B-HUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data

Paul G. Brown & Peter J. Haas
IBM Almaden Research Center
San Jose, CA
A Motivating Example

- Shipment data:

<table>
<thead>
<tr>
<th>orderID</th>
<th>shipDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A5</td>
<td>2001-01-03</td>
</tr>
<tr>
<td>3C2</td>
<td>2001-04-15</td>
</tr>
<tr>
<td>3B8</td>
<td>2002-11-25</td>
</tr>
<tr>
<td>2E1</td>
<td>2002-10-31</td>
</tr>
<tr>
<td>3D6</td>
<td>2002-07-25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>orderID</th>
<th>deliveryDate</th>
<th>deliveryTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A5</td>
<td>2001-01-06</td>
<td>09:50</td>
</tr>
<tr>
<td>3C2</td>
<td>2001-04-27</td>
<td>13:00</td>
</tr>
<tr>
<td>3B8</td>
<td>2002-12-19</td>
<td>11:20</td>
</tr>
<tr>
<td>2E1</td>
<td>2001-12-02</td>
<td>16:10</td>
</tr>
<tr>
<td>3D6</td>
<td>2002-07-29</td>
<td>08:50</td>
</tr>
</tbody>
</table>

orders | deliveries
Example: Fuzzy Constraints

SELECT DAYS(deliveryDate) – DAYS(shipDate) FROM orders, deliveries WHERE orders.orderID = deliveries.orderID

(deliveryDate BETWEEN shipDate + 2 AND shipDate + 5) (25%)
OR (deliveryDate BETWEEN shipDate + 12 AND shipDate + 19) (50%)
OR (deliveryDate BETWEEN shipDate + 31 AND shipDate + 35) (25%)
Exploiting the Constraints

SELECT COUNT(*) FROM orders, deliveries
WHERE shipDate = '2003-07-02'
AND deliveryTime > '17:00'
AND orders.orderID = deliveries.orderID

Indexes:
orders.ordersID,
deliveries.orderID
deliveries.deliveryDate
(NOT orders.shipDate)

Derived predicate:
(2003-07-04 [ deliveryDate [ 2003-07-07]
OR (2003-07-14 [ deliveryDate [ 2003-07-21]
OR (2003-08-02 [ deliveryDate [ 2003-08-06]

A plan

A better plan

IScan: Deliveries.orderID
Pred: deliveryTime

Scan: orders
Pred: shipDate

IScan: orders.ordersID
Pred: shipDate

IScan: deliveries.deliveryDate
Pred: *
Pred: deliveryTime
Example 2: Partitioned Data

<table>
<thead>
<tr>
<th>orderID</th>
<th>shipDate</th>
<th>deliveryDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A5</td>
<td>2003-01-03</td>
<td>2003-01-06</td>
</tr>
<tr>
<td>7D3</td>
<td>2003-01-17</td>
<td>2003-01-20</td>
</tr>
<tr>
<td>3B8</td>
<td>2003-06-19</td>
<td>2003-07-02</td>
</tr>
<tr>
<td>2E1</td>
<td>2003-06-16</td>
<td>2003-07-03</td>
</tr>
<tr>
<td>3D6</td>
<td>2003-08-25</td>
<td>2003-08-29</td>
</tr>
<tr>
<td>4D2</td>
<td>2003-09-12</td>
<td>2003-09-22</td>
</tr>
</tbody>
</table>

```
SELECT COUNT(*)
FROM orders
WHERE shipDate = '2003-07-01'
```

Derived predicate:

(2003-07-03 [ deliveryDate [ 2003-07-10])
OR (2003-07-13 [ deliveryDate [ 2003-07-24])
OR (2003-08-01 [ deliveryDate [ 2003-08-05])

Fragment elimination!

Horizontally range-partitioned
B-HUNT Overview

- Automatic discovery of fuzzy algebraic constraints

Why useful?
- Query optimization (new plans, costing)
- Advice on data partitioning, view/index creation
- Constraints interesting in themselves

Hidden constraints abound in real world
- Unknown to application developer and DBA
- Enforced by application but unknown to DBA
- Known to DBA but not enforced due to high cost
- Constraint is fuzzy, so not a standard DB “rule” per se
Fuzzy Algebraic Constraints

- **Algebraic relationships:** $a_1 \oplus a_2 \in I$
  - $\oplus$ is +, -, x, ÷, etc.
  - $a_1, a_2$ are attributes
  - $I$ is subset of real numbers

- **Pairing rule $P$**
  - Determines which $a_1$ value goes with which $a_2$ value
  - Trivial pairing rule $\emptyset_R$ for table $R$:
    - $a_1$ value paired with $a_2$ value in same row of $R$
  - If attributes in different tables: $P$ = join predicate
    - Self-joins OK also

- **Algebraic constraint:** $AC = (a_1, a_2, P, \oplus, I)$
Algebraic Constraints, Continued

- Previous Example 1:
  - $a_1 = \text{deliveries.deliveryDate}$, $a_2 = \text{orders.shipDate}$
  - $\oplus$ is subtraction operator
  - $P : \text{\textquote{orders.orderID = deliveries.orderID}}$
  - $I = \{2,3,4,5\} \cup \{12,13,19\} \cup \{31,32,33,34,35\}$

- Previous Example 2: same as Example 1 except
  - $a_1 = \text{orders.deliveryDate}$,
  - $P = \emptyset_{\text{orders}}$

- Focus on case where $I = I_1 \cup I_2 \cup \ldots \cup I_k$
  - The $I_n$'s are disjoint “bump intervals” (of real line or integers)
Outline of B-HUNT Algorithm

- Find *candidates* of form: $C = (a_1, a_2, P, \oplus)$
  - Find useful pairing rules
  - For each rule $P$ find useful triples $(a_1, a_2, \oplus)$
- For each candidate, construct bump intervals
  - Based on sampled rows of (key) table
  - Use histogramming, segmentation, or clustering
  - Choose sample size to control # of *exceptions*

For query optimization:
- At load time: partition data into compliant + exceptions
- During query processing: combine results of
  - Running modified query that incorporates constraints
  - Running original query over (small) exception data
Candidate Generation: Pairing Rules

1. Generate trivial pairing rules: $\emptyset_{T_1}, \emptyset_{T_2}, \emptyset_{T_3}, \emptyset_{T_4}$
2. Generate set $K$ of “key-like” attributes:
   - declared primary and unique keys (and declared compound keys)
   - attributes $a$ such that $\text{#rows}(a) \div \#\text{distinctValues}(a) \approx 1$
3. For each $a \in K$, add ‘$R.a = S.b$’ to set of pairing rules iff
   - (i) $a$ and $b$ are of same datatype and either
   - (ii) $(a,b)$ is declared (primary key,foreign key) pair; or
   - (iii) Every value in a sample from $b$ has a match in $a$
Pruning the Pairing Rules

- Adjustable heuristic pruning criteria:
  - Trade off thoroughness and efficiency
  - For optimization: want pairing rules that
    - Lead to constraints with impact
    - Are easy to exploit at run time
    - Occur frequently in workload

- Examples: prune a pairing rule “$R.a = S.b$” if
  - $R$ and $S$ are “small” (no impact)
  - $R$ or $S$ has no index (hard to exploit)
  - $a \in K$ and $|S.b|/|R.a|$ is “small” (spurious relationship)
  - $S.b$ is a system-generated key (spurious relationship)
For each pairing rule, consider all attribute pairs \((a_1, a_2)\) such that

- \(a_1\) and \(a_2\) can be operated on by \(\oplus\)
- \((a_1, a_2)\) not equal to attributes in pairing-rule join predicate

Prune candidate \((a_1, a_2, P, \oplus)\) if, e.g.,
- attributes have different data types
- too many NULL values
- either attribute lacks an index

\(P_2 = \emptyset_{T3}\)
Phrenology: Hunting the Bumps

- Each candidate $C = (a_1, a_2, P, \oplus)$ defines set of points $\Omega_C$
- Bump hunt on sample of points from $\Omega_C$
  - Because bump hunting must be scalable
- No exceptions in sample
  - I.e., segment the sample points

\[
\begin{align*}
I_1 & = x_4 - x_1 \\
I_2 & = x_7 - x_5 \\
I_3 & = x_9 - x_8
\end{align*}
\]

- Choose sample size to control # of exceptions in full DB

VLDB 2003
Direct “Optimal” Segmentation

- Trade off filtering power and complexity vs complexity
- Rough cost function (k = # intervals):

\[ c(S) = wk + (1-w) \left( \frac{1}{\Delta} \sum_{j=1}^{k} L_j \right) \]

- w is a weight between 0 and 1
- \( \Delta \) is estimated range of data values
- To minimize \( c(S) \):
  - adjacent points in same segment iff \( x_{i+1} - x_i < d^* \), where
    \[ d^* = \Delta \left( \frac{w}{(1-w)} \right) \]
  - For discrete data types use \( \max(d^*,1+\varepsilon) \)
Histogram-Based Segmentation

- Use $2h(n)$ buckets:
  - $h(n) = (2n)^{1/3}$ is “oversmoothing” lower bound
  - Minimizes asymptotic mean integrated squared error
  - Center an interval of length $2h(n)/\Delta$ around each isolated point
Choosing the Sample Size

- Uses approximate (conservative) estimate $n^*(k)$ of required sample size for a $k$-segmentation

$$n^*(k) = \frac{\chi^2_{1-p,2(k+1)}}{4f} + \frac{k}{2}$$

- With probability $p$, fraction of exceptions is at most $f$
- Uses theory of tolerance intervals (Tukey and Sheffé)

Iterative procedure:
1. (Initialization) Set $k = 1$
2. Take sample of size $n \geq n^*(k)$
3. Compute constraint and observe number $k'$ of bump intervals
4. If $n \geq n^*(k')$ then go to step 5, else set $k = k'$ and go to step 2
5. (Cleanup) Adjust for NULLs, Bernoulli fluctuations
Using the Constraints for Optimization

- Choose most important constraints (e.g. by filtering power)
- Partition data into “compliant” and “exception”
  - Physical partitioning or partial indexes
  - Table creation, e.g.:

```sql
CREATE TABLE exceptions(...);
INSERT INTO exceptions AS
(SELECT orders.orderID, deliveries.orderID,
orders.shipDate, deliveries.deliveryDate,
deliveries.deliveryTime
FROM orders, deliveries
WHERE orders.orderID = deliveries.orderID
AND NOT (deliveryDate BETWEEN shipDate + 2 DAYS
AND shipDate + 5 DAYS)
OR (deliveryDate BETWEEN shipDate + 12 DAYS
AND shipDate + 19 DAYS)
OR (deliveryDate BETWEEN shipDate + 31 DAYS
AND shipDate + 35 DAYS));
```

- Subsequent optimization builds on standard query processing technology
An Empirical Study

- **The Database**
  - 7 years of synthetic retail data
  - Similar to TPC-D schema
  - > 2.3 terabytes
  - Two largest tables exceed 13.8 billion and 3.45 billion rows

- **Discovered constraints include:**
  - \( t_1.orderDate \leq t_2.shipDate \leq t_1.orderDate + 4 \text{ MONTHS} \)
  - \( t_2.shipDate \leq t_2.receiveDate \leq t_2.shipDate + 1 \text{ MONTH} \)

- **Time to discover constraints:**
  - 4 minutes (in addition to ordinary statistics collection)
  - Versus hours or days for fancy mining methods
Empirical Study, Continued

- Improvement for 50% of the queries
- Significant improvement for 25%
- Best speedup: 6.8x (accesses to largest table reduced 100x)
- No significant performance decreases
Related Work

- Large literature on DB learning & relationship discovery
  - Query-driven methods (LEO, SITS, semantic constraints, etc.)
  - Data-driven methods (synopses, assoc. rules, reverse engrrg.)

- Novel aspects of our work:
  - Fuzziness + algebraic rules + sampling + data-driven
Conclusions

- Fuzzy algebraic constraints are useful and interesting
- B-HUNT algorithm(s) for discovering such constraints
  - Highly automated
  - Fast (sampling based)
  - Robust to noisy data
  - Can lead to significant speedups in query processing
- A step towards smarter DBMS
- A useful framework for learning about data
Future Work

- **Improvements**
  - More extensive experimentation
  - More efficient techniques to enumerate pairing rules
    - Bell and Brockhausen
  - Exploitation of unique indexes, UNIQUE clauses in DDL, etc.

- **Extensions**
  - Apply to fuzzy functional dependencies
    - Bump at #(Honda) - #(Accord), not at #(Honda) - #(Camry)
  - Extend to XML repositories
  - Combine with query-driven technologies
    - Better pruning in B-HUNT
    - Avoid bad-warm-up and knowledge-phobic behavior of q-d
Thanks to...

- Qi Cheng
- Shu Lin
- Wen-Bin Ma
- Hamid Pirahesh
- Haider Rizvi
- Richard Sidle
- Ashutosh Singh
- Jason Sun
- Calisto Zuzarte
Extra Slides
Correction for Real-Valued Data

- Expand endpoints by a few %
  - Merge overlapping intervals
  - Idea: deal with logarithmic rate of progress
  - Example: rightmost edge of rightmost bump interval

![Diagram showing correction of real-valued data with marked true maximum data value.](image)
Computing the Estimate

- Look at straw man algorithm to get $n^*(k)$
  - Chooses a random $k$-segmentation of data points
  - Yields a conservative sample size

- Computing the required sample size
  - Related to quality control problems for manufacturing
  - Theory of tolerance intervals
Upper Bound, Continued

- **Theorem:** \( \text{Prob}\{F > x\} \leq \text{Beta}(1-x; n-k, k+1) \)
  - \( F \) is fraction of points in \( \Omega_c \) that lie outside of \( k \)-segmentation
  - Beta is cumulative beta distribution function
    \[
    \text{Beta}(t; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha) \Gamma(\beta)} \int_0^t u^{\alpha-1}(1-u)^{\beta-1} du
    \]
  - Proof uses several results of Tukey and Sheffé from 1940’s
- **To get sample size, solve:** \( \text{Beta}(1-f; n-k, k+1) = 1-p \)
  - With probability at least \( p \), exception fraction is at most \( f \)
  - Use Tukey and Sheffé approximation of Beta inverse:
    \[
    n^*(k) \approx \frac{\chi^2_{1-p}}{4f} + \frac{k}{2}
    \]
  - \( \chi^2 \) is \( 100\alpha \)\% percentage point of \( \chi^2 \) distribution with \( 2(k+1) \) degrees of freedom
Related Work

- Large literature on DB learning & relationship discovery
  - Query-driven methods
    - LEO learning optimizer [Stillger et al.]
    - SITS [Bruno and Chaudhuri]
    - Discovering semantic integrity constraints [Siegel, Yu & Sun]
  - Data-driven methods
    - Computation of synopses of multidimensional distributions
      - Histograms, wavelets, samples, Bayesian networks, etc.
    - Association rules, etc.
    - Mining functional and multi-valued dependencies
      - Reverse engineering (usually based on schema info)
      - Approximate functional dependencies [Huhtala et al.]

- Novel aspects of our work:
  - Fuzziness + algebraic rules + sampling + data-driven