $\leftarrow |\Omega|$
    r(i).prevmatch $\leftarrow$ i
end if
```

Wxsoh0Sulrulw|

- Basically, the estimated ‘worth’ of the tuple
- On arrival:
  - $P(r(i), t) = P_{INIT}$
- At any time t’:
  - $P(r(i), t') = F_p(r(i).imp, r(i).matches, r(i).age)$

$$\text{Priority-function } F_p(imp, matches, age) = \frac{imp \times matches}{age}$$

Joredyv1Orfdo Sulrulw



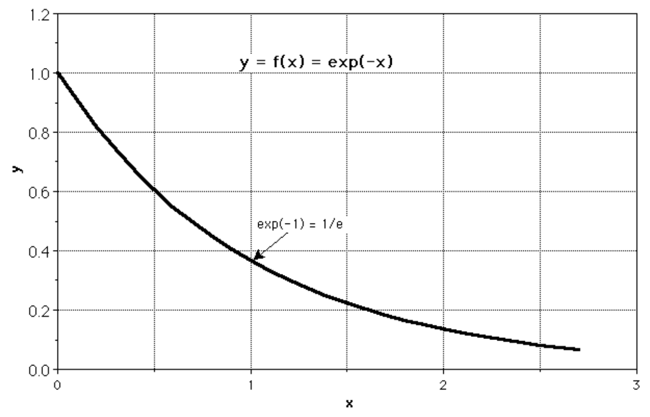
Orfdo Sulrulw| Ixqfwlrq

$$\text{Priority-function } F_p(\text{imp, matches}', \text{age}) = \frac{\text{imp x matches}'}{\text{age}}$$

Where,

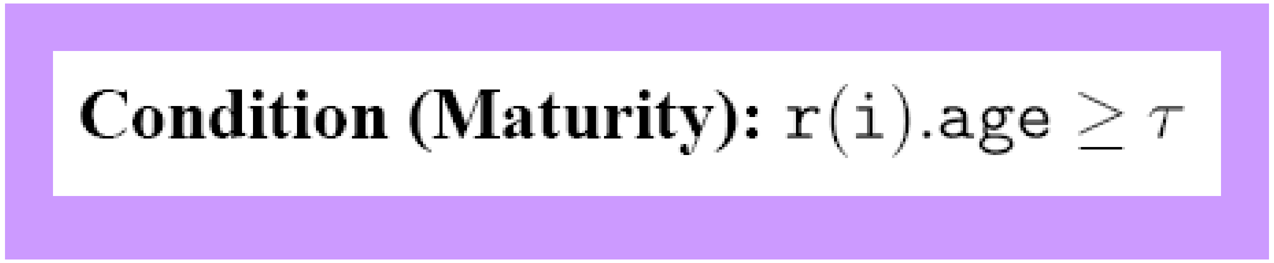
$$\texttt{matches}' = \sum_{t=t_a}^{t'} \{match_{[t,t-1]} \times e^{-d(t'-t)}\}$$

- d is a constant in the decay function
- $match_{[t,t-1]}$ is the number of matches in the window [t,t-1]



Wxsoh Pdwxulw|

- Only mature tuples should be considered as candidates for tuple eviction
- User-specified threshold: τ



Condition (Maturity): $\mathtt{r(i).age} \geq \tau$

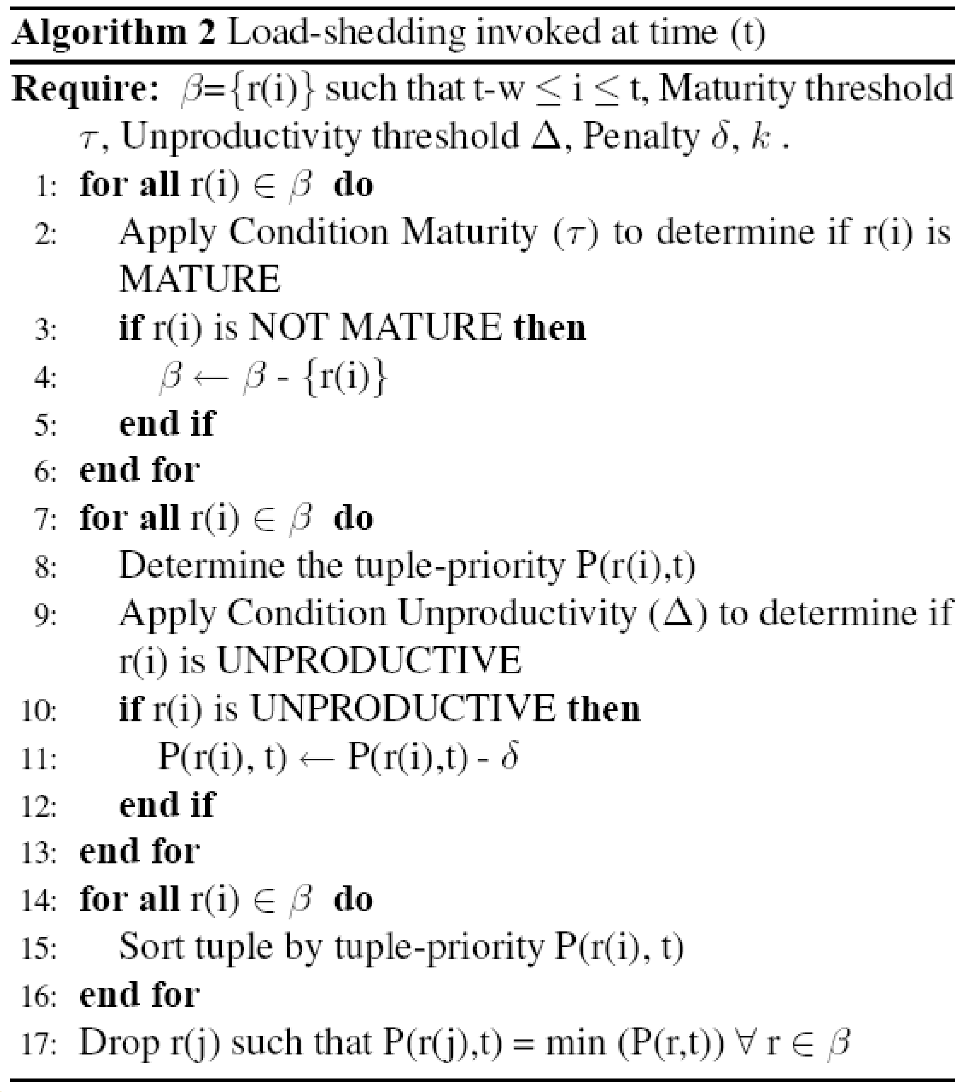
Xqsurgxfwlyh Wxsoh

- Unproductive tuples should be penalized.
- User-specified threshold: Δ

Condition (Unproductivity): $\mathtt{t}' - \mathtt{r(i).prevmatch} \geq \Delta$

$$\mathtt{Penalty}(\delta) = \mathtt{c} \times (\mathtt{t}' - \mathtt{r(i).prevmatch})$$

R xuOrdg0vkhgglqj Vfkhph



Algorithm 2 Load-shedding invoked at time (t)

Require: β={r(i)} such that t-w $\leq$ i $\leq$ t, Maturity threshold τ, Unproductivity threshold Δ, Penalty δ, k .

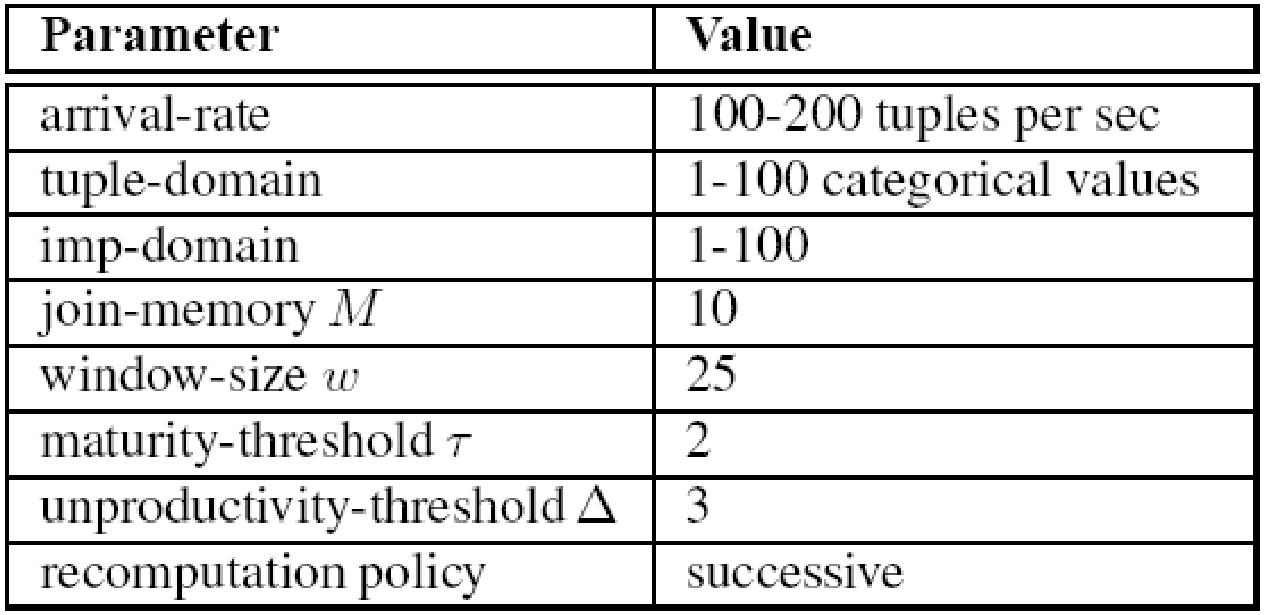
```
1:  for all r(i) $\in \beta$ do
2:      Apply Condition Maturity ($\tau$) to determine if r(i) is
        MATURE
3:      if r(i) is NOT MATURE then
4:          $\beta \leftarrow \beta$ - {r(i)}
5:      end if
6:  end for
7:  for all r(i) $\in \beta$ do
8:      Determine the tuple-priority P(r(i),t)
9:      Apply Condition Unproductivity ($\Delta$) to determine if
        r(i) is UNPRODUCTIVE
10:     if r(i) is UNPRODUCTIVE then
11:         P(r(i), t) $\leftarrow$ P(r(i),t) - $\delta$
12:     end if
13: end for
14: for all r(i) $\in \beta$ do
15:     Sort tuple by tuple-priority P(r(i), t)
16: end for
17: Drop r(j) such that P(r(j),t) = min (P(r,t)) $\forall$ r $\in \beta$
```

Z khq wr uh0frp sxwh wxsoh0sulrulw|B

- Frequent re-computation leads significant overhead
- Can we trade-off between overhead and join quality?
- We propose 3 re-computation schemes:
  1. Successive
     - Priority is determined after each tuple drop
  2. k-Successive
     - Priority is determined after k drops
  3. Adaptive
     - Priority is determined only if join-quality drops by some threshold

H{shulp hqw

- Synthetic dataset with importance semantics
- Equi-join queries
- Only memory is a constraint (FastCPU case)



Parameter	Value
arrival-rate	100-200 tuples per sec
tuple-domain	1-100 categorical values
imp-domain	1-100
join-memory M	10
window-size w	25
maturity-threshold τ	2
unproductivity-threshold Δ	3
recomputation policy	successive

P hdvxuhp hqw

- total importance: this is the measure of join-output quality
- output-size: number of output tuples generated by the query
- fairness: a measure of how the algorithm performs with respect to how long all tuples stay in memory

P ruh rq idluqhvv

- None of the previous work consider fairness
- Fairness is important for several reasons
  1. We have no knowledge of what is the ‘worth’ of a tuple if it is dropped.
  2. 100% fairness will give all tuples equal chance of being part of the join-result.
- We use Jain’s measure of fairness, where L_i is the time spent by each tuple in join-memory.

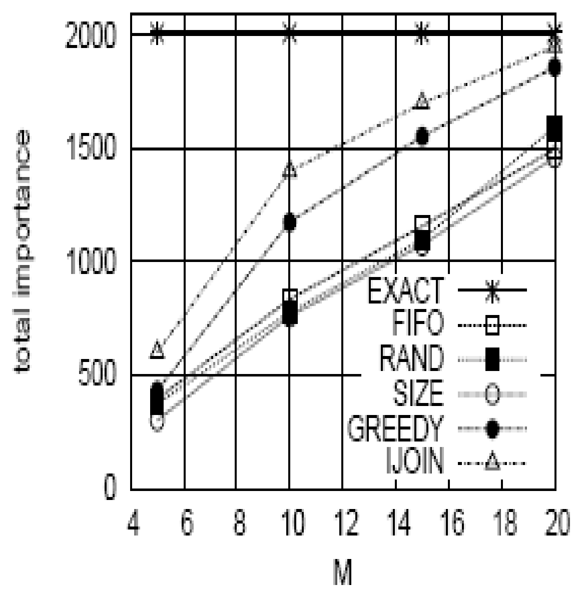
$$\text{fairness} = \frac{(\sum L_i)^2}{n \times (\sum L_i^2)}$$

- How ‘fair’ is ‘fair’?
  - Worst case: 1/n
  - Best case: 1

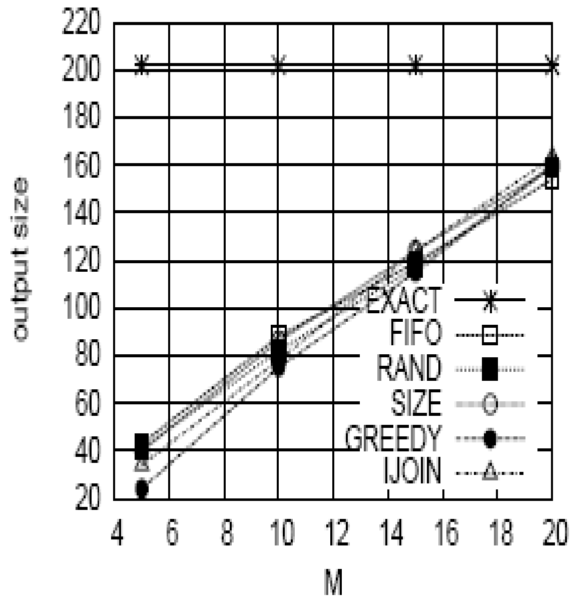
Vfkhp hvxvhg iru frp sdulvrq

1. EXACT
   - Unlimited Memory
2. FIFO
   - First-in-first-out
3. RAND
   - random drop
4. SIZE
   - tries to maximize the output size
5. GREEDY
   - drops the least important tuple
6. IJOIN
   - drops the only a mature tuple with lowest priority

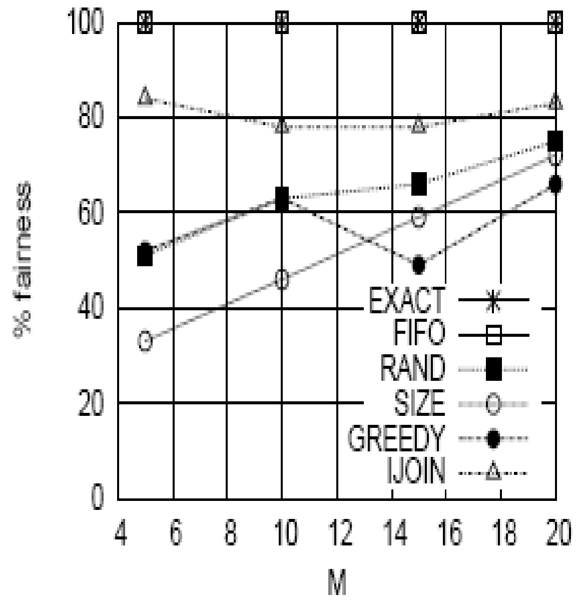
H iihfwri P hp ru| Vl}h



(a) Join output quality



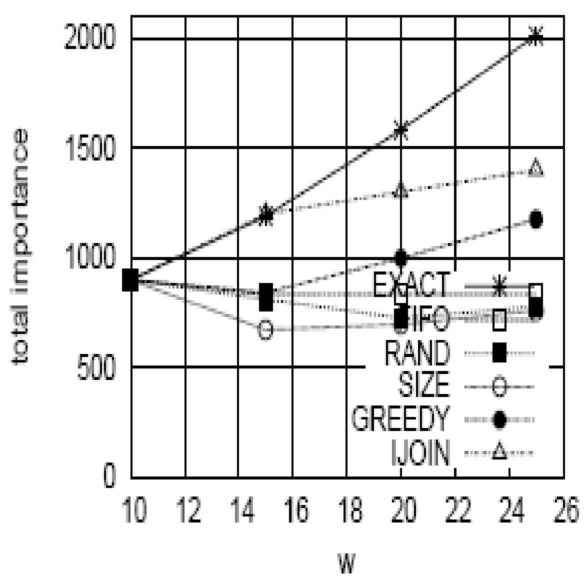
(b) Number of outputs



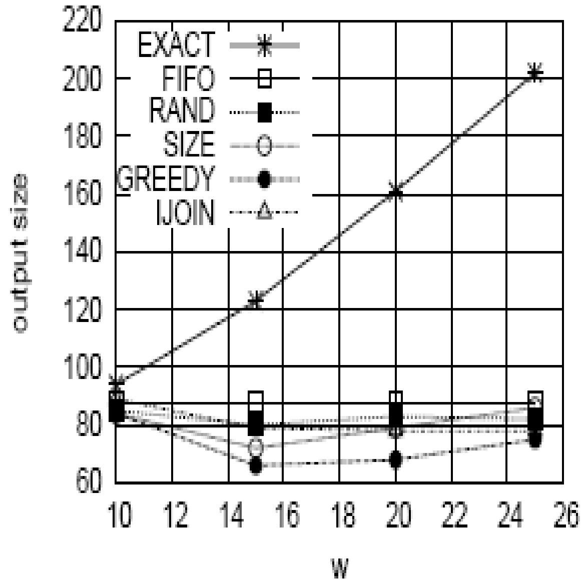
(c) Fairness Index

- FIFO is 100% fair, but join quality is low
- IJOIN scales well with increase in memory, and is consistently fair

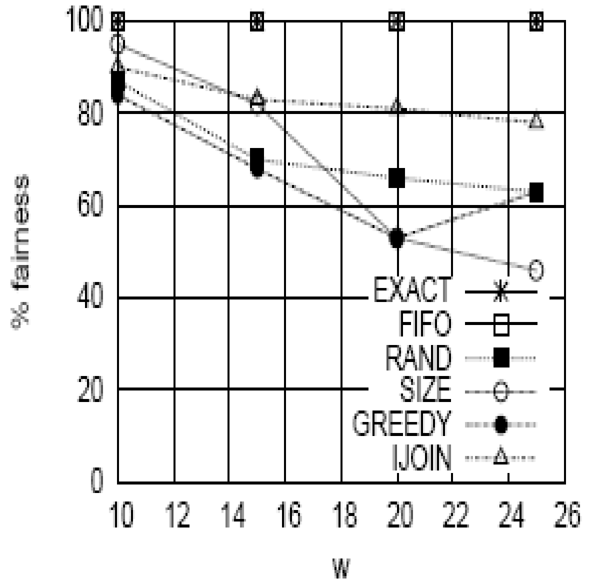
Hiihfwri Zlqgrz Vl}h



(a) Join output quality



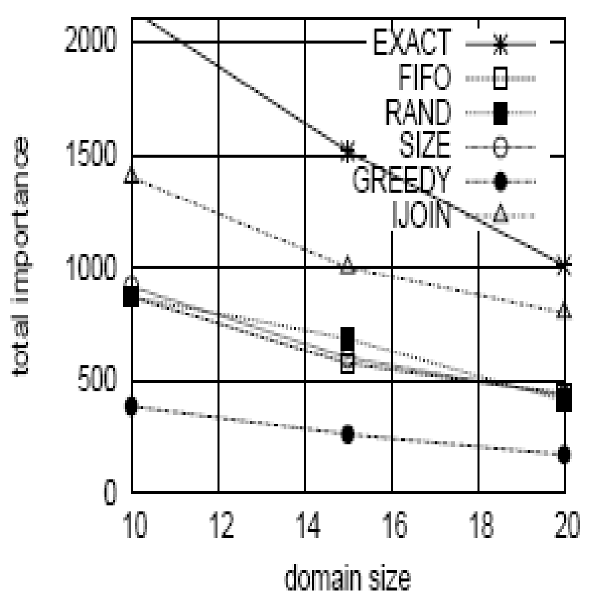
(b) Number of outputs



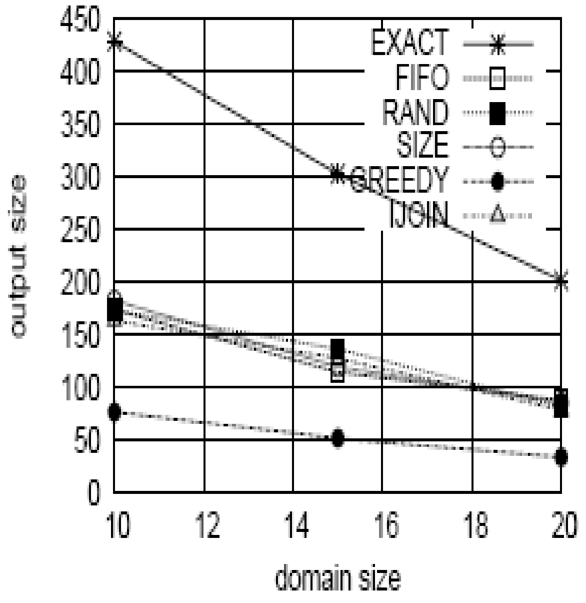
(c) Fairness Index

- Only IJOIN improves on join-quality
- Larger windows, provides better opportunity for IJOIN to estimate correlations
- Fairness drop from 90%-42% (SIZE)
- IJOIN is still between 80%-88% fair

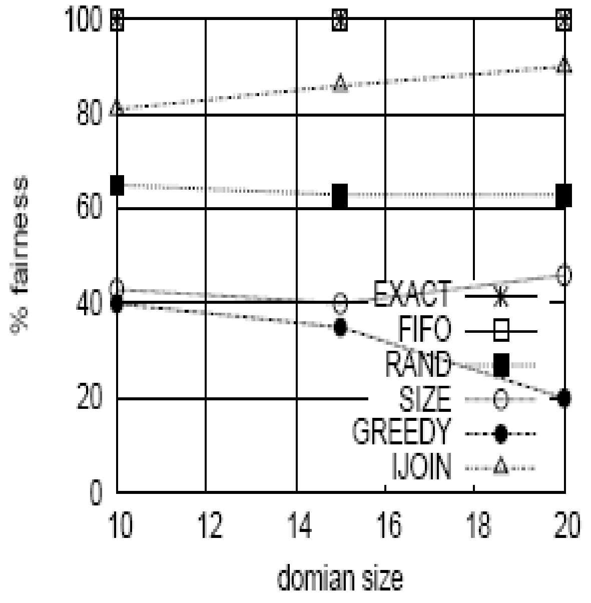
Hiihfw ri Grpdlq Vl}h ri Mrlq Dwwulexwh



(a) Join output quality



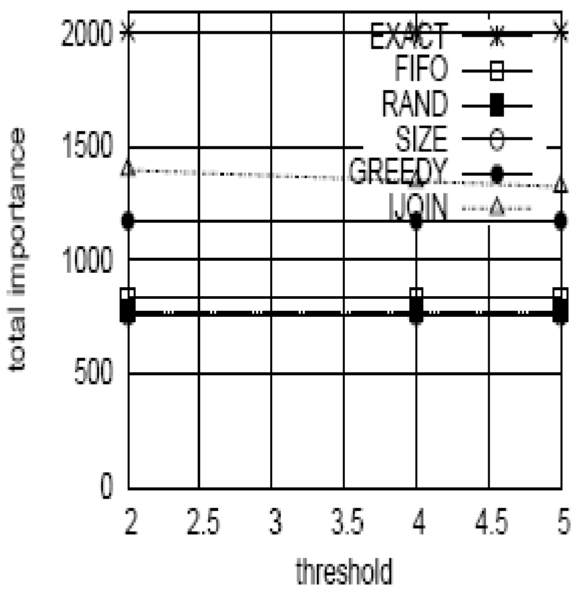
(b) Number of outputs



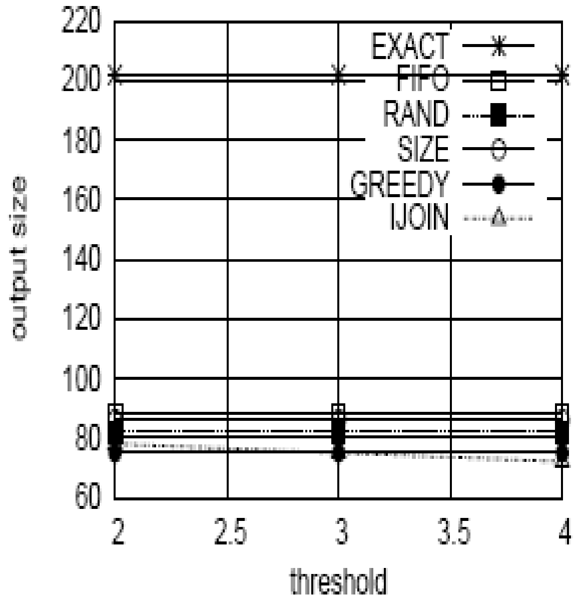
(c) Fairness Index

- All tuples have equal importance in this experiment
- Probability of finding a matching tuple drops with increase in domain size
- GREEDY is the worst in all measures

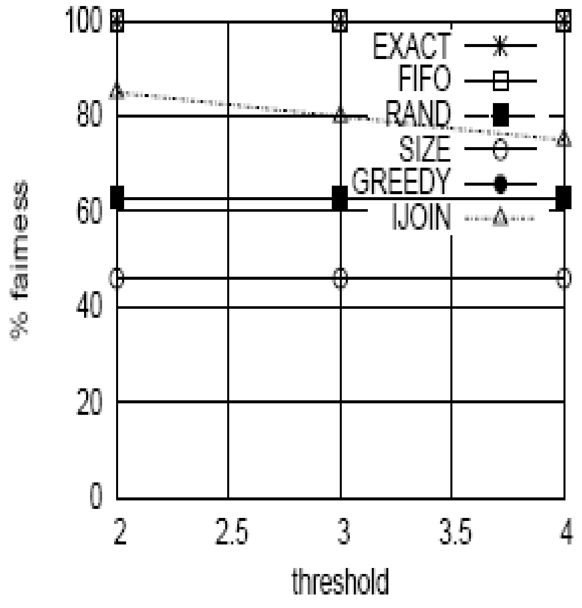
H iihfwri X qsurgxfwlylw| Wkuhvkrog



(a) Join output quality



(b) Number of outputs



(c) Fairness Index

- This threshold is IJOIN specific
- IJOIN fairness is low (75%) when threshold is coarse (high value)
- Use lower threshold, or apply higher penalty!

Shuirup dqfh ri Uh0frp sxwdwlrq Vfkhp hv

	Successive	K-Successive	Adaptive
Total importance	1400	1200	1300
Output Size	80	75	80
Fairness	81 %	75 %	80 %
Overhead	High (1000 times)	Low (100 times)	Medium (600 times)

- Use Successive scheme if data is expected to be erratic
- Use Adaptive scheme for relatively stable distributions

Mrlq=Lp sruwdqwUhvxow

- iJoin addresses tuple ‘importance’
- We have developed a framework that is very effective in maximizing join-quality
  - limits premature drops
  - penalizes unproductive tuple
- iJoin out-performs previous schemes used for load-shedding and join-approximation
- Fairness: Except for FIFO, IJOIN is the best
  - 80%-85% fair

Wkdqn \rx$