Efficient Join-Index-Based Spatial-Join Processing: A Clustering Approach

Chang-tien Lu

Advisor: Prof. Shashi Shekhar

Department of Computer Science

University of Minnesota

ctlu@cs.umn.edu

Ph. (612) 626-7703

Biography

- B.S. in Computer Sci. and Eng., Tatung Institute of Technology, Taiwan, 1991
- M.S. in Computer Sci., Georgia Institute of Technology, 1996
- Ph.D. student, University of Minnesota, 1996-present
- Main Publications
  - *"Optimizing Join Index Based Join Processing: A Graph Partitioning Approach," in 17th IEEE Symposium on Reliable Distributed Systems, Oct. 1998*
  - *"Spatial Databases: Accomplishments and Research Needs," in IEEE Knowledge and Data Engineering, Volume 11, Number 1(January/February 1999), Page 45-55*
  - *"Map Cube: A Visualization Tool for Spatial Data Warehouses," in Proc. NSF workshop on Data Mining in GIS, March 1999*
  - *"Efficient Join Index Based Join Processing: A Clustering Approach," submitted to IEEE Knowledge and Data Engineering*

Overview

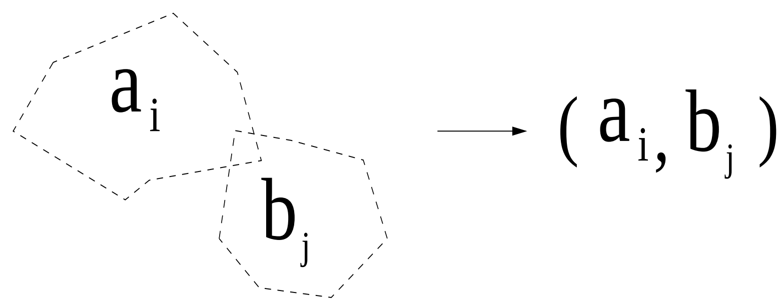
- Motivation, Basic Concept, Problem Formulation $\Leftarrow$
- Asymmetric Clustering
- Comparison of Asymmetric Approaches
- Symmetric Clustering
- Comparative Evaluation of SC, AC, and Competitors
- Conclusion and Future Work

Motivation

- Spatial data is complex
  - *Contains points, lines, polygons, etc.*
- Spatial data is large
  - *Contains polygons with thousands of edges*
- Spatial data processing is expensive
  - *a typical query on a 100,000 polygon data set takes 200 seconds*
- Research questions
  - *How to efficiently process a join query, given a join-index?*
- Application domains of Spatial Join & Join-index
  - *GIS: Overlay(Union, Intersection, Difference)*
  - *Data Warehousing: Multiple Join(A StarJoin) [5]*
  - *Data Mining: Neighborhood index [24]*
  - *Crime Analysis [19]*

Spatial Join

- A join between two sets of spatial objects
- Input: Two sets of spatial objects $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$
- Output:
  - *All pairs* (a_i, b_j) *with* $(a_i) \cap (b_j) \neq 0$



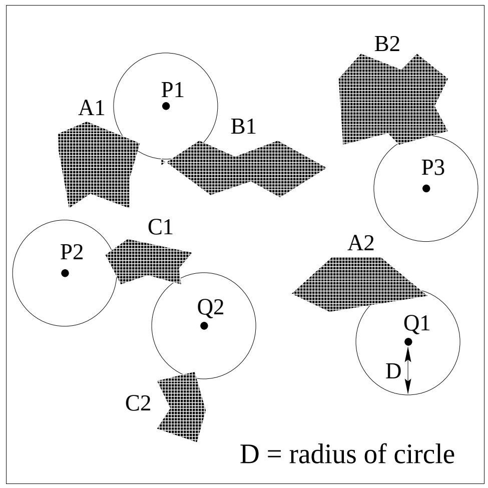
- Algorithms for Spatial Join
  - *Plane Sweep, Divide-and-conquer algorithm [10]*
  - *Spatial Hash Join [17, 21]*
  - *Space Partitioning [14]*
  - *Nested Loop [16]*
  - *Tree Matching*
    - Grid files [2]
    - R-Trees [3, 12]
    - Generalization trees [9]
  - *Join-index [22, 18]*

Join-index

- What is join index
  - *Describe a relationship between two relations*
  - *Each tuple identifies one tuple in the result*
  - *Pre-joined relation*
  - *Stored separately from the operand relations*
  - *Used only for the most important joins*
- Benefits
  - *Much smaller than joined relation*
  - *Simple structure*
  - *Compatible with other operations, e.g., select and union*
- Applications
  - *Low update rate, quick response, large data set*
  - *Example: Spatial data set*

An Example of Spatial Join and Join-index

- Spatial Join



- Join-index

R Dataset (Facility Location)

ID	Location (X,Y)	Non-Spatial Data
P1	(7.9,16.7)	(...)
P2	(3.4,11.4)	(...)
P3	(19.5,13.1)	(...)
Q1	(18.7,6.4)	(...)
Q2	(9.5,7.1)	(...)

S Dataset (Forest-Stand Boundary)

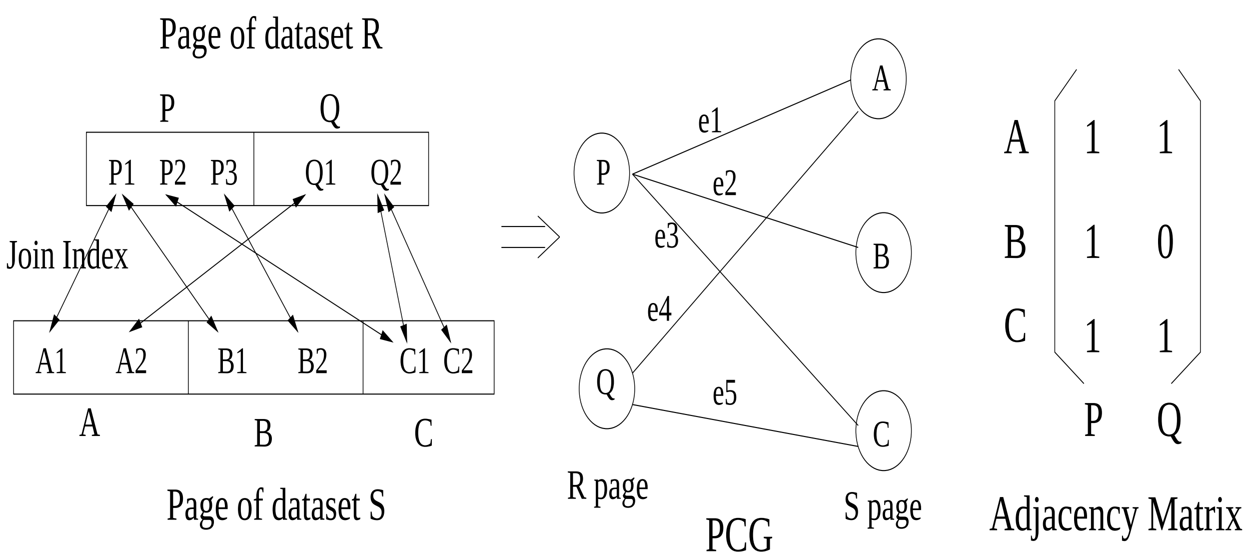
ID	Polygon	Non-Spatial Data
A1	(3,12.2,6.6,16,...)	(...)
A2	(13.4,7.7,19.5,10,..)	(...)
B1	(7.6,12.7,15,5.2,..)	(...)
B2	(15.5,15,20.4,19,..)	(...)
C1	(5.1,8.9,9.1,10.9,..)	(...)
C2	(7.5,2,9.7,15.1,..)	(...)

Join Index

R.ID	S.ID
P1	A1
P1	B1
P2	C1
P3	B2
Q1	A2
Q2	C1
Q2	C2

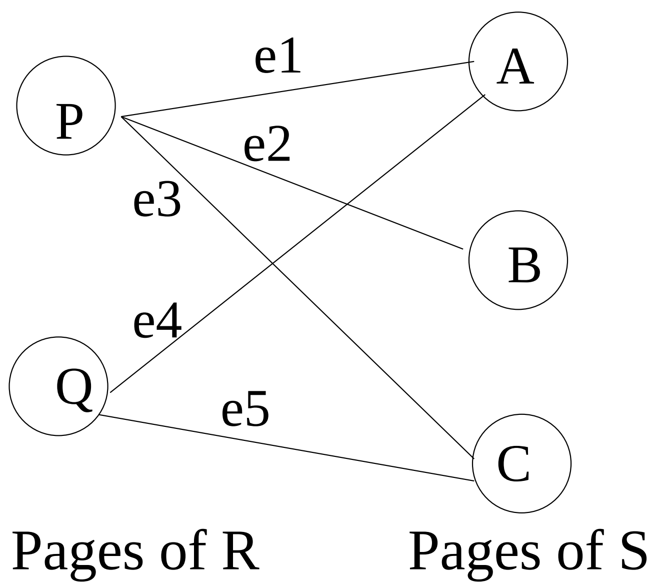
Join index and Page Connectivity Graph (PCG)

- What is PCG
  - *A PCG is a bi-partite graph*
    - Node = (P,Q,A,B,C)
    - Edge = (e1,e2,e3,e4,e5)
  - *A PCG describe the join-index in the form of a graph*
- Construction of PCG



Page Access Sequence for Join-Processing using Join-index

- Example
  - *Assume buffer space can hold two pages of base relations*
  - *Consider the two page access sequence*
    - (P, A, Q, B, P, C, Q): total 7 page accesses
    - (P, A, Q, C, P, B): total 6 page accesses
  - *Optimal number of page accesses = 5*



Problem Definition

- Optimal page access sequence with fixed buffer (OPAS-FB)
  - **Given:** *A page-connectivity graph and a buffer of size B.*
  - **Find:** *A page-access sequence.*
  - **Objective:** *Minimize the number of page accesses*
  - **Constraint:** *Number of pages in buffer* $\leq$ *B.*
- Example
  - *Buffer size = 2*
  - *Page access sequence: P, A, Q, C, P, B*
- OPAS-FB problem is NP-hard.
- Techniques for solving the problem
  - *Cluster all the related pages*
  - *Reuse pages in memory*

Related Work and Our Contribution

- Related Work
  - *Asymmetric*
    - Sorting Based [23]
  - *Symmetric*
    - FP[Fotouhi 89] [6]: Greedy heuristic
    - OM[Omiecinski 89] [20]: Greedy heuristic
    - Chan[Chan 97] [4]: Greedy heuristic
- Our Contributions
  - *Asymmetric*
    - AC - uses clustering on one relation
    - Lemma to characterize its I/O cost
    - Experiment Evaluation - Dominates Sorting based approach
  - *Symmetric*
    - SC - A global clustering approach on both relations
    - Lemma to characterize its I/O cost
    - Experiment Evaluation - Dominates other methods for small memory size

Overview

- Motivation, Basic Concept, Problem Formulation
- Asymmetric Clustering $\Leftarrow$
- Comparison of Asymmetric Approaches
- Symmetric Clustering
- Comparative Evaluation of SC, AC, and Competitors
- Conclusion and Future Work

Asymmetric Clustering(AC): Motivation

- Sorting method[Valduriez 87] [23]
  - *Read in as much as possible of the primary relation(R)*
  - *Sort the connected S nodes to reduce redundant access*
  - *No redundant on primary relation*
  - *Redundant access to the second relation*
- AC method
  - *Cluster pages of the primary relation(R)*
  - *Read in as much as possible of the primary relation(R)*
  - *No redundant on primary relation*
  - *Reduce redundant access to the second relation*

Comparison of AC and Sorting: One example

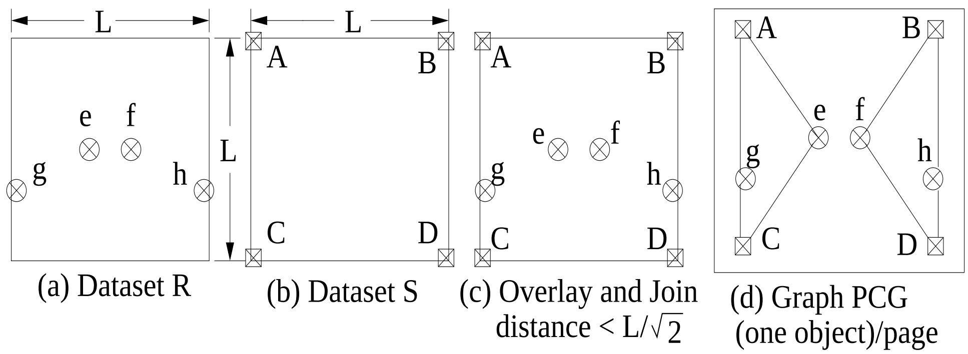


Figure 1: Spatial join for the two data set

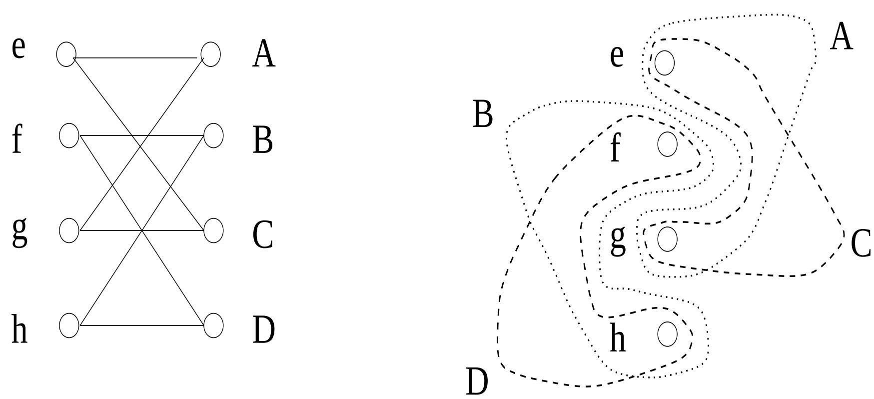


Figure 2: Construction of a one-side hyper graph from the data set

- *Loading Sequence*
  - Buffer size = 3
  - Sorting: e, f, A, B, C, D, g, h, A, B, C, D
  - AC: e, g, A, C, f, h, B, D
  - AC reduces the redundant I/Os from 4 to zero

Comparison of AC and Sorting: One example

- 49 nodes PCG
  - *R relation pages : 1-24*
  - *S relation pages : A-Y*
- Page size: 7

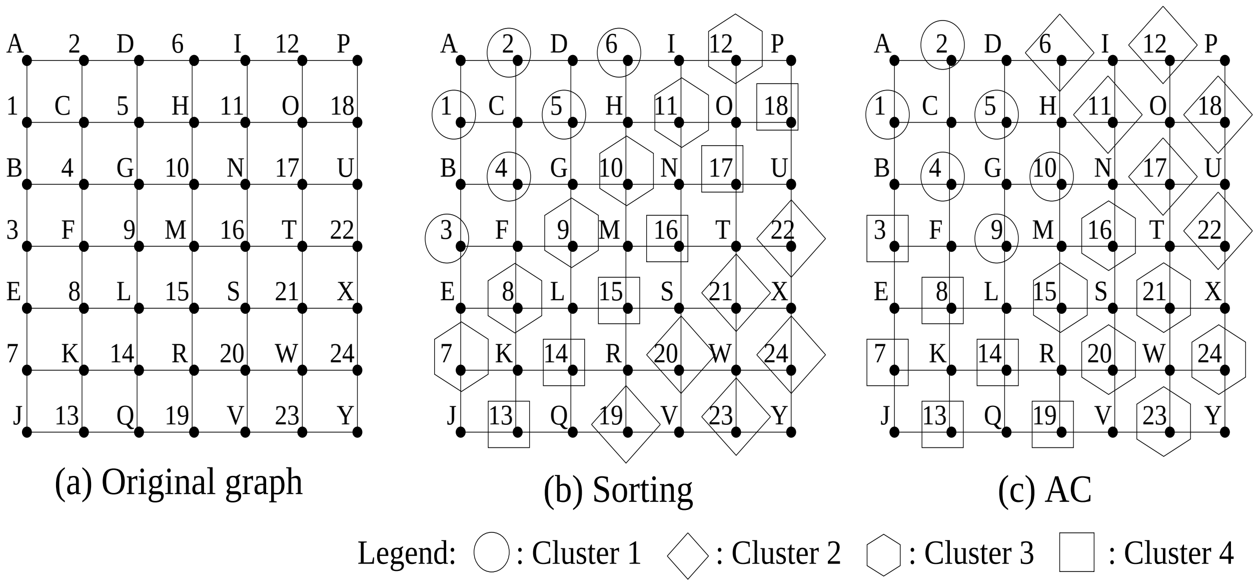


Figure 3: Example of the AC and Sorting heuristic

	Sorting Heuristic		AC	
Round	First Six Buffers	The 7th buffer	First Six Buffers	The 7th buffer
1	1,2,3,4,5,6	A,B,C,D,E,F,G,H,I	1,2,4,5,9,10	A,B,C,D,F,G,H,L,M,N
2	7,8,9,10,11,12	E,F,G,H,I,J,K,L,M,N,O,P	3,7,8,13,14,19	B,E,F,J,K,L,Q,R,V
3	13,14,15,16,17,18	J,K,L,M,N,O,P,Q,R,S,T,U	6,11,12,17,18,22	D,H,I,N,O,P,T,U,X
4	19,20,21,22,23,24	Q,R,S,T,U,V,W,X,Y	15,16,20,21,23,24	L,M,N,R,S,T,V,W,X,Y
Total I/O	**66**		**62**	

Table 1: Loading Sequence of Sorting and APG for the above figure

AC Algorithm

Input: $G = (V_r, V_s, E)$ is a page connectivity graph

Output: $S =< P_1, P_2, ..., P_r >$ is a page access sequence

$HG_r(V_r, HE_r)$ = DeriveHypergraph(G);

$PSet_r$ = hMetis-Partition($HG_r, B - 1$) [13]

i=0;

while ((P_{i_r}=SelectUnprossedPartition($PSet_r$))!=NULL)

{

  AddPageSequence(S, P_{i_r});

  P_{i_s} = Sort-Eliminate-Dup(G, P_{i_r});

  AddPageSequence(S, P_{i_s});

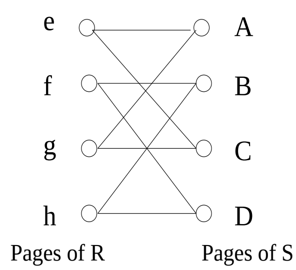
  P_{i_r}.flag = "processed";

  i++;

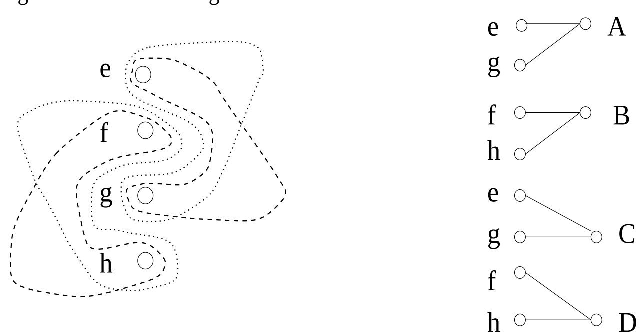
}

Trace of AC algorithm: an example

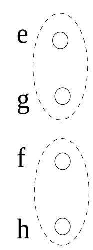
Step 1. G =



Step 2. HG =

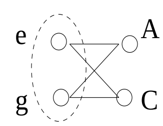


Step 3. hMetis-Partition(HG,2) PSetr =



Step 4. AddPageSequence(S, Pir) S=(e, g)

Step 5. AddPageSequence(S, Pis) S=(e, g, A, C)



Step 6. AddPageSequence(S, Pir) S=(e, g, A, C, f, h)

Step 7. AddPageSequence(S, Pis) S=(e, g, A, C, f, h, B, D)

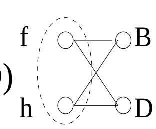
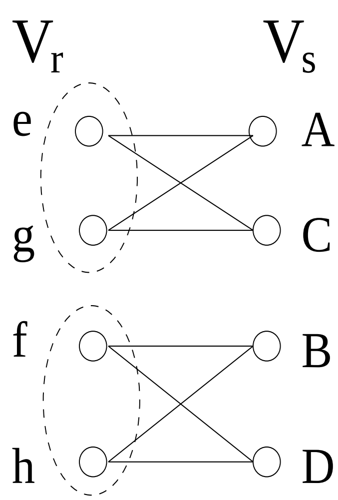


Figure 4: Trace of Algorithm

I/O cost of AC

- Lemma
  - **Given**: *Page-connectivity graph* $PCG = (V_r, V_s, E)$*, and partition* $\{P_{r_1}, P_{r_2}, \ldots, P_{r_p}\}$ *of* V_r
  - **I/O Cost**: $K = |V_r| + \sum_{v \in V_s} f(v)$
  - **Where**: $f(v)$ *denotes the number of partitions of* V_r *that have an edge to node* v *in* V_s

- Proof
  1. *A node* v *in* V_s *is connected to* $f(v)$ *partitions of* V_r
  2. *The node* v *in* V_s *has to be loaded* $f(v)$ *times to compute the join*
  3. *Total I/O cost:* $|V_r| + \sum_{v \in V_s} f(v)$



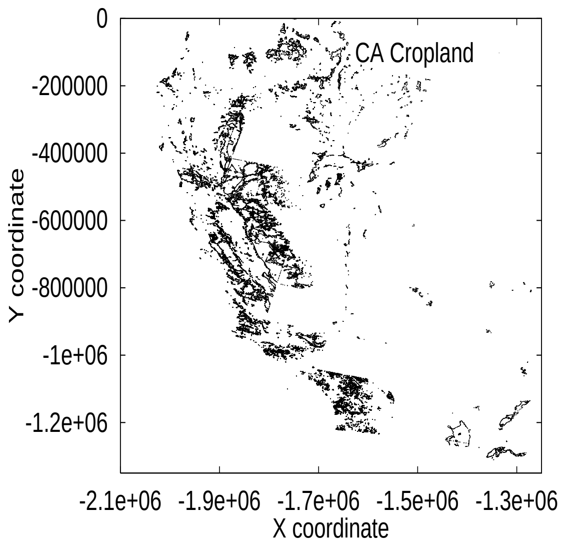
Overview

- Motivation, Basic Concept, Problem Formulation
- Asymmetric Clustering(AC)
- Comparison of Asymmetric Approaches $\Leftarrow$
- Symmetric Clustering(SC)
- Comparative Evaluation of SC, AC, and Competitors
- Conclusion and Future Work

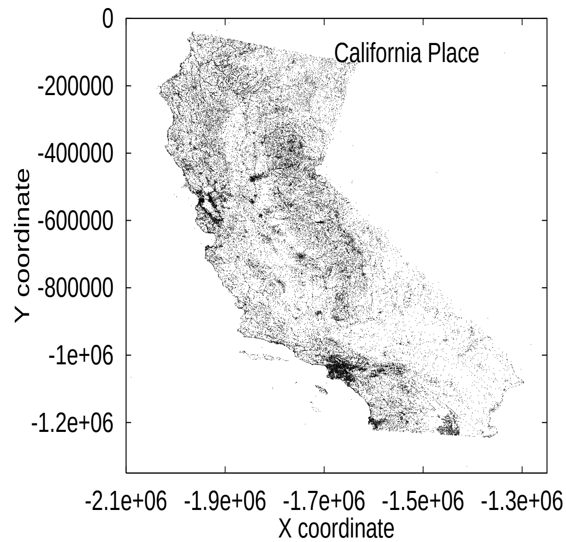
Experiment design

- Asymmetric Methods
  - *Sorting, AC*
- Parameters change
  - *Page size*
  - *Buffer size*
  - *Edge Ratio*
    - Give a join graph $G = (V_R, V_S, E)$, the edge ratio $\Theta = \frac{|E|}{|V_R||V_S|}$
    - The edge ratio provides a measure of the page-level join selectivity
- Data set
  - *Sequoia 2000*
  - *Points: 62,584 California place names + locations*
  - *Polygons: 4388 records, Crop landuse in CA*
- Join Index
  - *For each place in the* $Point$ *table, find all croplands which are within a distance* **D**
  - *i.e. (Place i, Cropland j) within distance* **D**

Experiment design(Cont.)



(a) Polygons for Landuse



(b) GNIS Points

Figure 5: Example of the Sequoia 2000 data set

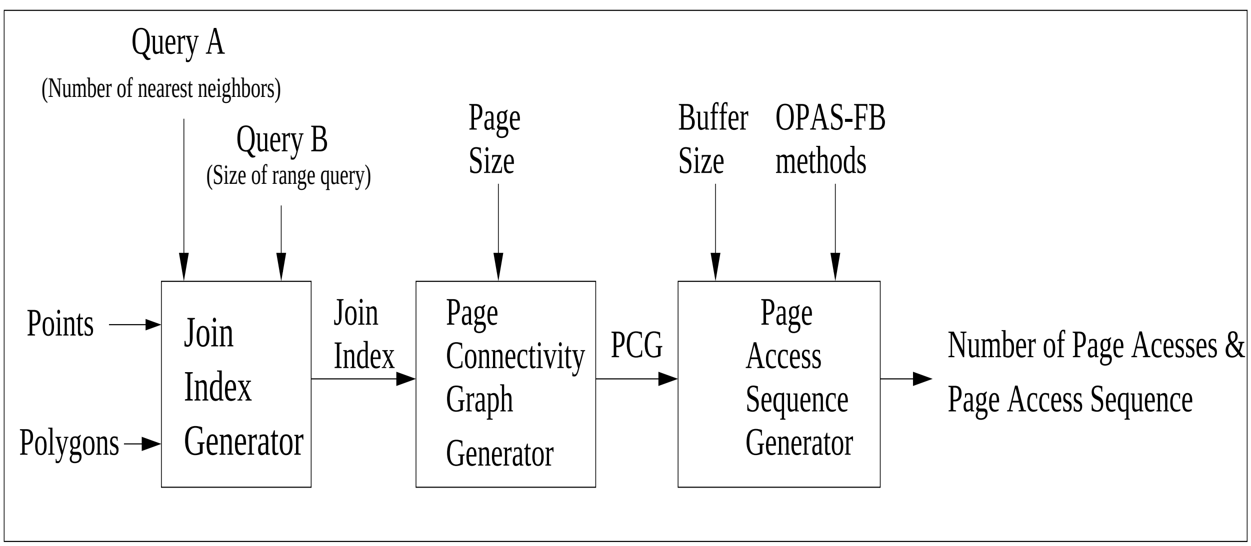
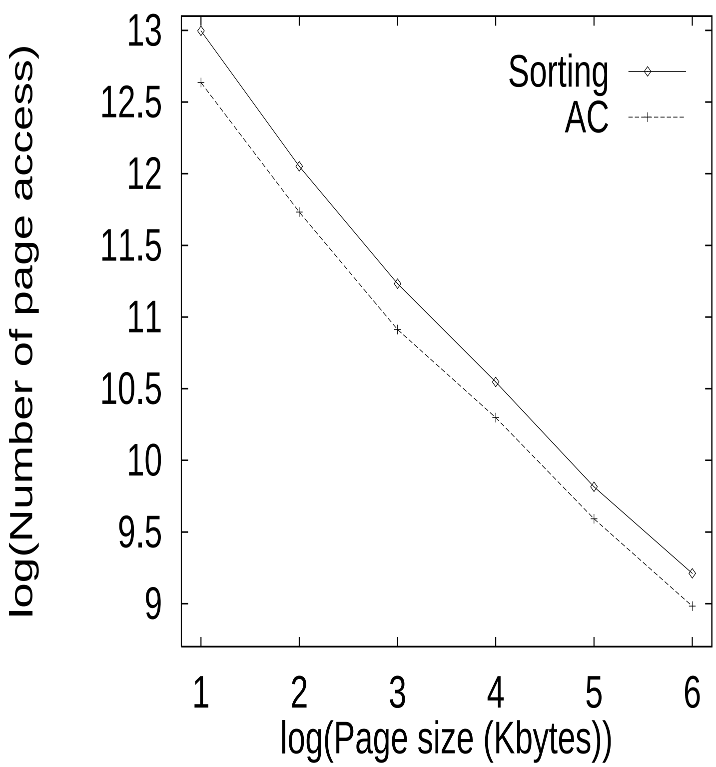


Figure 6: Experimental setup and design.

Effect of Page Size

- Fixed Parameters
  - *Buffer Size: 10% of the smaller relation*
  - *Range Query Size: 3800 units (extent* $\cong 10^6$ *units)*
- Variable Parameters
  - *Page size: 2k to 64k*
  - *OPAS-FB heuristics: Sorting, AC*

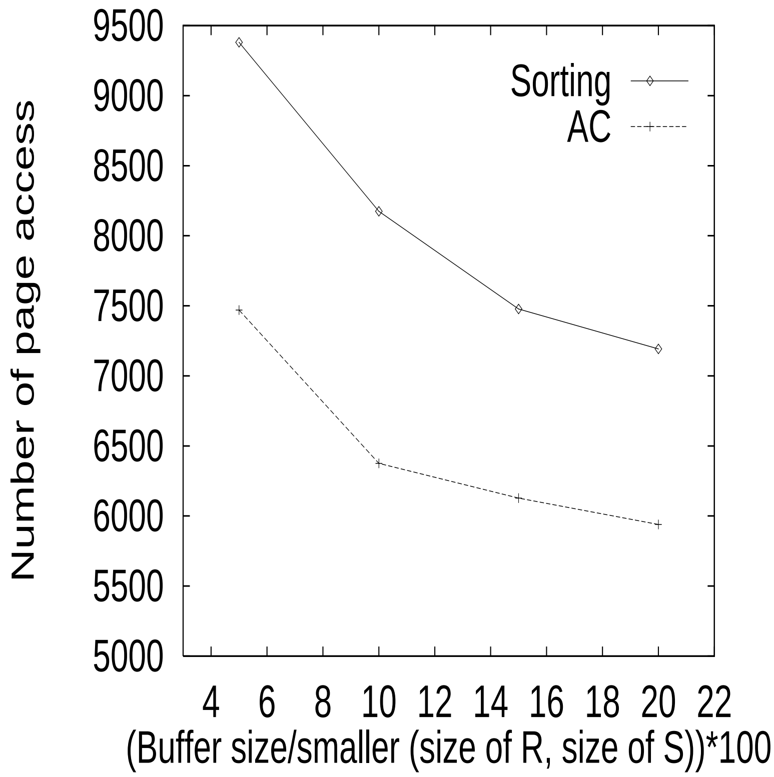


- Trend
  - *The AC method is uniformly better than the Sorting heuristic*

Effect of Buffer Size

- Fixed Parameters
  - *Page size = 2k*
  - *Range query size = 3800 units(Θ=0.003)*

- Variable Parameters
  - *Buffer size: 5% to 20% of the smaller relation*
  - *OPAS-FB heuristics: Sorting, AC*

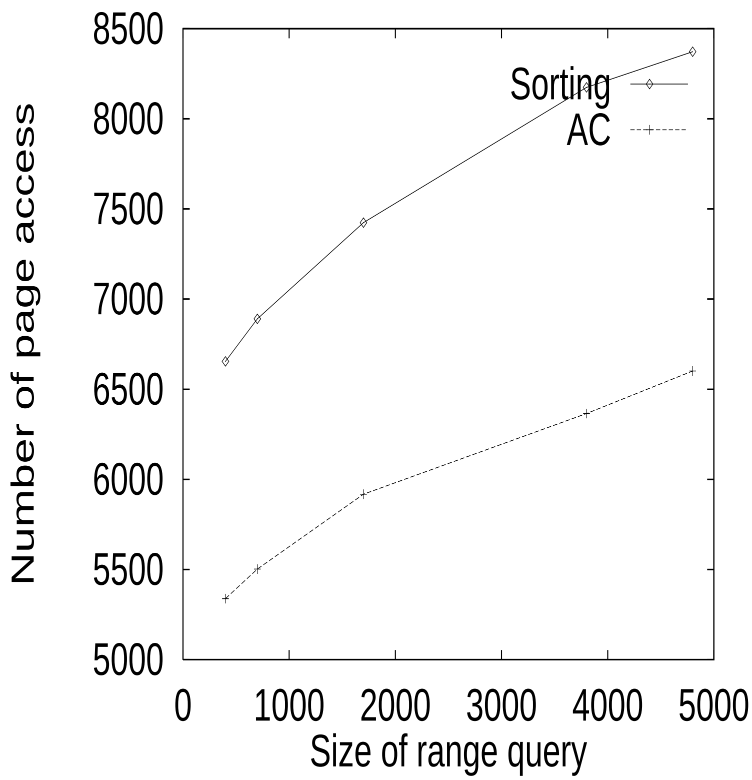


- Trend
  - *The AC method is uniformly better than the Sorting heuristic*

Effect of Edge Ratio(Range Query)

- Fixed Parameters
  - *Page size = 2k*
  - *Buffer size = 5% of the smaller relation*

- Variable Parameters
  - *Range query size = 400-4800 units(Θ=0.002-0.003)*
  - *OPAS-FB heuristics: Sorting, AC*



- Trend
  - *The AC method is uniformly better than the Sorting heuristic*

Overview

- Motivation, Basic Concept, Problem Formulation
- Asymmetric Clustering
- Comparison of Asymmetric Approaches
- Symmetric Clustering $\Leftarrow$
- Comparative Evaluation of SC, AC, and Competitors
- Conclusion and Future Work

Motivation for Symmetric Method

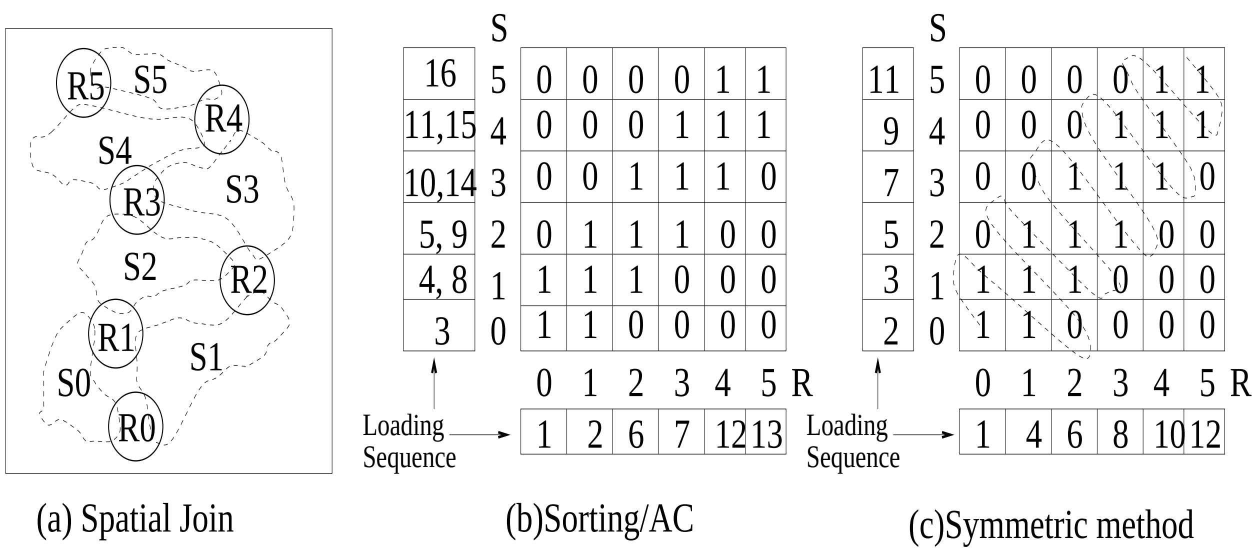


Figure 7: Comparison of symmetric and asymmetric methods

- Example: Blocking factor = 1 for simplicity
- Buffer size: 3
- Sorting/AC method
  - *Loading sequence: R0,R1,S0,S1,S2,R2,R3,S1,S2,S3,S4,R4,R5,S3,S4,S5*
  - *Redundant I/Os: S1,S2,S3,S4*
- Symmetric method
  - *Loading sequence: R0,S0,S1,R1,S2,R2,S3,R3,S4,R4,S5,R5*
  - *No Redundant I/O*

Related Work

- FP method[Fotouhi 89] [6]
  - *Add the largest resident degree node to buffer*
- OM method[Omiecinski 89] [20]
  - *Choose the next page, p*
    - Find a node q in the buffer, with smallest non-resident degree
    - Find the node p, such that exist pq edge, and with smallest non-resident degree
- Chan method[Chan 97] [4]
  - *In each iteration, select the smallest segment with the smallest non-resident degree node*
  - *Within the iteration, select the node with the largest resident degree*
- Symmetric Method
  - *FP, OM, Chan: incremental, consider only local information in PCG*
  - *SC: exploit global information across entire PCG*

Symmetric Clustering(SC) Algorithm

Input: $G = (V, E)$ is a page connectivity graph, B is the number of buffers.

Output: $S =< P_1, P_2, ..., P_r >$ is a page access sequence with $r \geq |V|$.

$< G_{BD}, P_{order} >$=Band-diagonalize(G, B);

$< G_{OBD} >$=Get-off-B-diagonal-entries(G_{BD});

$< VC >$=Find-vertex-cover(G_{OBD});

$< S >$=Access-sequence-generator(P_{order}, G_{BD}, VC);

Band-diagonalize

- Define
  - *B-diagonal edge([i,j],B) iff* $|i - j| \leq \lfloor B/2 \rfloor$ *for square matrix*
- Band-diagonalize(Matrix M)
  - *Reorder rows and columns of M to maximize B-diagonal edges*
  - *General Problem: Envelop reduction [1, 8, 15]*
- Indirect method
  - *(a)Partition e.g. min-cut graph partition*
  - *(b)Order partition to improve band*
  - *Heuristic approach*
  - *Has room to improve, but*
  - *Already outperformed competitors for OPAS-FB*

Band-diagonalize using grpah-partition and partition ordering

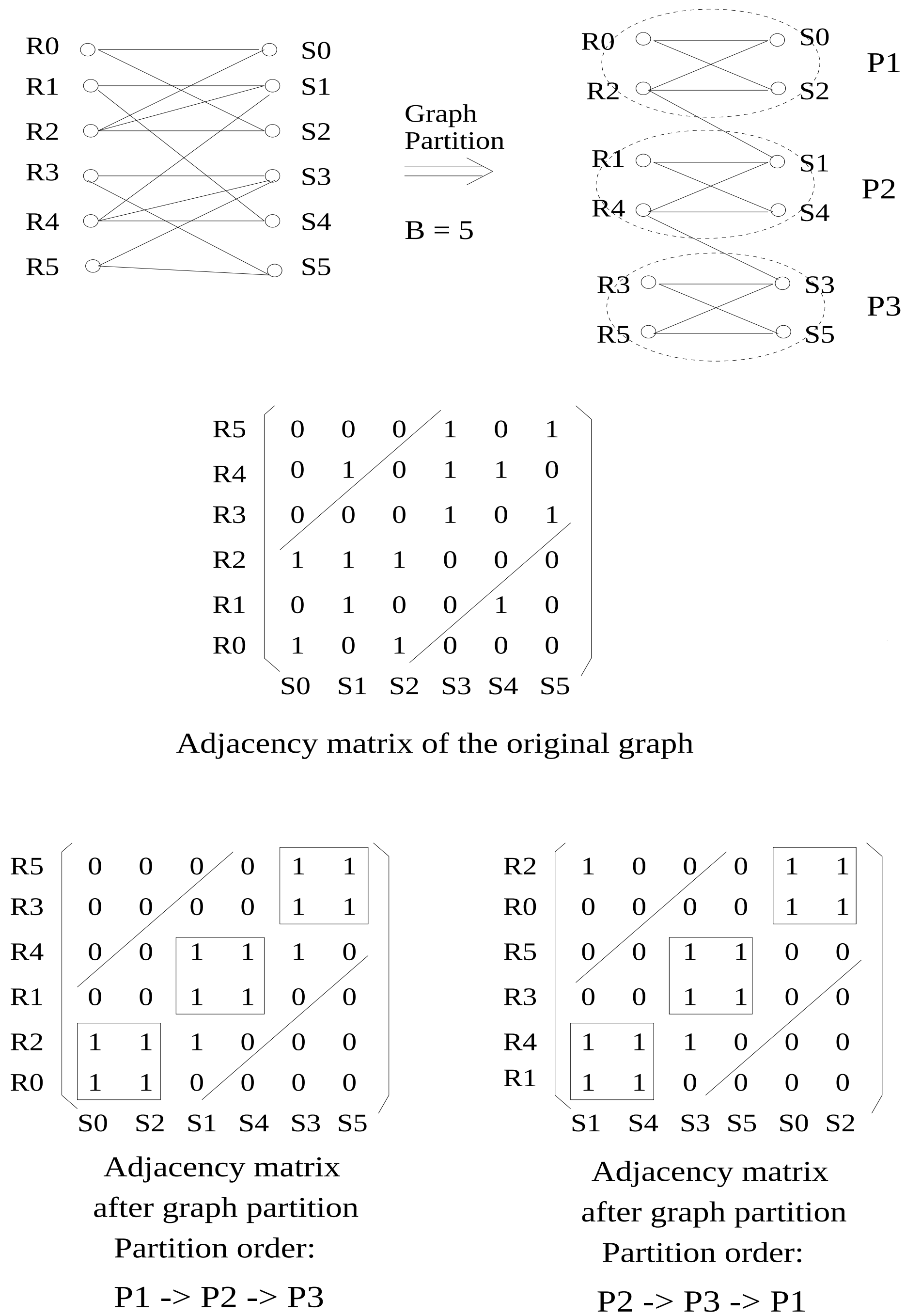


Figure 8: Band-diagonalize using grpah-partition and partition ordering

Band-diagonalization - indirect method

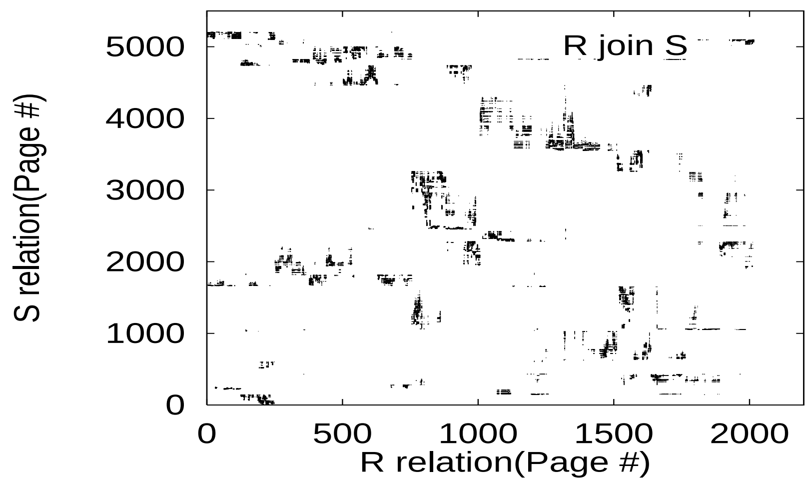


Figure 9: The original PCG adjacency matrix

- Buffer size = 100
- After Graph-Partition

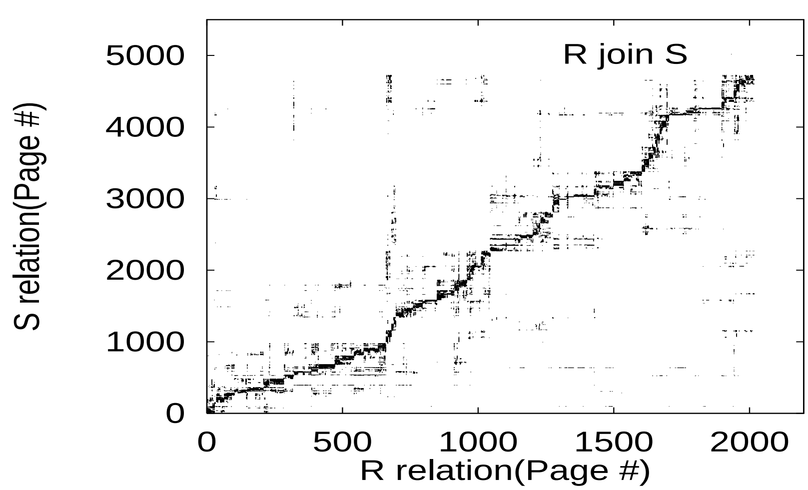


Figure 10: B-diagonalized matrix(graph-partition), 28% edges outside B-diagonal

Band-diagonalization - indirect method(Cont.)

- After Reordering Partitions

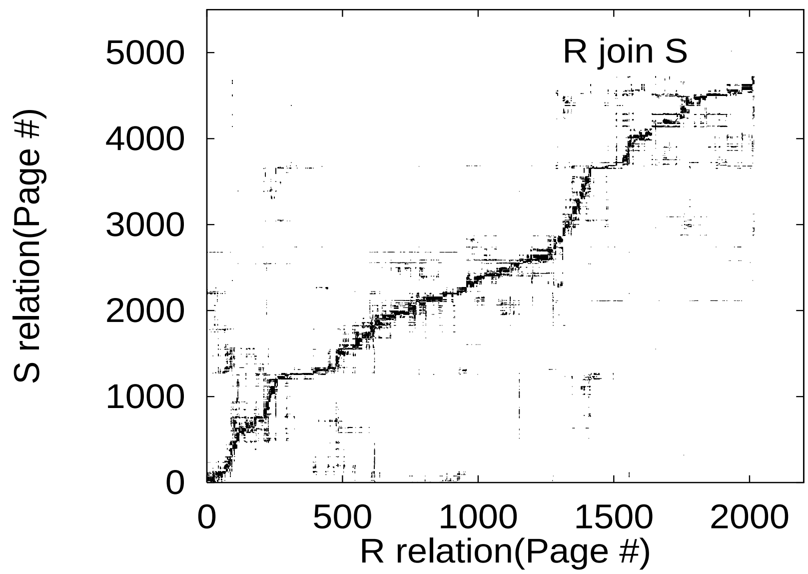


Figure 11: After ordering partitions, 22% edges outside B-diagonal

Band-diagonalize

Band-diagonalize

Input: G is a page connectivity graph, B is the number of buffers.
Output: G_{BD} is the B-diagonal connectivity graph, P_{order} is the partition order

$PSet_m$ = Graph-Partition($G, B-1$); /* *Using graph partition software* */
P_{order} = Order-Partition($PSet_m$); } /* *Order the partitions using the greedy heuristic* */

- Graph-partition
  - **Given**: *A connectivity graph* $G = (V, E)$ *with* $|V| = n$, *and the number of buffers,* $B \geq 2$.
  - **Find**: *A partition of* V *into* p *subsets,* $V_1, V_2, \ldots, V_p$ *such that* $V_i \bigcap V_j = \emptyset$ *for* $i \neq j$ *and* $\bigcup_i V_i = V$.
  - **Objective**: *Minimize the size of the set of edges* $E_C \subseteq E$ *whose incident vertices belong to different subsets.*
  - **Constraint**: $|V_i| \leq (B-1)$, *and the number of partitions,* $p = \lceil |V|/(B-1) \rceil$.

- Order-Partition

1. *Build a partition-interaction matrix* M
2. *An entry* $M[P_i, P_j]$ *denotes the number of cut-edges between the nodes in partition* P_i *and* P_j
3. *Sort the entries* $M[P_i, P_j]$ *in descending order*
4. *Choose the entry with the largest value*
5. *Extend the loading sequence on both sides in a greedy manner*

Band-diagonalization: a simple example

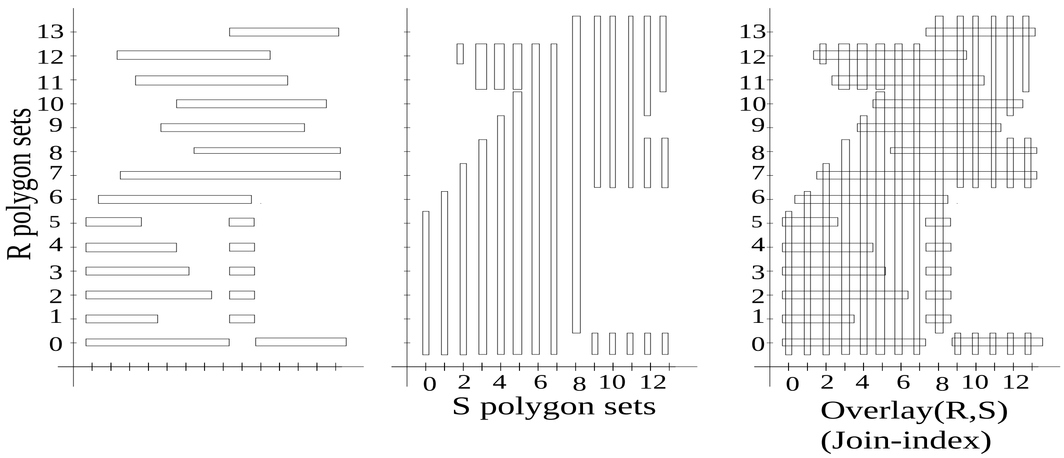


Figure 12: Overlay of Two Relations

$$
\begin{array}{r|cccccccccccccc|}
13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
12 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
10 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
9 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
7 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
6 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
5 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
4 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
3 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
\text{R} & & & & & & & & & & & & & & \\
\text{S} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13
\end{array}
$$

(a) Adjacency matrix of PCG

Redundant I/O

$$
\begin{array}{r|cc|ccccc|cc|ccccc|l}
 & \text{P1} & & \text{P2} & & & & & \text{P3} & & \text{P4} & & & & & \\
13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & \text{P4} \\
12 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
\hline
11 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
10 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & \\
9 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & \text{P3} \\
8 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \\
7 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \\
\hline
6 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \text{P2} \\
5 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & \\
\hline
4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \\
3 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \\
2 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \text{P1} \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \\
\text{R} & & & & & & & & & & & & & & & \\
\text{S} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 &
\end{array}
$$

(b) Adjacency matrix of PCG after Band-diagonalize (B=8)

Figure 13: Effect of Band-diagonalize using graph partitioning

Vertex Cover Computation

- Vertex Cover: A set of vertices that cover all the edges in the graph
- Vertex Cover problem: Find a vertex cover of minimum size in a given graph
- NP-hard for general graph [7], but not for bi-partite graph [11]
- Algorithm for Vertex Cover Computation: Heuristic approach

Input: $G_{OBD} = (V_{OBD}, E_{OBD})$ is the off-B-diagonal graph

Output: VC is the vertex cover for G_{OBD}

```
while(E_OBD is not empty) {
    V_highest=Find_highest_degree_node(V_OBD);
    /* Find the highest degree node V_highest */
    VC = VC ⋃ V_highest;
    /* Add this node to the set of vertex cover */
    Update(G_OBD);
    /* Remove node V_highest and its corresponding edges */
}
```

Vertex cover: an example

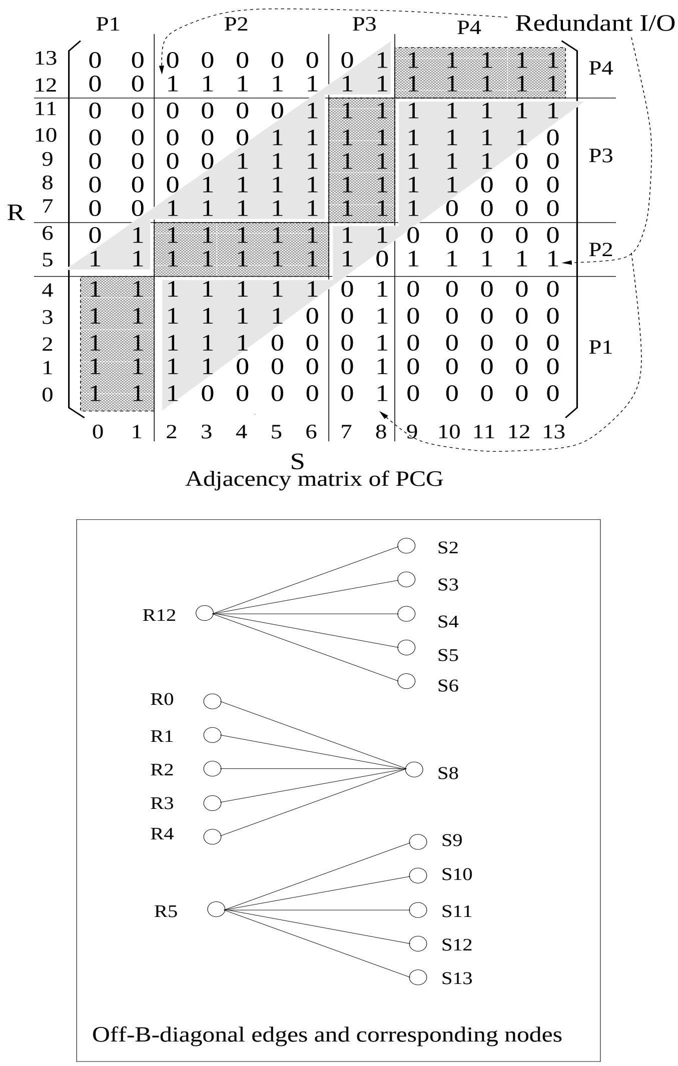


Figure 14: Find the vertex cover for all the off-B-diagonal edges

- Vertex cover for all the off-B-diagonal edges
  - *R5,R12,S8*

Generate Access Sequence

Input: P_{order} is the loading sequence of the partitions,

G_{BD} is the B-diagonal connectivity graph,

VC is the vertex cover for all the Off-B-diagonal edges.

Output: $S =< P_1, P_2, ..., P_r >$ is a page access sequence.

```
for(i = 1; i ≤ |P_order|; i++){
    P_i = GetPartition(P_order, i)
    if(i==1) {
        AddPageSequence(S, P_i);
    else {
        OrderAndAddPageSequence(S, P_{i-1}, P_i);
    }
    PVC_Set = FindConnected_node_from_VC(P_i, VC);
    AddPageSequence(S, PVC_Set);
}
```

Access sequence generator: an example

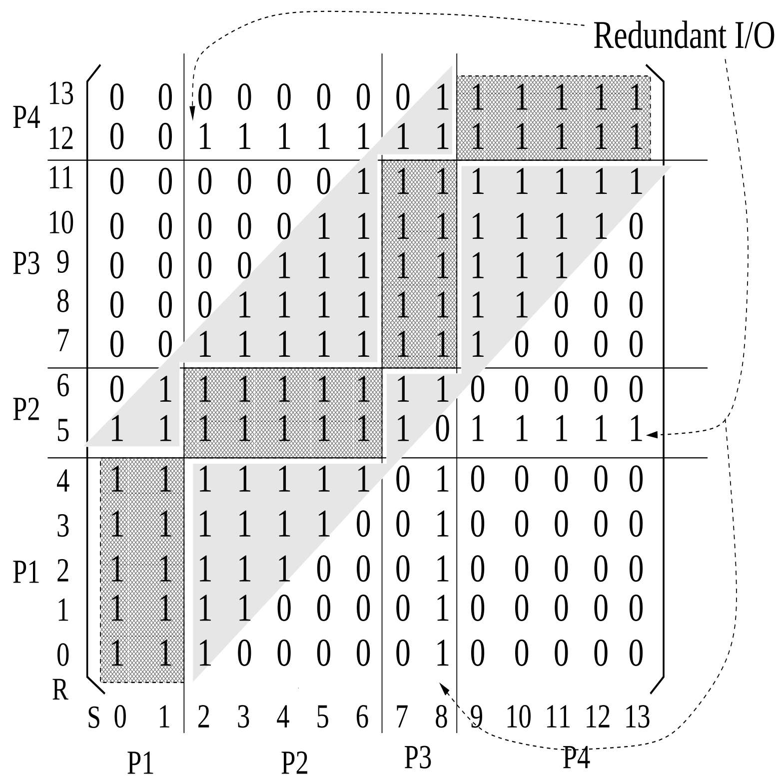


Figure 15: Generate the access sequence

- Loading Sequence
  - *Round 1(Partition 1): R0 R1 R2 R3 R4 S0 S1 S8(VC)*
  - *Round 2(Partition 2): S2 S3 S4 S5 S6 R5 R6 R12(VC)*
  - *Round 3(Partition 3): R7 R8 R9 R10 R11 S7 S8*
  - *Round 4(Partition 4): R12 R13 S9 S10 S11 S12 S13 R5(VC)*

Comparison for all Symmetric methods

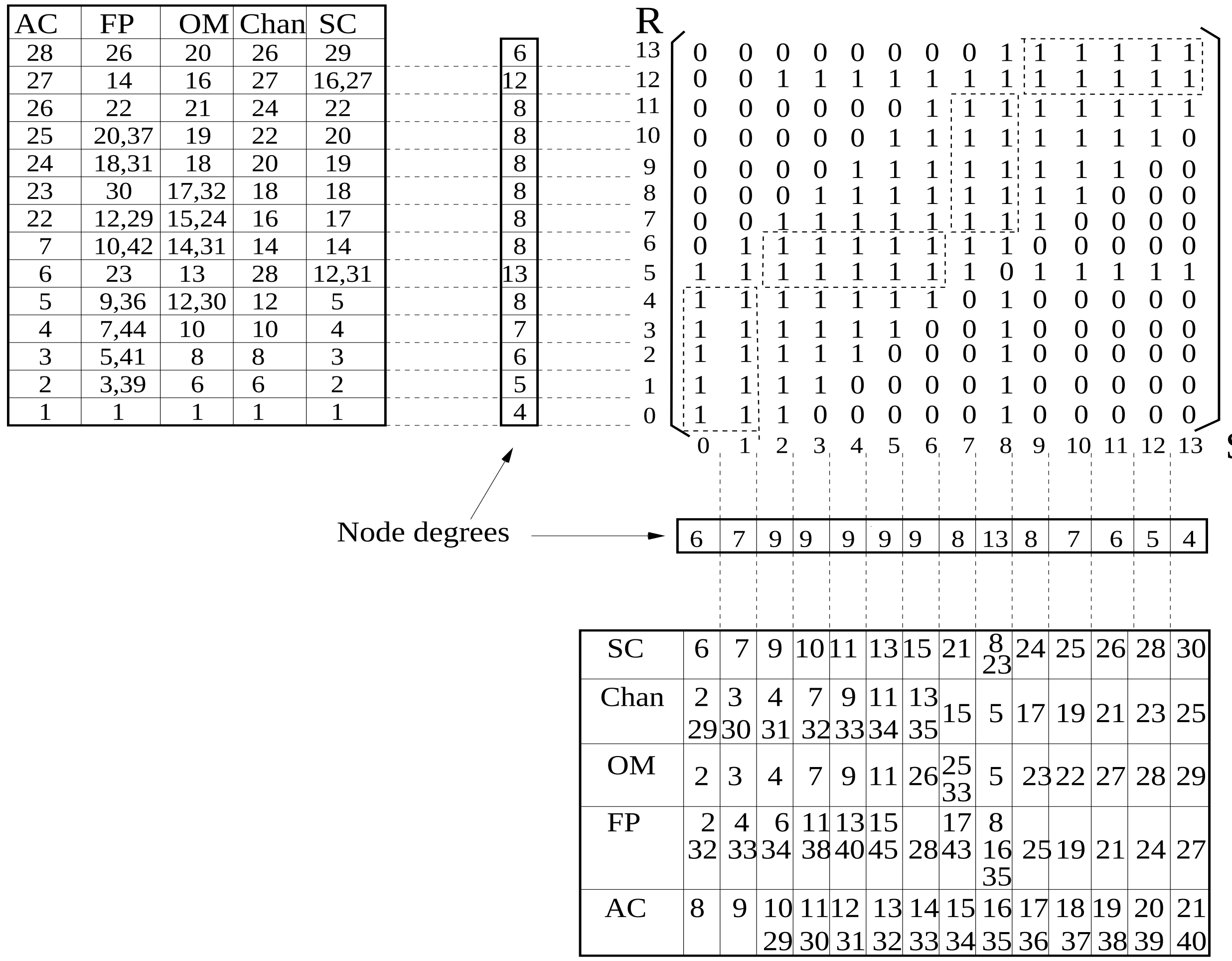


Figure 16: Comparison of different methods

Method	AC	FP	OM	Chan	SC
Total I/O	**40**	**45**	**33**	**35**	**31**

Table 2: Summary of page accesses for different algorithms

The I/O cost of SC method

- Lemma
  - **Given**: *Page-connectivity graph* $PCG = (V, E)$*, and ordered partition* $\{P_1, P_2, \ldots, P_n\}$
  - **I/O Cost**: $K = |V| + \sum$ *Partition-degree*(V_i)
  - **Where**: $V_i \in$ *vertex cover of outside B-diagonal edges*
- Proof
  1. *All of the edges within the B-diagonals can be processed without redundant I/O*
  2. *The redundant I/Os come from the vertex cover of the off-B-diagonal edges*
  3. *Each node in the vertex cover has to be loaded once for each connected partition*
  4. *Total I/O Cost:* $K = |V| + \sum$ *Partition-degree*(V_i)

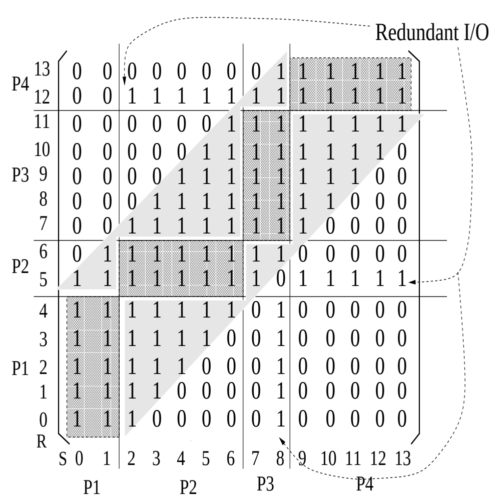


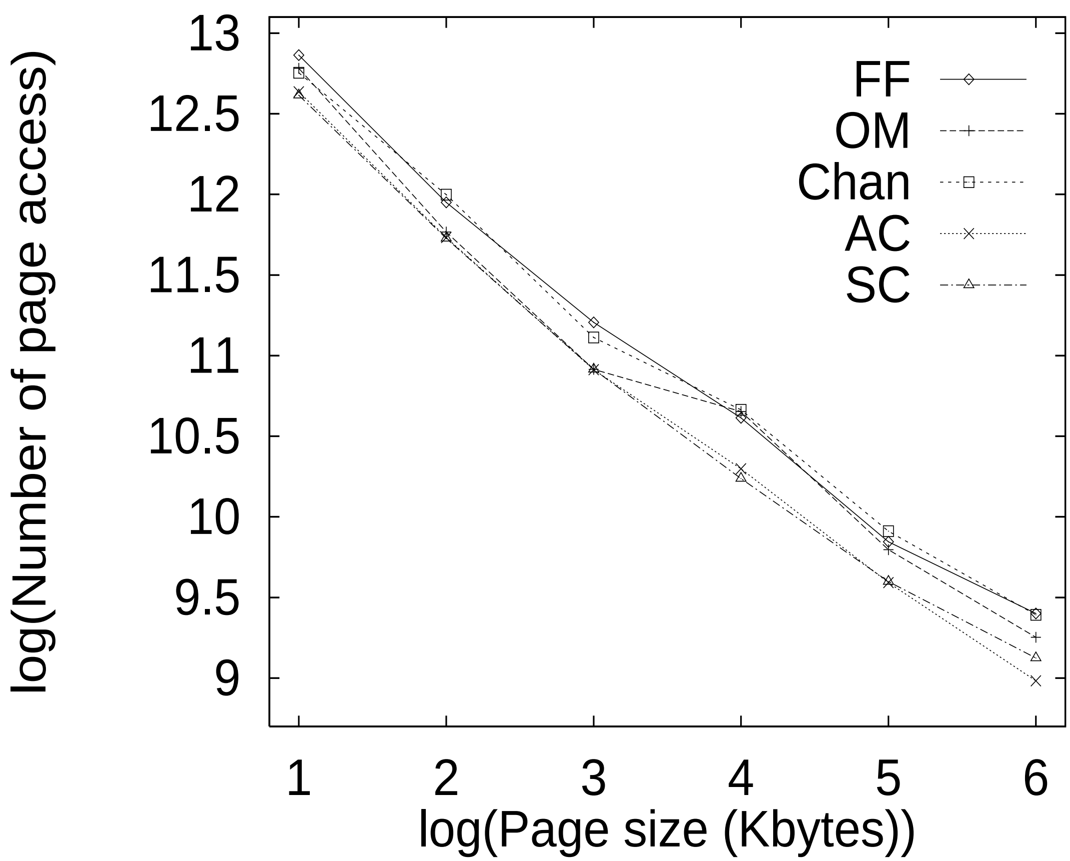
Figure 17: Generate the access sequence

Overview

- Motivation, Basic Concept, Problem Formulation
- Asymmetric Clustering
- Comparison of Asymmetric Approaches
- Symmetric Clustering
- Comparative Evaluation of SC, AC, and Competitors $\Leftarrow$
- Conclusion and Future Work

Effect of page size

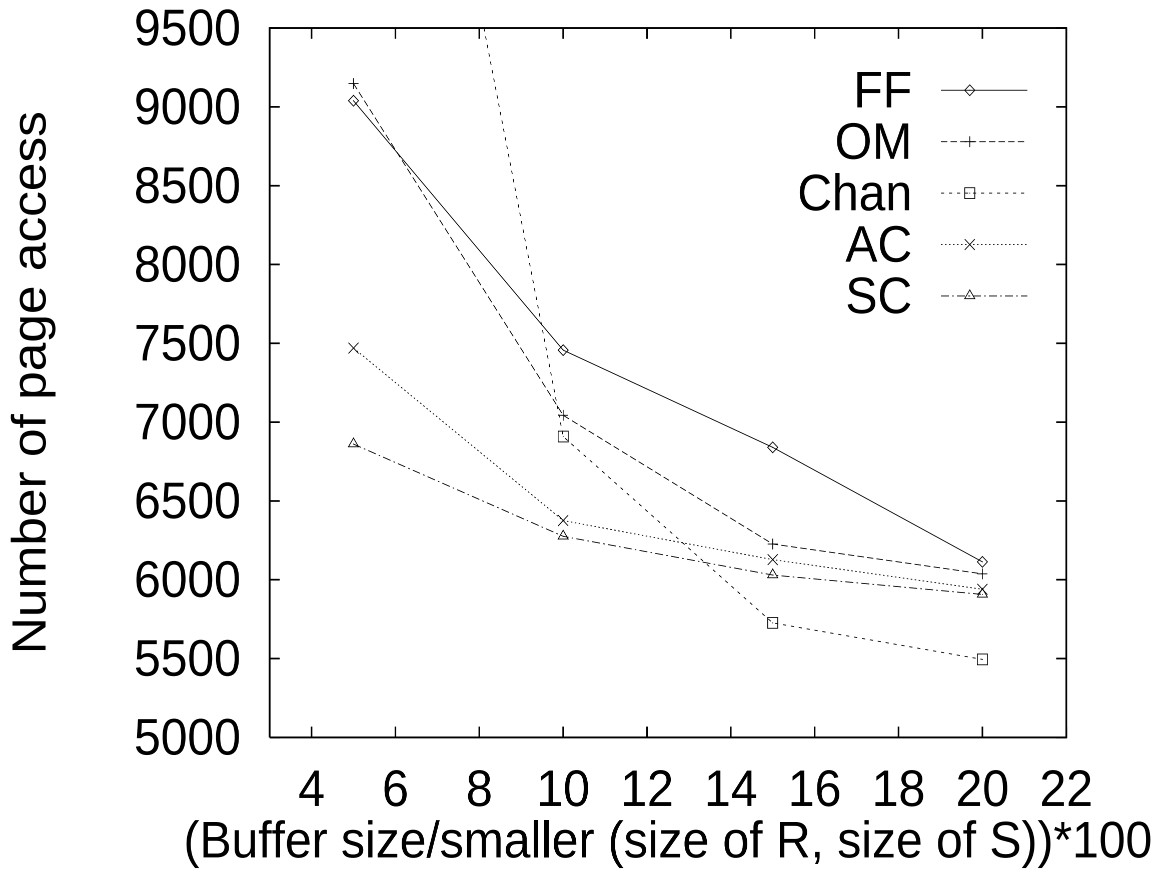
- Fixed Parameters
  - *Buffer Size: 10% of the smaller relation*
  - *Range Query Size: 3800*
- Variable Parameters: Page size (2k to 64k)



- Trend
  - *The AC outperforms SC when page size is greater than 64 kbytes*
  - *The OM's method has good performance with 4 and 8 kbyte page size*

Effect of Buffer Size

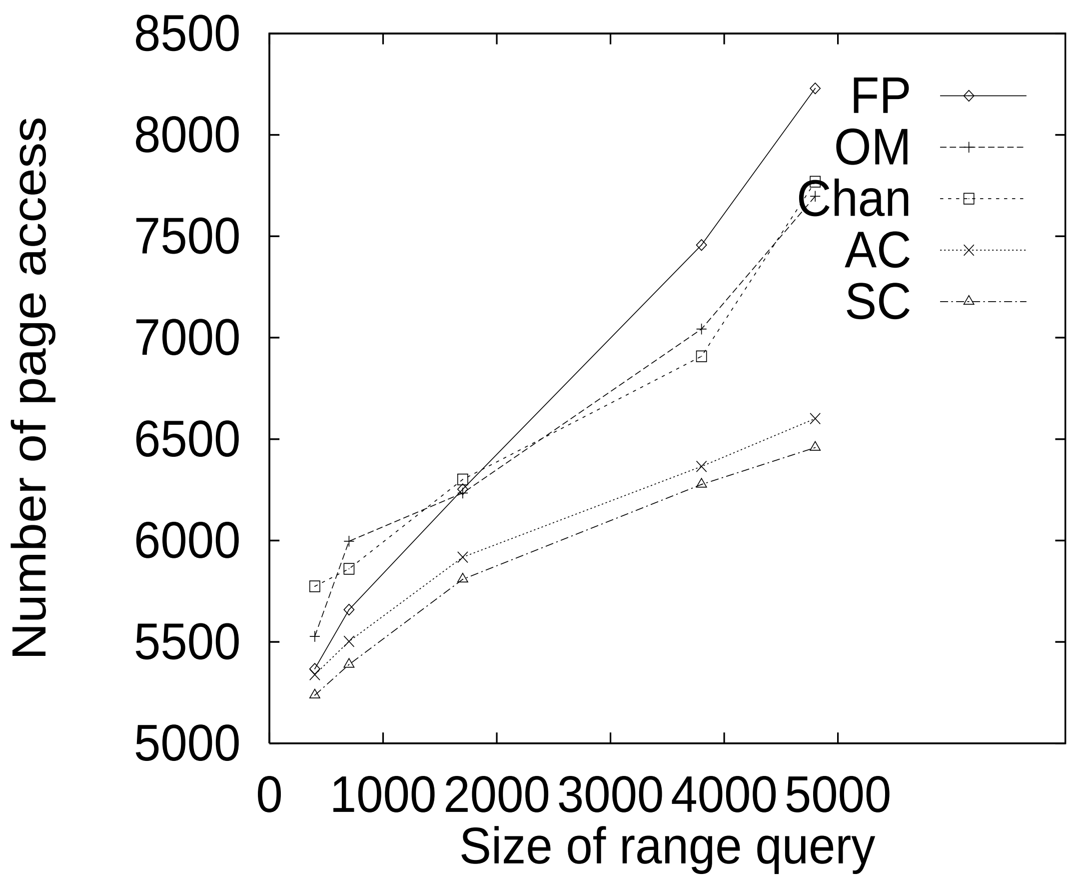
- Fixed Parameters
  - *Page size = 2k*
  - *Range query size = 3800 (Θ=0.003)*
- Variable Parameters: Buffer size (5% to 20% of the smaller relation)



- Trend
  - *AC and SC perform better than all of the other methods when the buffer size is relatively low, e.g. 5 to 10 percent*
  - *Chan and OM do well with a large buffer size*

Effect of Edge Ratio

- Fixed Parameters
  - *Page size = 2k*
  - *Buffer size = 5% of the smaller relation*
- Variable Parameters: Range query size (400-4800) (Θ=0.002-0.003)



- Trend
  - *AC and SC outperform all of the other methods uniformly*
  - *SC has fewer page accesses than AC*

Overview

- Motivation, Basic Concept, Problem Formulation
- Asymmetric Clustering
- Comparison of Asymmetric Approaches
- Symmetric Clustering
- Comparative Evaluation of SC, AC, and Competitors
- Conclusion and Future Work $\Leftarrow$

Conclusion and Our Contribution

- Conclusion
  - *The proposed AC and SC algorithms outperform the traditional methods when*
    - The memory buffer is small($<$10%), relative to the size of the spatial relations
    - In small memory buffer, the tradition methods generate redundant I/Os in swap in/out

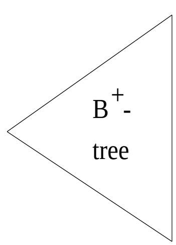
- Our Contribution
  - *Propose two new algorithms for spatial join computation given a join index*
  - *Key idea: Use clustering*
  - *Provide a formal characterization of the I/O upper bound needed by AC and SC*

Future Work

- Explore the usefulness of AC and SC in spatial data mining
  - *Neighborhood index [24]*

- Why neighborhood index
  - *Typically, R-trees are used to process:*
    - Region queries
    - Nearest neighbor queries
  - *For complex spatial object, retrieving the neighbors of some object is time consuming*
  - *Many SDBS are static*
  - *Not many updates on geographic maps*

- Idea of Neighborhood index
  - *Materialize the relevant neighborhood relations*
  - *Avoid access to the spatial objects*



Object-ID	Neighbor	Distance	Direction	Topology
O1	O2	2.7	southwest	disjoint
O1	O3	0	northwest	overlap
O1	O6	3.7	south	disjoint
....	....	....	....	....

Figure 18: Neighborhood Index

Future Work

- Explore the usefulness of AC and SC in data warehouses
  - *Processing star-joins using STARindex*
  - *It is helpful to build join indexes between*
    - Keys of the dimension table
    - Corresponding foreign keys of the fact table
  - *Bitmap join index can solve this problem*
  - Problem of bitmap join index: Sparsity

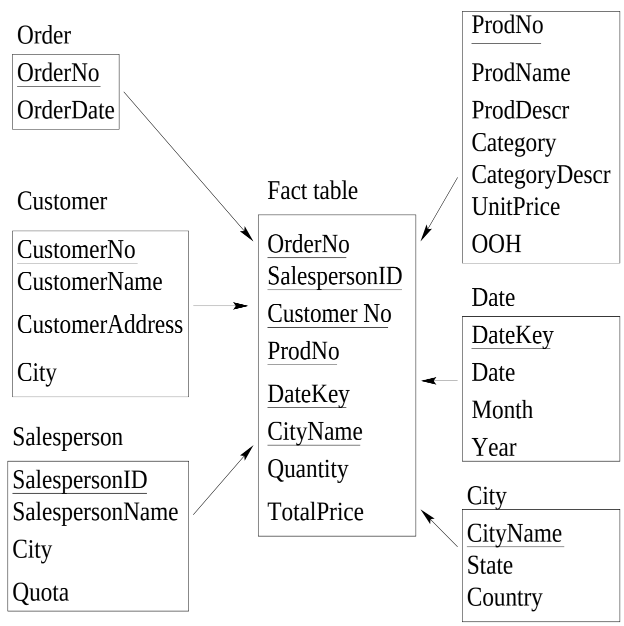


Figure 19: Star Join

References

[1] S. T. Barnard, A. Pothen, and H. D. Simon. A Spectral Algorithm for Envelope Reduction of Sparse Matrices. *Numerical Linear Algebra with Applications*, 2:317–334, 1995.

[2] L. Becker, K. Hinrichs, and U. Finke. A New Algorithm for Computing Joins With Grid Files. In *Proceedings of International Conference on Data Engineering*, 1993.

[3] T. Brinkhoff, H. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-trees. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1993.

[4] Chee Yong Chan and Beng Chin Ooi. Efficient Scheduling of Page Access in Index-Based Join Processing. *IEEE Transactions on Knowledge and Data Engineering*, November/December 1997.

[5] M. Ester and J. Sander S. Gundlach, H. Kriegel. Database Primitives for Spatial Data Mining. *Proc. Int. Conf. on Databases in Office, Engineering and Science, Freiburg, Germany*, 1999.

[6] F. Fotouhi and S. Pramanik. Optimal Secondary Storage Access Sequence for Performing Relational Join. *IEEE Transactions on Knowledge and Data Engineering*, September 1989.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1993.

[8] A. George and A. Pothen. An Analysis of Spectral Envelope-Reduction via Quadratic Assignment Problems. *SIAM Journal of Matrix Analysis and its Applications*, 18(3):706–732, 1997.

[9] O. Gunther. Efficient Computation of Spatial Joins. In *Proceedings of International Conference on Data Engineering*, 1993.

[10] R.H. Guting and W. Schilling. A Practical Divide-and-Conquer Algorithm for the Rectangle Intersection Problem. *Information Sciences*, 42:95–112, 1987.

[11] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching of graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[12] Y.W. Huang, N. Jing, and E.A. Rundensteiner. Spatial Joins Using R-Trees: Breadth-First Traversal with Global Optimizations. *Proc. 23rd Intl. Conf. on Very Large Data Bases*, pages 396–405, 1997.

[13] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. hMetis Home Page. *http://www-users.cs.umn.edu/karypis/metis/hmetis/main.html.*

[14] N. Koudas and K.C. Sevcik. Size Separation Spatial Join. *Proc. ACM SIGMOD Conf.*, pages 324–335, 1997.

[15] G. Kumfert and A. Pothen. Two Improved Algorithms for Envelope and Wavefront Reduction. *BIT*, 37(3):001–032, 1997.

[16] M. Lo and C. V. Ravishankar. Spatial Joins Using Seeded Trees. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 209–220, 1994.

[17] M.-L. Lo and C.V. Ravishankar. Spatial Hash-Joins. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 1996.

[18] W. Lu and J. Han. Distance-Associated Join Indices for Spatial Range Search. In *Proceedings of the International Conference on Data Engineering, IEEE*, 1992.

[19] Exploring Crime Mapping. National Inst. of Justice Crime Mapping Research Center. *http://www.ojp.usdoj.gov/cmrc/briefingbook/welcome.html.*

[20] Edward R. Omiecinski. Heuristics for Join Processing Using Nonclustered Indexes. *IEEE Transactions on Software Engineering*, 15, January 1989.

[21] J. M. Patel and D. J. DeWitt. Partition Based Spatial-Merge Join. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 1996.

[22] D. Rotem. Spatial Join Indices. In *Proceedings of International Conference on Data Engineering*, 1991.

[23] P. Valduriez. Join Indices. *ACM Transactions on Database Systems*, pages 218–246, December 1987.

[24] M.C. Wu and A.P. Buchmann. Research Issues in Data Warehousing. *BTW, Ulm*, March 1997.