An Algorithmist’s Take on Relational Algorithms
(Intermediate Relational Algorithm Design)
Kirk Pruhs

Acknowledgements to:
• Academic Network: Sungjin Im, Ben Moseley, Alireza Samadian, Yuyan Wang
• RAI folks: Mahmoud Abo-Khamis, Ryan Curtin, Hung Ngo
Recall: Relational Algorithms

- **Relational algorithms**: Algorithms that are
  - efficient (say polynomial time), and
  - accept the input is in relational form

- Relational algorithms necessarily can not afford to join the tables
Analogous Situation

- Goal: \text{StringSearch(Compressed\ String)}

- Standard Approach: \text{StandardAlgorithm(Uncompress(Compressed\ String))}

- [SMTSA, CPM2000] \text{NewSearchAlgorithm(Compressed\ String)}
Intended Take Away Points

• Barrier to entry into relational algorithms is relatively low

• Potentially interesting open algorithmic problems

• But problems have to be mined (not picked)
Problem Mining Is Important In Restricted Computation Model Research

- **Kindred Restricted Computational Model**
  - **Streaming**
    - Algorithm only is allowed a small number (most commonly 1) linear passes over the data
  - **Massive Parallel Computing (MPC)**
    - think MapReduce
    - Distributed model in which no computer has enough memory to store all of the input

- My experience is that the key to doing research in these areas is finding/mining problems where positive results are achievable
  - Not problem solving!
Necessary Background Before Getting Started Designing Relational Algorithms

• Graphic and geometric views of a join

• Sum-Product query

• Variable elimination algorithm

• How to use Sum-Product queries to develop algorithms
### Graphic View of Join

**Table A**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table B**

<table>
<thead>
<tr>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

**Join Design Matrix**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
Intuitive Geometric View of a Join

• $\prod_{xy} (A(x, y) \bowtie B(y,z)) \approx A(x, y)$
• $\prod_{yz} (A(x, y) \bowtie B(y,z)) \approx B(y,z)$

• Join is intuitively the maximal inverse of projection
Necessary Background Before Getting Started Designing Relational Algorithms

• Graphic and geometric views of a join

• Sum-Product query

• Variable elimination algorithm

• How to use Sum-Product queries to develop algorithms
Design of Relational Algorithms

Key:
- Problems
- Algorithmic technique

- k-NN
- Linear Regression
- Greedy Decision Tree
- k-means
- SVM gradient
- SumProd Aggregation
  - with an Additive Inequality
- Stability Analysis
- Dynamic Programming Semi-rings
- SumProd Aggregation
Sum-Product Query

• Mahmoud’s view:
  – $\oplus_{x_1, \ldots, x_k} \otimes_{x_S} f_S(x_S)$
  – Where each $S$ is conceptually a table with variables $x_S$

• My take abstracts out the tables
  – $\oplus_r \otimes_c f_c(M_{rc})$
  – Where $r$ is a generic row, and $c$ is a generic column, in the joined table $M$
  – Where $\oplus$ and $\otimes$ form a commutative semiring
Candidate Problems to Develop Relational Algorithms For

• Any geometric problem where the input is a collection of points in some higher dimensional space
  – Example: How many points are in the input?
  – Example: Which point is furthest from the origin?
  – Example: k-means
Geometric Problem: How Many Points are in the Input

- Standard input representation: Trivial

- Input is in relational format:
  - NP-hard to decide if the input is nonempty
  - \#P-complete
  - But this is not important

- Sum-Product Query
  - ⨁ is addition
  - ⊗ is multiplication
  - \( f_c(x) = 1 \)
  - So \( \oplus_r \otimes_c f_c(M_{rc}) = \sum_r \prod_c 1 \)
  - Note \( r \) is a point and \( c \) is a coordinate/dimension
  - \( (1*1*1)+(1*1*1)+(1*1*1)+(1*1*1)+(1*1*1)+(1*1*1)+(1*1*1)+(1*1*1) = 8 \)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Geometric Problem: Distance of Furthest Point From the Origin

• Sum-Product Query
  – $\oplus$ is max
  – $\otimes$ is addition
  – $f_c(x) = x^2$
  – So $\oplus_r \otimes_c f_c(M_{rc}) = \max_r \sum_c M_{rc}^2$

• $(3^2+1^2+7^2)$ max $(3^2+1^2+8^2)$
  max ...
Necessary Background Before Getting Started Designing Relational Algorithms

• Graphic and geometric views of a join

• Sum-Product query

• Variable elimination algorithm

• How to use Sum-Product queries to develop algorithms
Illustrative Example Problem: k-means Clustering
k-means Problem

- Input: points $x_1, \ldots, x_m$ in Euclidean space and integer $k$
k-means Problem

• Input: points $x_1, ..., x_m$ in Euclidean space and integer $k$

• Feasible solution: $k$ centers/points $S_1, ..., S_k$

• Objective: Minimize aggregate 2-norm squared distances to nearest center
  - $\text{Min} \ \sum_{i \in [m]} \ \min_{j \in [k]} \langle x_i - S_h \rangle$
  - Where $\langle x_i - S_h \rangle$ is 2-norm squared
Strategic Plan Once You’ve Picked a Problem

A. Design a relational implementation of a/the standard non-relational algorithm

B. Design a relational algorithm that doesn’t exactly implement the standard algorithm, but that has the same theoretical guarantees as the standard algorithm

C. Design a relational algorithm that has some reasonable theoretical guarantee
Standard k-means++ Algorithm [AV2007]

• K-means++ Algorithm: Pick a point as the next center with probability proportional to its distance to its nearest previous center

• Plan A succeeds for k-means++: There is a relational implementation
Standard Adaptive k-means Algorithm [ADK2009]

• **Plan A fails**: A relational implementation of the adaptive k-means algorithm would imply P=NP
  – NP-hardness is a reasonably effective tool for proving the likely nonexistence of relational algorithms

• **Plan B succeeds**: We can modify the adaptive k-means algorithm so that
  – It can be implemented relationally
  – It still has the same theoretical guarantee of bounded relative error
Algorithmic Design Strategies

A. Express the problem using a Sum-Product query
   – Implementation of 1-means++

B. Design algorithm from scratch
   1. First try cross-product join
   2. Then try path join
   3. Then try express computation as sum-product query
      – Implementation of 2-means++
      – Implementation of 3-means++
      – Approximately counting points in a hypersphere
        • subroutine to our relational modification of the adaptive k-means algorithm

C. Build algorithm from components one knows how to compute using Sum-product queries
   – Adaptive k-means algorithm
Relational Implementation of 1-means++ Algorithm

• 1-means++ Algorithm
  – Pick center $S_1$ uniformly at random from $x_1$, ..., $x_n$

• Uniform generation can be reduced to counting
  – Standard variable elimination algorithm keeps track of Sum-Product ala shortest path algorithms

• Implementation of counting as a SumProd query $\sum_r \prod_c 1$
Computing Aggregate Number of Points \((\sum_r \prod_{c=1}^5 1)\) on a Path Join

<table>
<thead>
<tr>
<th>Input Table T1</th>
<th>Input Table T2</th>
<th>Input Table T3</th>
<th>Input Table T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F2</td>
<td>F2</td>
<td>F3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Each source to sink path can be viewed as a point in 5 dimensional space

Computed by variable elimination algorithm
Algorithmic Design Strategies

A. Express the problem using a Sum-Product query
   – Implementation of 1-means++

B. Design algorithm from scratch
   1. First try cross-product join
   2. Then try path join
   3. Then try express computation as sum-product query
      – Implementation of 2-means++
      – Implementation of 3-means++
      – Approximately counting points in a hypersphere
         • subroutine to our relational modification of the adaptive k-means algorithm

C. Build algorithm from components one knows how to compute using Sum-product queries
   – Adaptive k-means algorithm
Relational Implementation of 2-means++ Algorithm

• 2-means++ Algorithm
  – Pick center $S_1$ uniformly at random from $x_1, \ldots, x_n$
  – Pick $x_i$ as center $S_2$ with probability proportional to $\langle x_i - S_1 \rangle$, the 2-norm squared distance to $S_2$

• Implementation of second step
  – Again reduce sampling to summing
  – Need aggregate 2-norm squared
Start with a Path Join

Each source to sink path can be viewed as a point in 5 dimensional space

Computed by variable elimination algorithm
Compute aggregate 2-norm squared on a path join

- Algorithm: Process edges left to right
  - $\text{Sum}^2(z) = \text{Sum}^2(y) + z^2 \times \text{numpaths}(y)$
  - $\text{Numpaths}(z) = \text{numpaths}(z) + \text{numpaths}(y)$

- Take away: You need to remember aggregate square sum and number of paths
Compute aggregate 2-norm squared on a general join

• Base elements of semi-ring pairs \((n, s)\) of numbers
  – \(n\) is a row count
  – \(s\) is a sum of squares
• Need to design \(⊕\) and \(⊗\) such that variable elimination yields

\[
(n(a), s(a)) ⊕ \[n(a), s(a)] ⊗ (1, y^2) = (n(z) + n(a), s(z) + s(a) + n(a)y^2)
\]

Intuition: Think shortest paths

\[sp(z) = \min(sp(z), sp(a) + y^2)\]
Compute aggregate 2-norm squared on a general join

- Dynamic Programming Semiring
  - \((a, b) \oplus (c, d) = (a + c, b + d)\)
  - \((a, b) \otimes (c, d) = (ac, ad + bc)\)
  - Multiplicative identity \((1, 0)\)
  - Additive identity \((0, 0)\)

\[
(n(a), s(a)) \oplus [(n(a), s(a)) \otimes (1, y^2)] = (n(z) + n(a), s(z) + s(a) + n(a)y^2)
\]
Algorithmically Interesting Insight

• Known: Dynamic Programs can be used to compute sum-product queries
  – For example, standard shortest path algorithms such as Dijkstra and Bellman-Ford extend to computing sum-product query over a commutative semiring

• New to me: Many standard dynamic programs can be expressed as sum-product queries where the elements of the ground set in the semiring are the rows in the dynamic programming table
Algorithmic Design Strategies

A. Express the problem using a Sum-Product query
   – Implementation of 1-means++

B. Design algorithm from scratch
   1. First try cross-product join
   2. Then try path join
   3. Then try express computation as sum-product query
      – Implementation of 2-means++
      – Implementation of 3-means++
      – Approximately counting points in a hypersphere
         • subroutine to our relational modification of the adaptive k-means algorithm

C. Build algorithm from components one knows how to compute using
   Sum-product queries
   – Adaptive k-means algorithm
3-means++ Algorithm

• Pick center $S_1$ uniformly at random from $x_1, \ldots, x_n$
• Pick $x_i$ as center $S_2$ with probability proportional to $\langle x_i - S_1 \rangle$
• Pick $x_i$ as center $S_3$ with probability proportional to $\min(\langle x_i - S_1 \rangle, \langle x_i - S_2 \rangle)$, the 2-norm squared distance to previous center
Picking $S_3$

Pick each point with probability proportional to distance to $S_2$

Pick each point with probability proportional to distance to $S_1$
Theorem: It's NP-hard to compute aggregate distance of points to the dividing line even if tables are simple.

Therefore we can't reduce random selection to summing.
Digression: Rejection Sampling

• Given a uniform sample over the red square:
  – Generate a uniform sample over the blue circle
  – Estimate area of the blue circle
More Rejection Sampling

• Assumptions:
  – Want to sample an element \( r \) with probability proportional to \( h(r) \)
    • Easy to compute \( h(r) \)
    • Hard to compute \( H = \Sigma_r h(r) \)
  – Surrogate distribution \( e \)
    • Easy to compute \( e(r) \)
    • \( h(r) < e(r) \)
    • Easy to compute \( E = \Sigma_r e(r) \)

• Rejection sampling
  – Pick \( r \) with probability \( \frac{e(r)}{E} \)
  – Accept \( r \) with probability \( \frac{h(r)}{e(r)} \) else resample

• Theorem: \( r \) is sampled with probability proportional to \( h(r) \) in expected time \( \frac{E}{H} \)
Defining Easy Distribution $e$

- $e(x_i) = \begin{cases} 
\text{Distance from } x_i \text{ to } S_2 \text{ if } x_i \text{ in box } B_2 \\
\text{Distance from } x_i \text{ to } S_1 \text{ otherwise}
\end{cases}$
Computing $E = \sum_i e(x_i)$ Using Sum-Product Query

- $f_c(x) =$
  - $(x-S_2(c))^2$ if $\text{LB}_2(c) < x < \text{UB}_2(c)$
  - $(x-S_1(c))^2$ if otherwise

- $E = \sum_r \prod_c f_c(M_{rc})$
Picking $S_3$

Pick each point with probability proportional to distance to $S_2$

Pick each point with probability proportional to distance to $S_1$

Conclusion: Using rejection sampling one can sample from this hard distribution using $d$ samples in expectation from the easy distribution.
Algorithmic Design Strategies

A. Express the problem using a Sum-Product query
   – Implementation of 1-means++

B. Design algorithm from scratch
   1. First try cross-product join
   2. Then try path join
   3. Then try express computation as sum-product query
      – Implementation of 2-means++
      – Implementation of 3-means++
      – Approximately counting points in a hypersphere
        • subroutine to our relational modification of the adaptive k-means algorithm

C. Build algorithm from components one knows how to compute using Sum-product queries
   – Adaptive k-means algorithm
Sum-Product Query with Additive Inequality

• **Definition**
  – Compute $\bigoplus_r \bigotimes_c f_c(M_{rc})$
  – For those $r$ where $\sum_c g_c(M_{rc}) \leq R$

• **Fact:** Can be approximated within a $(1+\varepsilon)$ factor by a sum product query that implements a dynamic program
  – Assuming operations are approximation preserving (so not subtraction)

• **Special Case:** Count the points in hypersphere centered at origin
  • $\sum_r \prod_c 1$
  • For those $r$ where $\sum_c M_{rc}^2 \leq R$
Intended Take Away Points

• Barrier to entry into relational algorithms is relatively low

• Potentially interesting open algorithmic problems

• But problems have to be mined (not picked)
Discussion Problems

- **Onboarding Warmup Problem:** Find a relational implementation of the ID3 decision tree construction algorithm that is as efficient at possible.

- **Open Problem:** Identify geometric problems that would are potentially interesting to develop relational algorithms for.
Core of ID3 Algorithm

- Table T entropy
  - $H(T) = q \log \frac{1}{q} + (1-q) \log \frac{1}{1-q}$
  - $q =$ probability label is 0

- This example: $H(T) = (2/6)(\log \frac{6}{3}) + (4/6)(\log \frac{6}{4})$

<table>
<thead>
<tr>
<th>Table T</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Label</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Core of ID3 Algorithm

• Find comparison C of the form:
  – attribute $\leq$ value
  – that gives maximum information
    • Equivalent to minimizing the resulting conditional entropy $H(T \mid C)$
    • $H(T \mid C) = \text{Prob}(C=0) \cdot H(T \mid C=0) + \text{Prob}(C=1) \cdot H(T \mid C=1)$
Core of ID3 Algorithm

• Consider C is \( U \leq 4 \)

• \( H(T \mid C) = (2/3) H(T \mid U \leq 4) + (1/3) H(T \mid U > 4) = \\
\quad - (2/3) (1/4 \lg 4 + 3/4 \lg 4/3) + (1/3) (1/2 \lg 2 + ½ \lg 2) \)

<table>
<thead>
<tr>
<th>U</th>
<th>V</th>
<th>W</th>
<th>X</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
Core of ID3 Algorithm

• Find comparison C of the form:
  – attribute ≤ value
  – that gives maximum information
    • Equivalent to minimizing the resulting conditional entropy $H(T \mid C)$
    • $H(T \mid C) = \text{Prob}(C=0) \cdot H(T \mid C=0) + \text{Prob}(C=1) \cdot H(T \mid C=1)$

• Onboarding warmup problem: Find a relational algorithm to find this comparison C that is as efficient as possible
Workshop Outing

• San Francisco Giants baseball game Wednesday evening
• It is not important that you understand/like baseball
• Contact me if you are interested in joining
Thank you for listening

PSYCHIATRIC HELP 7¢

THE DOCTOR IS IN

© PNTS

SCHULZ
Dynamic Programming

- \( D[r] = \) number of points at distance \( r \)
- Need to design \( \oplus \) and \( \otimes \) such that variable elimination yields

\[
D_z^{[r]} = D_z \oplus (D_a \otimes D_y) = D_z^r + D_a^{[r - y^2]}
\]
Counting Points in Hypersphere Centered at the Origin

- Dynamic Programming Semiring
  - $D_a \oplus D_b = \text{coordinatewise addition}$
  - $D_a \otimes D_b[r] = \sum_e D_a[e] * D_b[r-e]$
  - Multiplicative identity is 1 point at distance 0
  - Additive identity is zero vector

$r^{\text{th}}$ entry of $D_z = D_z \oplus (D_a \otimes D_y)$

$= D_z[r] + D_a[r - y^2]$