Similarity-aware Query Processing and Optimization

Yasin N. Silva
Purdue University

*This work was partially supported by NSF Grant IIS-0811954 and by NIH Grant NIGMS U24 GM077905 for the EcoliHub project
Outline

• Motivation
• Similarity-aware Operators
• Implementing Similarity-aware Operators
• Optimizing Similarity-aware Operators
• Performance Evaluation
• Exploiting Similarity-aware Operators
• Future Work
• Many applications scenarios need the support of queries that capture and take advantage of similarities in the data.
  – Record linkage
  – Data cleaning
  – Pattern recognition
  – Multimedia and video applications
  – Bioinformatics
  – Phenomena detection on sensor networks
  – Marketing analysis
Shift from Exact to Similar

- Shift from systems that focus on exact semantics of data and queries to systems that focus on approximate and imprecise semantics.

- Areas driving this paradigm shift:
  - Similarity-aware query processing in DB
  - Integration of IR and DB operations
  - Uncertain or probabilistic databases
• Motivation
• **Similarity-aware Operators**
  • Implementing Similarity-aware Operators
  • Optimizing Similarity-aware Operators
  • Performance Evaluation
  • Exploiting Similarity-aware Operators
• Future Work
• Similarity Group-by: \((G_1, S_1), ..., (G_n, S_n) \gamma F_1(A_1), ..., F_m(A_m)(R)\)

• SGB Instances:

- **Unsupervised SGB**

- **Sim. Group Around**

- **SGB with Delimiters**

---

**a)** GROUP BY Temperature MAXIMUM_ELEMENT_SEPARATION 2 MAXIMUM_GROUP_DIAMETER 6

**b)** GROUP BY Temperature AROUND \{30,50\} MAXIMUM_ELEMENT_SEPARATION 2 MAXIMUM_GROUP_DIAMETER 20

**c)** GROUP BY Temperature DELIMITED BY (SELECT Value FROM Thresholds)
Similarity-aware operators: SJ

- Similarity Join: \( A \bowtie_{\theta_s} B = \{(a, b) | \theta_s(a, b), a \in A, b \in B\} \)

- SJ Instances:
  
  a) SELECT … FROM A, B WHERE A.a WITHIN Eps OF B.b
  
  b) SELECT … FROM A, B WHERE B.b k NEAREST_NEIGHBOR_OF A.a
Similarity-aware operators: SJ

* SJ Instances (cont.):*

**kDistance-Join**

\[ k = 2 \]

\[
\text{SELECT} \ldots \text{FROM A, B WHERE A.a} \text{ k TOP_CLOSEST_PAIRS B.b}
\]

**Join-Around**

\[
\text{SELECT} \ldots \text{FROM A, B WHERE A.a AROUND B.b} \text{ [MAX_DIAMETER 2r]}
\]
Outline

• Motivation
• Similarity-aware Operators
• **Implementing Similarity-aware Operators**
• Optimizing Similarity-aware Operators
• Performance Evaluation
• Exploiting Similarity-aware Operators
• Future Work
Implementing Similarity-aware Operators

- **The Parser**
  - Extended the grammar rules and parse tree structure

- **The Planer/Optimizer**
  - Each SGB/SJ node processes 1 SGA/SJP and 1 or more GAs/JPs
  - SGB nodes make use of their inner input plan tree

```sql
1. SELECT ... FROM (T) GROUP BY a1 AROUND (T1), a2 AROUND (T2)
2. SELECT ... FROM (T) GROUP BY a1 DELIMITED BY (T1), a2 DELIMITED BY (T2)
```
**The executor**

- Single plane sweep approach used to form the groups
- The tuples to be grouped and the reference points are processed simultaneously
- Data tuples and reference points are sorted before being processed by the aggregation node
- Hash-based approach used to maintain the formed groups

---

![Diagram](image-url)

**Sweeping Plane**

- Data points
- Central points
- Group 1
- Current Group
- MGD

**Similarity Group Around using MAXIMUM_GROUP_DIAMETER (MGD)**
## SJ: Implementation (PostgreSQL)

### The executor

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SMJoin</strong> {</td>
<td><strong>EpsilonJoin</strong> {</td>
<td><strong>JoinAround</strong> {</td>
<td><strong>INITIALIZE</strong></td>
</tr>
<tr>
<td>1 get initial outer tuple</td>
<td>1 get initial outer tuple</td>
<td>1 get initial outer tuple</td>
<td><strong>SKIP_TEST</strong></td>
</tr>
<tr>
<td>2 get initial Inner tuple</td>
<td>2 get initial inner tuple</td>
<td>2 get initial inner and <code>nextInner</code></td>
<td><strong>SKIPOUTER_ADVANCE</strong></td>
</tr>
<tr>
<td>3 do forever {</td>
<td>3 do forever {</td>
<td>3 do forever {</td>
<td><strong>SKIPINNER_ADVANCE</strong></td>
</tr>
<tr>
<td>4 while (outer != inner) {</td>
<td>4 while (outer !~ inner) {</td>
<td>4 while ((inner != nextInner) &amp; &amp; (outer !~ inner)) {</td>
<td><strong>SKIP_TEST</strong></td>
</tr>
<tr>
<td>5 if (outer &lt; inner)</td>
<td>5 if (outer &lt; inner)</td>
<td>5</td>
<td><strong>JOINTUPLES</strong></td>
</tr>
<tr>
<td>6 advance outer</td>
<td>6 advance outer</td>
<td>6</td>
<td><strong>NEXTINNER</strong></td>
</tr>
<tr>
<td>7 else</td>
<td>7 else</td>
<td>7</td>
<td><strong>NEXTOUTER</strong></td>
</tr>
<tr>
<td>8 advance inner</td>
<td>8 advance inner</td>
<td>8</td>
<td><strong>TESTOUTER</strong></td>
</tr>
<tr>
<td>9 }</td>
<td>9 }</td>
<td>9</td>
<td><strong>TESTOUTER</strong></td>
</tr>
<tr>
<td>10 mark inner position</td>
<td>10 mark inner position</td>
<td>10 mark inner position</td>
<td><strong>TESTOUTER</strong></td>
</tr>
<tr>
<td>11 do forever {</td>
<td>11 do forever {</td>
<td>11 do forever {</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>12 do{}</td>
<td>12 do{}</td>
<td>12 do{}</td>
<td><strong>&lt;Testin&gt;</strong></td>
</tr>
<tr>
<td>13 join outer and inner</td>
<td>13 join outer and inner</td>
<td>13 join outer and inner</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>14 advance inner position</td>
<td>14 advance inner position</td>
<td>14 advance inner position</td>
<td><strong>&lt;Testin&gt;</strong></td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
<td><strong>&lt;Testin&gt;</strong></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td><strong>&lt;Testin&gt;</strong></td>
</tr>
<tr>
<td>17 }</td>
<td>17 }</td>
<td>17 }</td>
<td><strong>&lt;Testin&gt;</strong></td>
</tr>
<tr>
<td>18 while (outer == inner)</td>
<td>18 while (outer ~ inner)</td>
<td>18 while (prevInner == inner)</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>19 advance outer position</td>
<td>19 advance outer position</td>
<td>19 advance outer position</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>20 if (outer == mark)</td>
<td>20 if (outer ~ prevInner)</td>
<td>20 if (outer ~ prevInner)</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>21 restore inner to mark</td>
<td>21 restore inner to mark</td>
<td>21 restore inner to mark</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>22 }</td>
<td>22 }</td>
<td>22 }</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>23 else</td>
<td>23 else</td>
<td>23 else</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>24 break</td>
<td>24 break</td>
<td>24 break</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>25 }</td>
<td>25 }</td>
<td>25 }</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>26 }</td>
<td>26 }</td>
<td>26 }</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
<tr>
<td>27 }</td>
<td>27 }</td>
<td>27 }</td>
<td><strong>&lt;Testout&gt;</strong></td>
</tr>
</tbody>
</table>
• Motivation
• Similarity-aware Operators
• Implementing Similarity-aware Operators
• **Optimizing Similarity-aware Operators**
• Performance Evaluation
• Exploiting Similarity-aware Operators
• Future Work
Optimizing similarity-aware operators

- We proposed:
  - Core Equivalence Rules for similarity-aware operators
  - Equivalence Among Similarity Operators
  - Eager and Lazy Aggregation Transformations with SJ and SGB
  - Extended techniques to use Materialized views to answer similarity-aware queries
• Examples of Equivalence Rules:

Q1: SELECT $e_1$, $e_2$ FROM $E_1$, $E_2$
WHERE $e_1$ within 5 of $e_2$ and $20 < e_1 \leq 25$

a) Distribution of selection over SJ

20 < $e_1$ ≤ 25

b) Pushing selection predicate under originally unrelated join operand

15 < $e_2$ ≤ 30
Optimizing similarity-aware operators

• **Examples of Equivalence Rules:**

  1. **Q2:** SELECT $e_1$, $e_2$, $e_2$ FROM $E_1$, $E_2$, $E_3$ WHERE $e_1$ within 5 of $e_2$ and $e_2$ within 0.5 of $e_3$

  2. **Q3:** SELECT $e_1$, $e_2$, $e_4$ FROM $E_1$, $E_2$, $E_4$ WHERE $e_1$ within 5 of $e_2$ and $e_2$ within 5 of $e_4$

  c) **Associativity of SJ operators**

  d) **Associativity rule that enables join on originally unrelated attributes**
### Eager/Lazy Aggr. Transformation with SGB

#### a) Lazy Similarity Aggregation

- **SGB**
  - G1 on Seg1, G2 on Seg2
  - SUM(S1) join SUM(S2)
  - J1=J2

- **Join**
  - T1, T2

- (G1,J1,S1), (G2,J2,S2)

#### b) Eager Similarity Aggregation

- **SGB**
  - G1, G2 on Seg2
  - SUM(S1) as SS1, CNT
  - J1=J2

- **Join**
  - T1, T2

- (G1,J1,S1), (G2,J2,S2)

---

**Business question:** Study the discount level (DL) given by each type of clerk.

**Lazy1**

```
SELECT L.l_discount as DcntLevel, O.o_clerkType, sum(L.l_discount)
FROM L, O
WHERE L.l_orderkey=O.o_orderkey
GROUP BY O.o_clerkType, L.l_discount AROUND <DiscLevel>
```

**Eager1**

```
SELECT R1.l_discount as DcntLevel, O.o_clerkType, sum(R1.CNT)
FROM O,
(SELECT L.l_discount, L.l_orderkey, count(L.l_discount) as CNT
FROM L
GROUP BY L.l_orderkey, L.l_discount AROUND <DiscLevel>) AS R1
WHERE R1.l_orderkey=O.o_orderkey
GROUP BY R1.l_discount, O.o_clerkType
```

---

2/15/2010

*Similarity-aware Query Processing and Optimization - Yasin N. Silva, et al.*
Eager/Lazy Aggr. Transformation with SJ

Q5: SELECT sum(S1), sum(S2) FROM T1, T2 WHERE J1 within 5 of J2 GROUP BY G1, G2

- a) Lazy Aggregation
- b) Eager Aggregation
Optimizing similarity-aware operators

- Pushing Similarity Predicate from Join-Around to GB

SELECT sum(S1), sum(S2) FROM T1, T2 WHERE J1 around J2 MAX_DIAMETER 10 GROUP BY G1, G2

<table>
<thead>
<tr>
<th>T1</th>
<th>G1</th>
<th>J1</th>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>22</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T2</th>
<th>G2</th>
<th>J2</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>40</td>
<td>5</td>
</tr>
</tbody>
</table>

a) Lazy Aggregation

b) Eager Aggregation
Outline

• Motivation
• Similarity-aware Operators
• Implementing Similarity-aware Operators
• Optimizing Similarity-aware Operators
• Performance Evaluation
• Exploiting Similarity-aware Operators
• Future Work
Performance Evaluation (TPC-H)

- Performance of SGB while increasing dataset size
Performance Evaluation (TPC-H)

- Comparison of SGB and SJ with queries that obtain the same answer using regular operators.

![Graph showing performance comparison between SGB and SJ](image)
Effectiveness of Associativity transformation for SJ
Performance Evaluation (TPC-H)

- Effectiveness of pushing selection under SJ

![Diagram showing execution time for different operations]

- Execution Time (s)
  - PushSel_LHS
  - PushSel_RHS1
  - PushSel_RHS2

Effectiveness of pushing selection under SJ
SGB: Performance Evaluation (TPC-H)

- Effectiveness of Lazy and Eager aggregation transformations with SJ
Outline

- Motivation
- Similarity-aware Operators
- Implementing Similarity-aware Operators
- Optimizing Similarity-aware Operators
- Performance Evaluation
- Exploiting Similarity-aware Operators
- Future Work
Studying groups of large-volume customers with similar buying power

Studying profit of a line of parts around marketing campaigns

Studying groups of orders around revenue levels of interest

SELECT TotalBuy as TotalBuyLevelRef, min(TotalBuy), max(TotalBuy), count(TotalBuy), avg(TotalBuy)
FROM (SELECT c_name, c_custkey, sum(l_extendedprice) as TotalBuy
FROM C, O, L
WHERE c_custkey = o_custkey and o_orderkey = l_orderkey and
o_orderkey IN (SELECT l_orderkey FROM L
GROUP BY l_orderkey
HAVING sum(l_quantity) > A)
GROUP BY c_name, c_custkey)
GROUP BY TotalBuy

SELECT revenue as RevLevel, count(revenue), min(revenue), max(revenue), avg(revenue)
FROM (SELECT l_orderkey, sum(l_extendedprice * (1-l_discount)) as revenue
FROM C, O, L
WHERE c_mktsegment = A and c_custkey = o_custkey and
l_orderkey = o_orderkey and o_orderdate < date B and
l_shipdate > date C
GROUP BY l_orderkey) as R1
GROUP BY revenue

SELECT nation, o_orderdate as MktCmpRefDate, sum(amount) as sum_profit
FROM (SELECT n_name as nation, o_orderdate, l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity * (1+C) as amount
FROM P, S, L, PS, O, N
WHERE s_suppkey = l_suppkey and ps_suppkey = l_suppkey and
ps_partkey = l_partkey and p_partkey = l_partkey and
o_orderkey = l_orderkey and s_nationkey = n_nationkey
and p_name like '%A%') as profit
GROUP BY nation, o_orderdate
GROUP BY nation
ORDER BY nation

Note: l_discount is replaced by D in case D is not 'Actual Discounts'

Note: l_discount should be replaced by E if E is not 'Actual Discounts'
• Motivation
• Similarity-aware Operators
• Implementing Similarity-aware Operators
• Optimizing Similarity-aware Operators
• Performance Evaluation
• Exploiting Similarity-aware Operators
• Future Work
Future Work

Similarity-aware Query Processing and Optimization

• Publications

• Future Work
  – Similarity-aware database for sensor networks
  – Massively parallel data stream management with similarity embedded
  – Other similarity-aware database operators
  – Similarity-aware data warehousing operators: Similarity CUBE and ROLLUP
  – Benchmark for Similarity-based Query Processing
Future Work

Web-scale Data Management Systems – Cloud Computing (and large data indexing)

• Publications

• Future Work
  – Create and extend optimization techniques for web-scale data management systems
    • Study how optimization techniques, proposed in the context of conventional database systems, can be applied and extended to these massively parallel systems
    • Study query transformation techniques, considering the demanding partition requirements common to this scenario
    • Study pre-aggregation techniques
    • Extension of the core concepts behind the use of materialized views to improve query execution time
  – Propose and study indexing structures for web-scale data management systems
    • Design and implement indexing structures that efficiently support map-reduce-based operations
  – Propose and implement similarity-aware operations for fast analysis of data in the cloud
Future Work

Privacy Assurance

• Publications

• Future Work
  – Design and implement key components of a privacy-preserving database:
    • Limited collection (mechanisms that ensure that the collected personal information is minimized to accomplish the specified purposes)
    • Compliance (a component that allows the data owner to verify the compliance of the data management system with the privacy policy)
    • Limited use (mechanisms that ensure the database will only run queries consistent with the purpose for which the information was collected)
  – Study, design, and implement privacy preserving features for scientific databases
Future Work

Scientific Data management

• Publications
  – M. Eltabakh, W. G. Aref, A. Elmagarmid, M. Ouzzani, Y. N. Silva. Supporting Annotations on Relations. The 12th International Conference on Extending Database Technology (EDBT), Russia, 2009 (full paper, acceptance rate 32.5%).

• Future Work
  – Study the role of similarity-aware operations in scientific databases
    • Example: to find naturally formed clusters and cluster representatives in microdata records
  – Study the properties of similarity-aware operators when specialized similarity functions for scientific data are used
  – Propose a family of similarity-aware database operators for fast scientific data analysis
## Comparison of implementation approaches

<table>
<thead>
<tr>
<th>Similarity Operator Implementation Approach</th>
<th>Integrated in DB Engine</th>
<th>Using Basic SQL Operators</th>
<th>Outside of DB</th>
<th>As Stored Procedures (SP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Operator Instances</td>
<td>All</td>
<td>Certain types may be unfeasible or require very complex queries</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Implementation complexity</td>
<td>Can reuse and extend DB operators and structures</td>
<td>Queries use a complex mix of joins and aggregations</td>
<td>Requires the support of specialized structures, mechanisms to deal with very large data sets, etc.</td>
<td>Requires the support of specialized structures, spilling mechanisms, etc.</td>
</tr>
<tr>
<td>Composable with other DB operators (pipelining)</td>
<td>Yes</td>
<td>Yes (resulting queries can be highly complex)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Take advantage of DB cost based optimization</td>
<td>Yes (e.g., queries can be pre-aggregated, use MVs, be translated, etc.)</td>
<td>No directly</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
SGB: Performance Evaluation (TPC-H)

- Effectiveness of Lazy and Eager aggregation transformations with SJ

- Performance of complex TPC aggregation queries