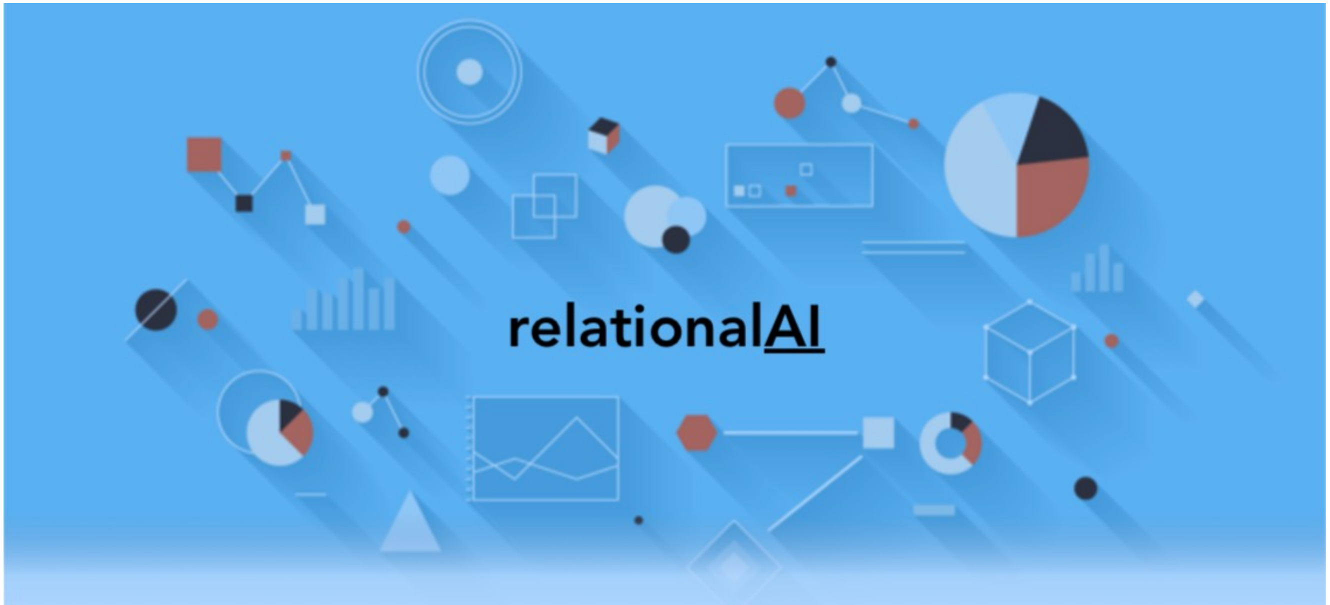


An Algorithmist's Take on Relational Algorithms (Intermediate Relational Algorithm Design)

Kirk Pruhs

Acknowledements to:

- Academic Network: Sungjin Im, Ben Moseley, Alireza Samadian, Yuyan Wang
- RAI folks: Mahmoud Abo-Khamis, Ryan Curtin, Hung Ngo

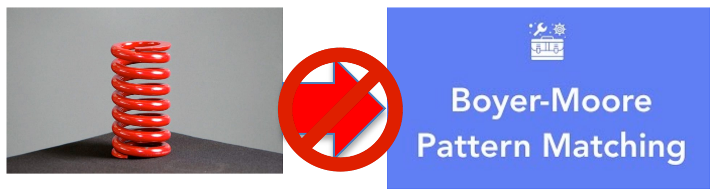
Recall:
Relational Algorithms

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

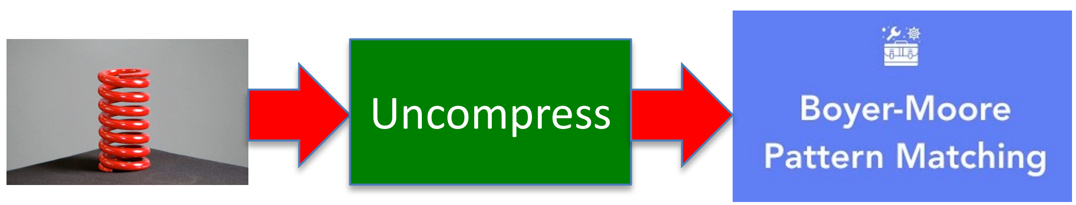
- Relational algorithms necessarily can not afford to join the tables

Analogous Situation

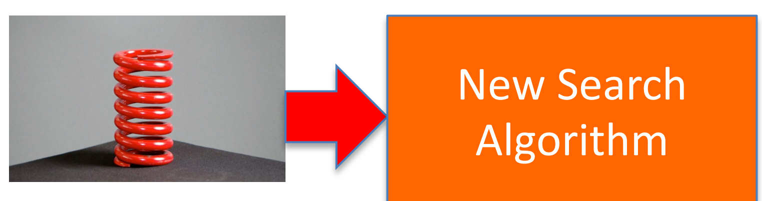
- Goal: StringSearch(Compressed String)



- Standard Approach: StandardAlgorithm(Uncompress(Compressed String))



- [SMTSA, CPM2000] NewSearchAlgorithm(Compressed String)



Intended Take Away Points

- Barrier to entry into relational algorithms is relatively low

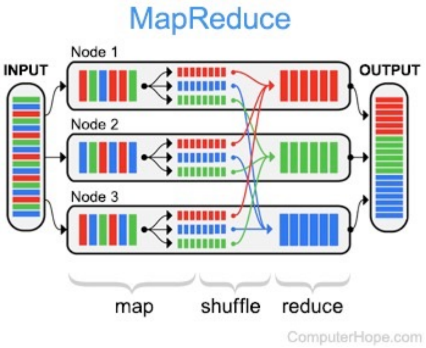
- Potentially interesting open algorithmic problems

- 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 a allow 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	
x	y
3	1
4	1
5	1
6	2

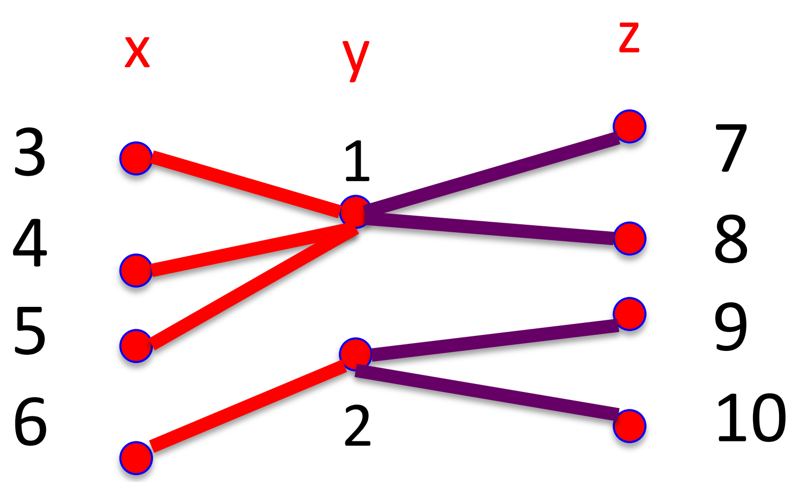
⋈

Join

Table B	
y	z
1	7
1	8
2	9
2	10

=

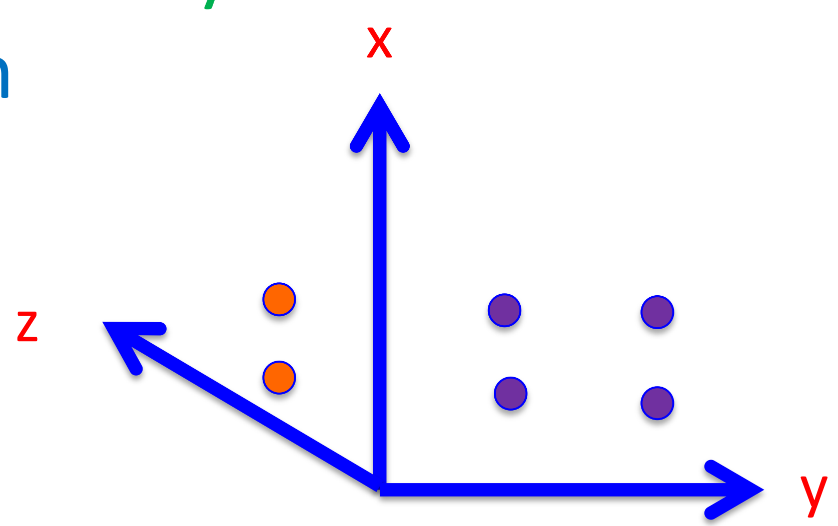
Design Matrix		
x	y	z
3	1	7
3	1	8
4	1	7
4	1	8
5	1	7
5	1	8
6	2	9
6	2	10



Intuitive Geometric View of a Join

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

- Join is intuitively the maximal inverse of projection



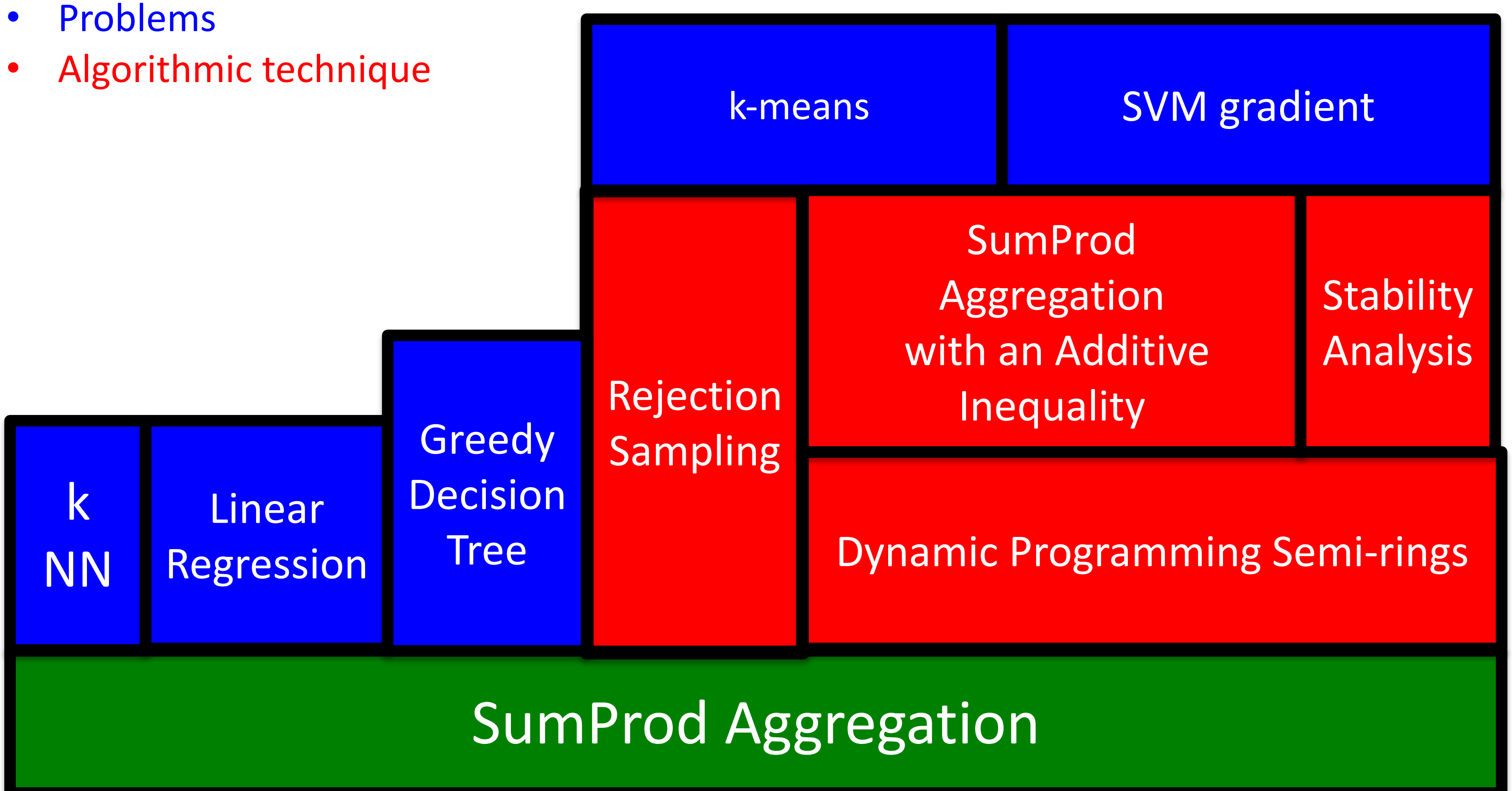
Necessary Background Before Getting Started Designing Relational Algorithms

- 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

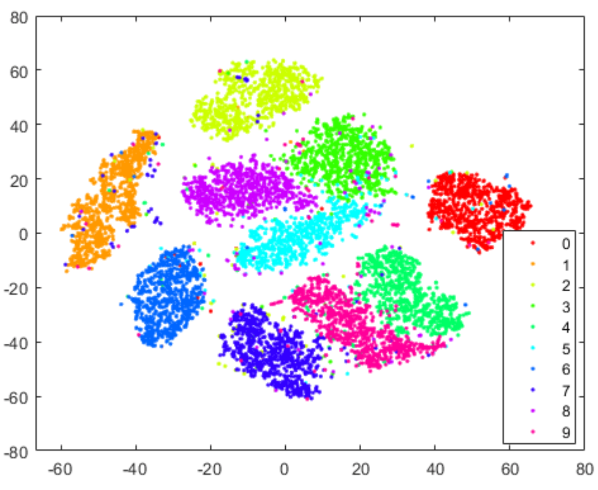


Sum-Product Query

- Mahmoud's view:
  - $\bigoplus_{x\_1, .. x\_k} \bigotimes_{x\_S} f_S(x_S)$
  - Where each S is conceptually a table with variables x_S
- My take abstracts out the tables
  - $\bigoplus_r \bigotimes_c f_c(M_{rc})$
  - Where r is a generic row, and c is a generic column, in the joined table M
  - Where $\bigoplus$ and $\bigotimes$ 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
  - $\bigoplus$ is addition
  - $\bigotimes$ is multiplication
  - $f_c(x) = 1$
  - So $\bigoplus_r \bigotimes_c f_c(M_{rc}) = \sum_r \prod_c 1$
  - 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

Design Matrix		
x	y	z
3	1	7
3	1	8
4	1	7
4	1	8
5	1	7
5	1	8
6	2	9
6	2	10

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 …

Design Matrix		
x	y	z
3	1	7
3	1	8
4	1	7
4	1	8
5	1	7
5	1	8
6	2	9
6	2	10

Necessary Background Before Getting Started Designing Relational Algorithms

- Graphic and geometric views of a join

- Sum-Product query

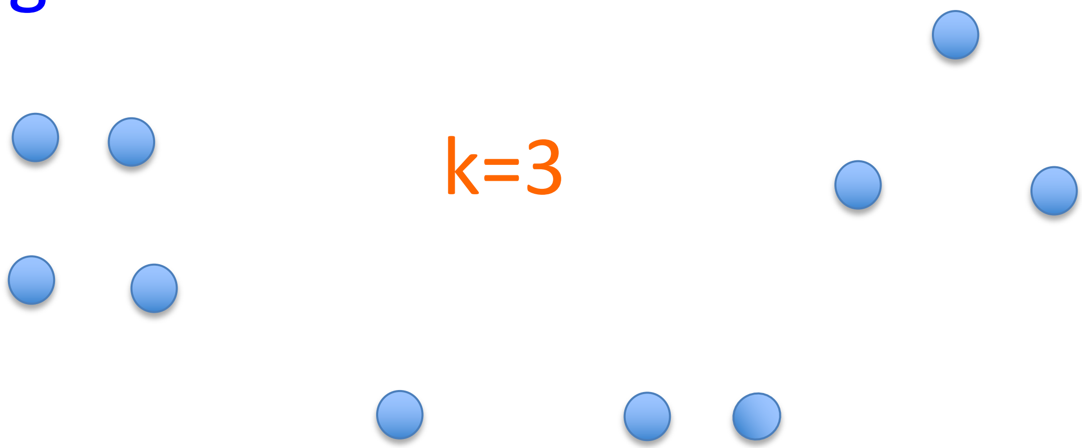
- Variable elimination algorithm

- How to use Sum-Product queries to develop algorithms

Illustrative Example Problem: k-means Clustering

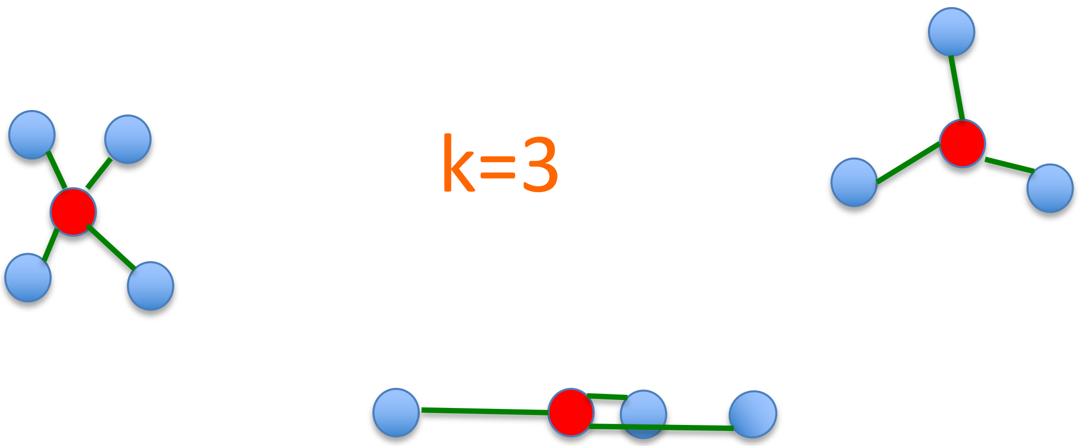
k-means Problem

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



k-means Problem

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



- Feasible solution: k centers/points $S_1, \ldots, S_k$
- Objective: Minimize aggregate 2-norm squared distances to nearest center
  - Min $\sum_{i \in [m]} \min_{j \in [k]} \langle\langle x_i - S_h \rangle\rangle$
  - Where $\langle\langle x_i - S_h \rangle\rangle$ 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

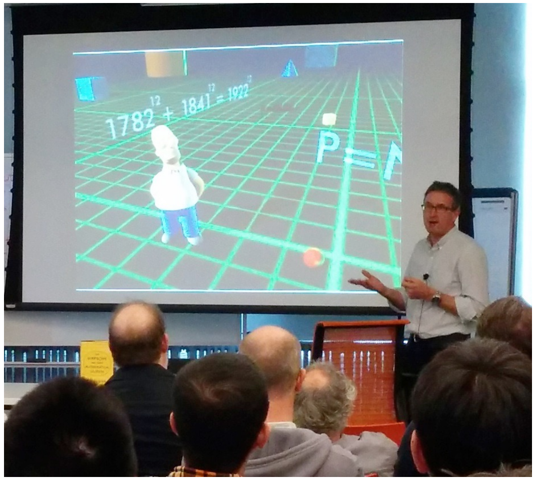
C. Design a relational algorithm that has some reasonable theoretical guarantee

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 $x_1, \ldots, x_n$
- 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 $(\sum_r \prod_c 1)$ on a Path Join

Input Table T1

F1	F2
1	1
2	1
3	3

Input Table T2

F2	F3
1	1
1	2
3	3

Input Table T3

F3	F4
1	1
2	1
3	3

Input Table T4

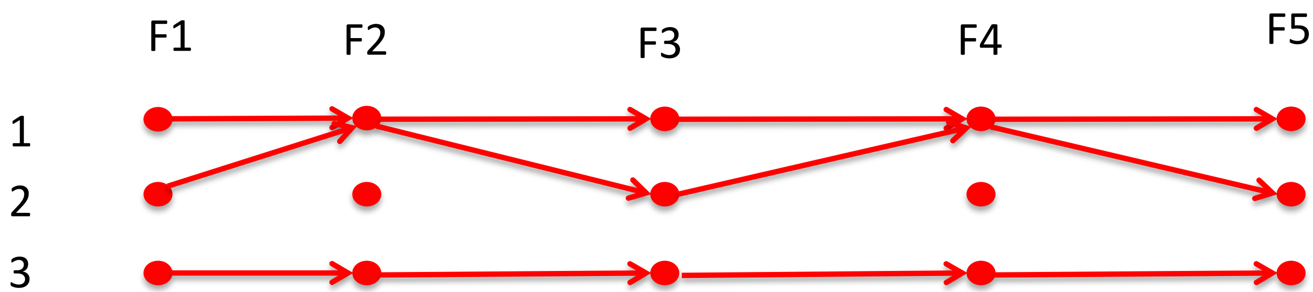
F4	F5
1	1
1	2
3	3

4

4

1

Computed by variable elimination algorithm



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

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 $\langle\langle x_i - S_1 \rangle\rangle$, 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

Input Table T1	
F1	F2
1	1
2	1
3	3

Input Table T2	
F2	F3
1	1
1	2
3	3

Input Table T3	
F3	F4
1	1
2	1
3	3

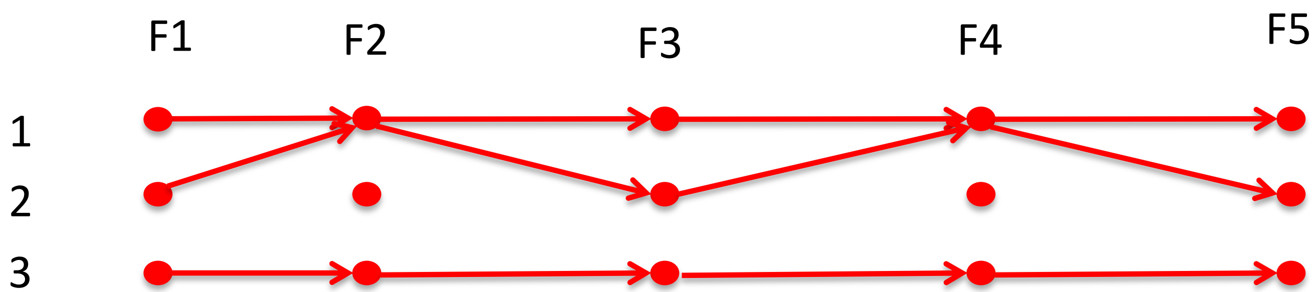
Input Table T4	
F4	F5
1	1
1	2
3	3

32

44

45

Computed by variable elimination algorithm

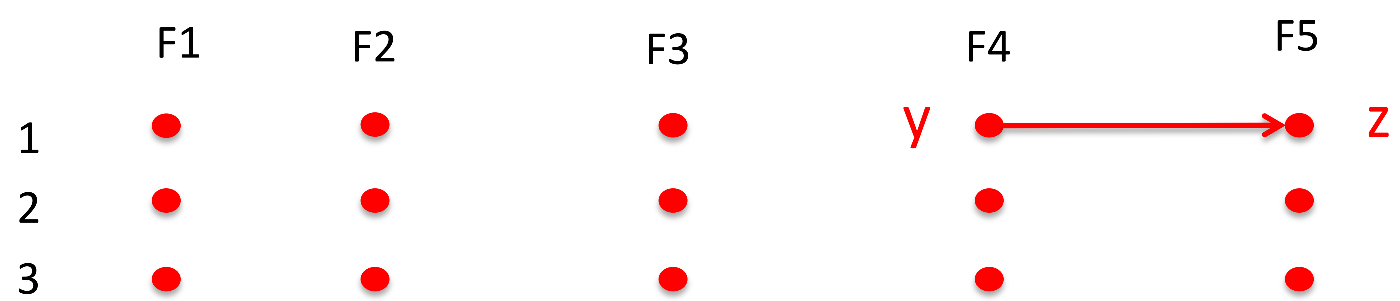


Each source to sink path can be viewed as a point in 5 dimensional space

Compute aggregate 2-norm squared on a path join

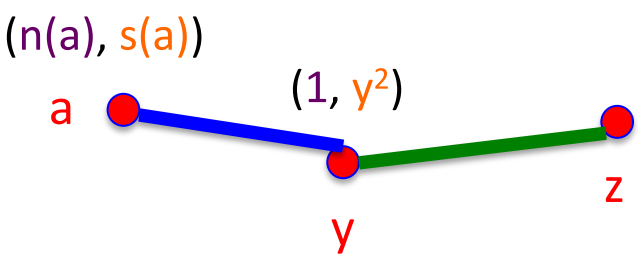
- Algorithm: Process edges left to right
  - $\mathrm{Sum}^2(z) = \mathrm{Sum}^2(y) + z^2 * \mathrm{numpaths}(y)$
  - $\mathrm{Numpaths}(z) = \mathrm{numpaths}(z) + \mathrm{numpaths}(y)$

- 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 $\oplus$ and $\otimes$ such that variable elimination yields

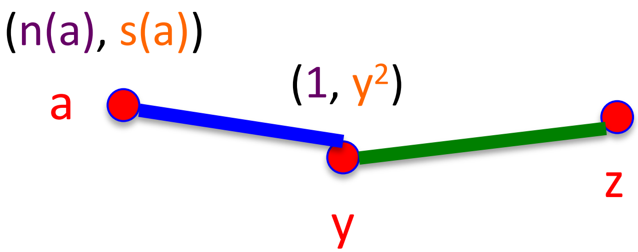


$$(n(z), s(z)) = (n(z), s(z)) \oplus [(n(a),s(a)) \otimes (1,y^2)]$$
$$= (n(z) + n(a), s(z) + s(a) + n(a)y^2)$$

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)

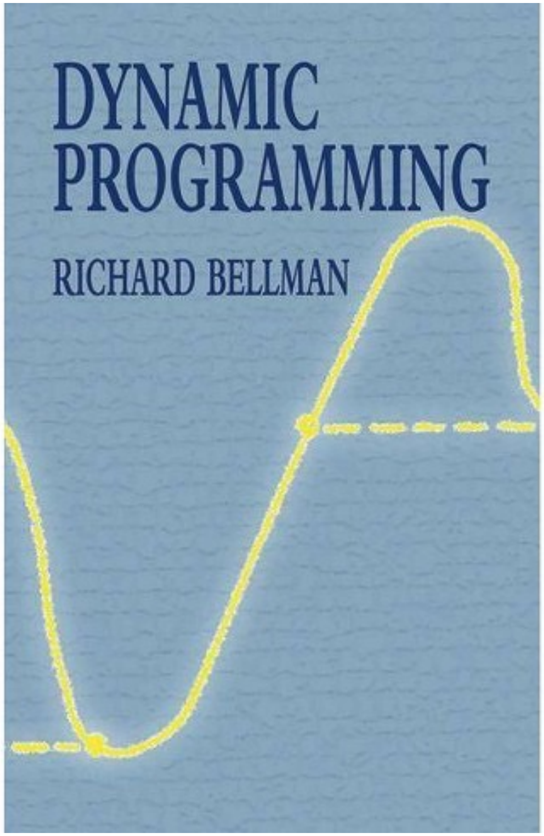


$$
\begin{aligned}
(n(z), s(z)) &= (n(z), s(z)) \oplus [(n(a), s(a)) \otimes (1, y^2)] \\
&= (n(z), s(z)) \oplus [(n(a), s(a) + n(a)y^2)] \\
&= (n(z) + n(a), s(z) + s(a) + n(a)y^2)
\end{aligned}
$$

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++

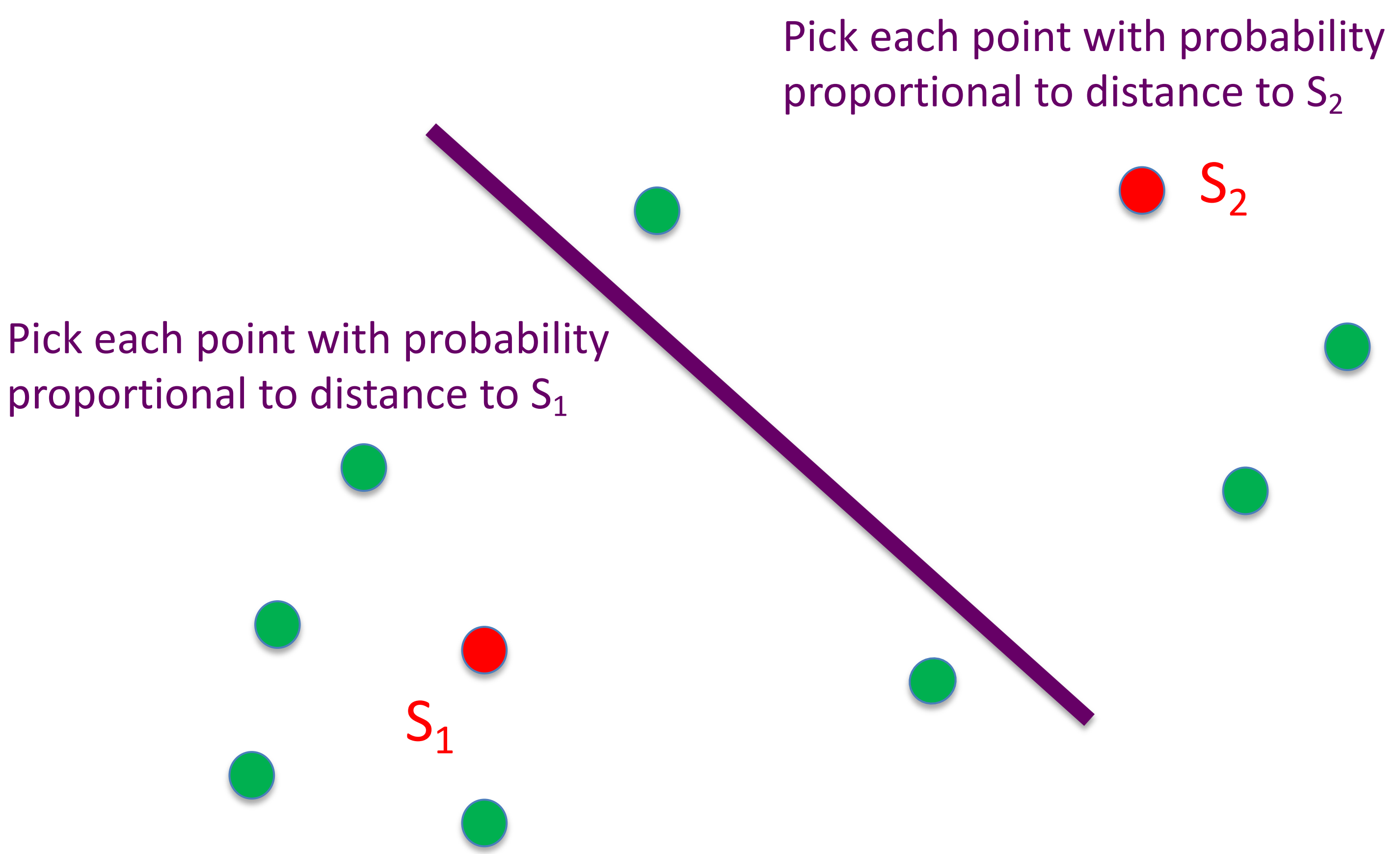
B. Design algorithm from scratch
   1. First try cross-product join
   2. Then try path join
   3. Then try express computation as sum-product query
   – Implementation of 2-means++
   – Implementation of 3-means++
   – Approximately counting points in a hypersphere
      • subroutine to our relational 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 $\langle\langle x_i - S_1 \rangle\rangle$
- Pick x_i as center S_3 with probability proportional to min($\langle\langle x_i - S_1 \rangle\rangle$, $\langle\langle x_i - S_2 \rangle\rangle$), the 2-norm squared distance to previous center

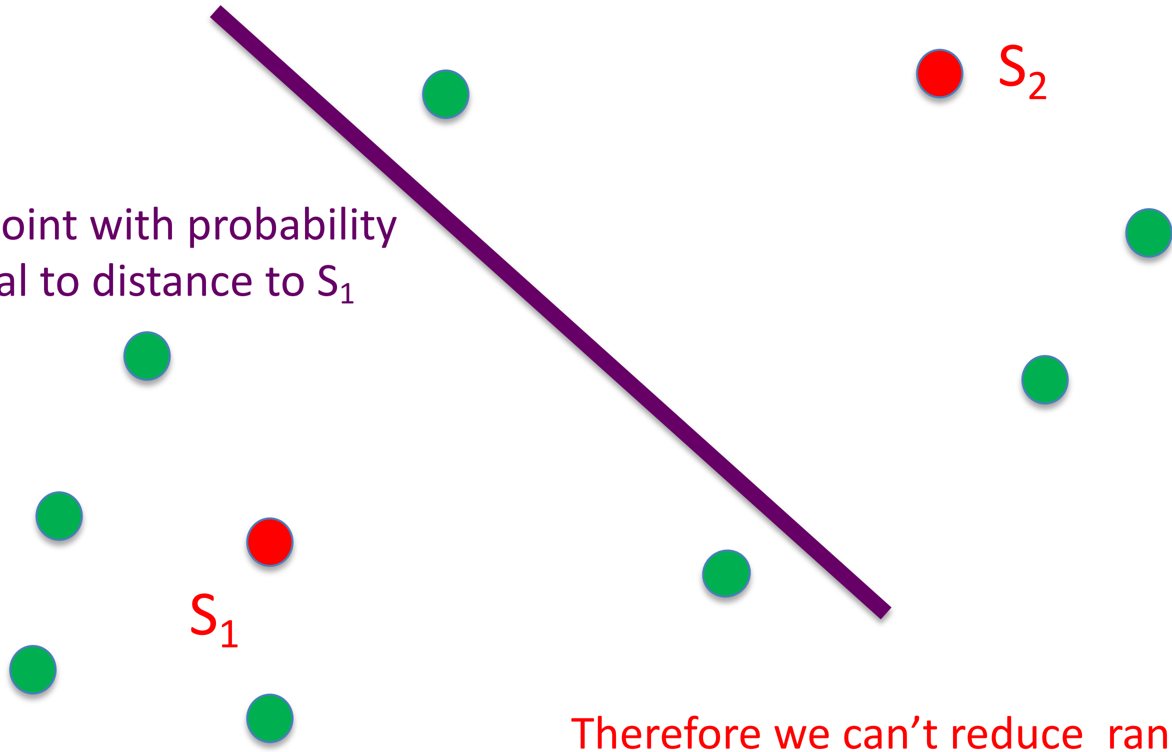
Picking S_3


Pick each point with probability proportional to distance to S_2
S_2
Pick each point with probability proportional to distance to S_1
S_1

Picking S_3

Theorem: Its NP-hard to compute aggregate distance of points to the dividing line even if tables are simple

Pick each point with probability proportional to distance to S_2

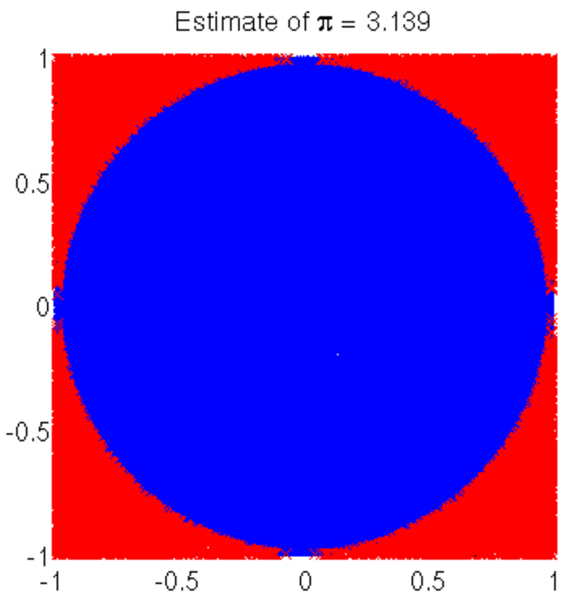


Pick each point with probability proportional to distance to S_1

Therefore we can't reduce random selection to summing

Digression: Rejection Sampling

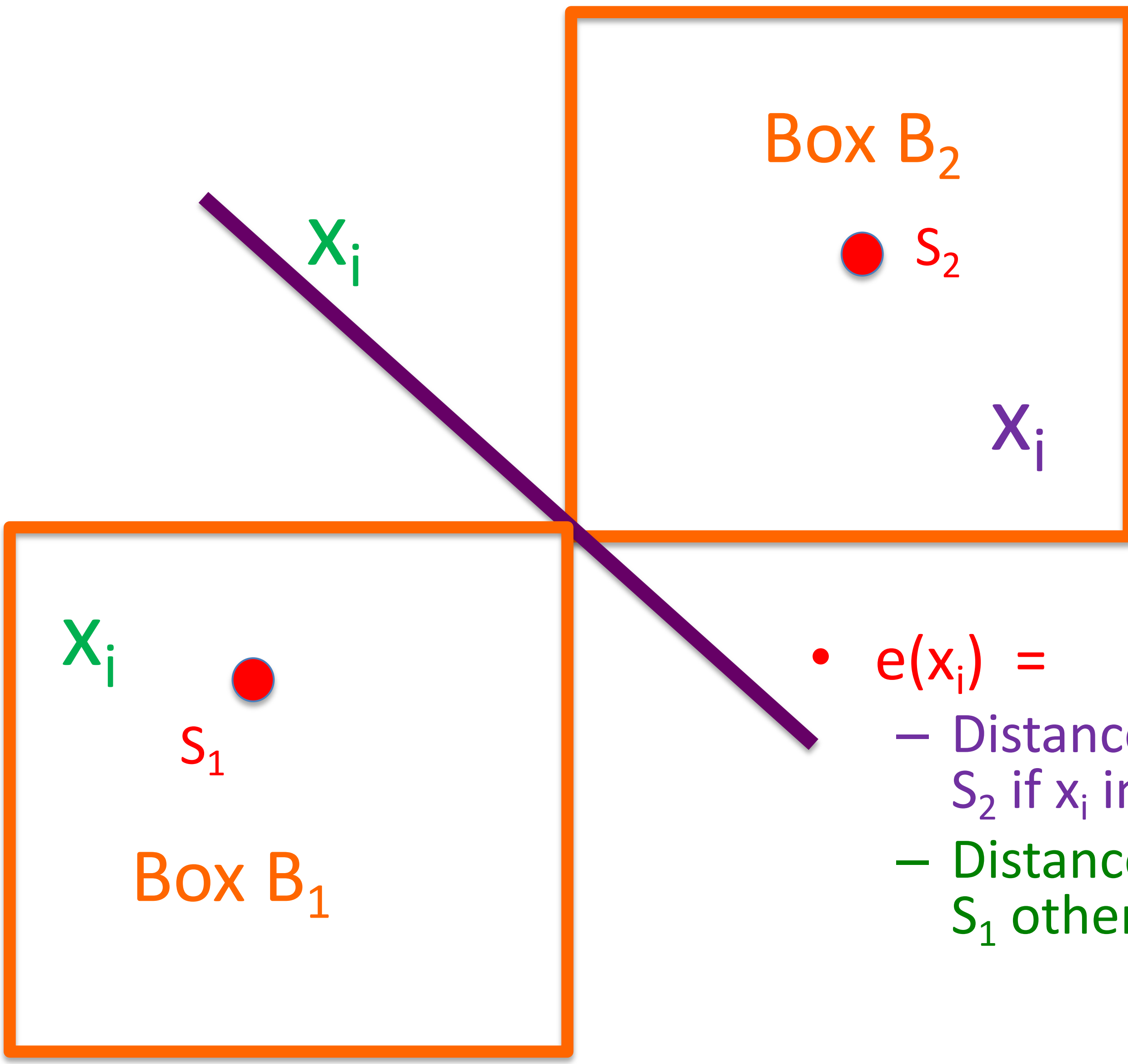
- Given a uniform sample over the red square:
  - Generate a uniform sample over the blue circle
  - Estimate area of the blue circle



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 e(r)/E
  - Accept r with probability h(r)/e(r) else resample
- Theorem: r is sampled with probability proportional to h(r) in expected time E/H

Defining Easy Distribution e



- $e(x_i)$ =
  - Distance from x_i to S_2 if x_i in box B_2
  - Distance from x_i to S_1 otherwise

Computing $E = \Sigma_i e(x_i)$ Using Sum-Product Query

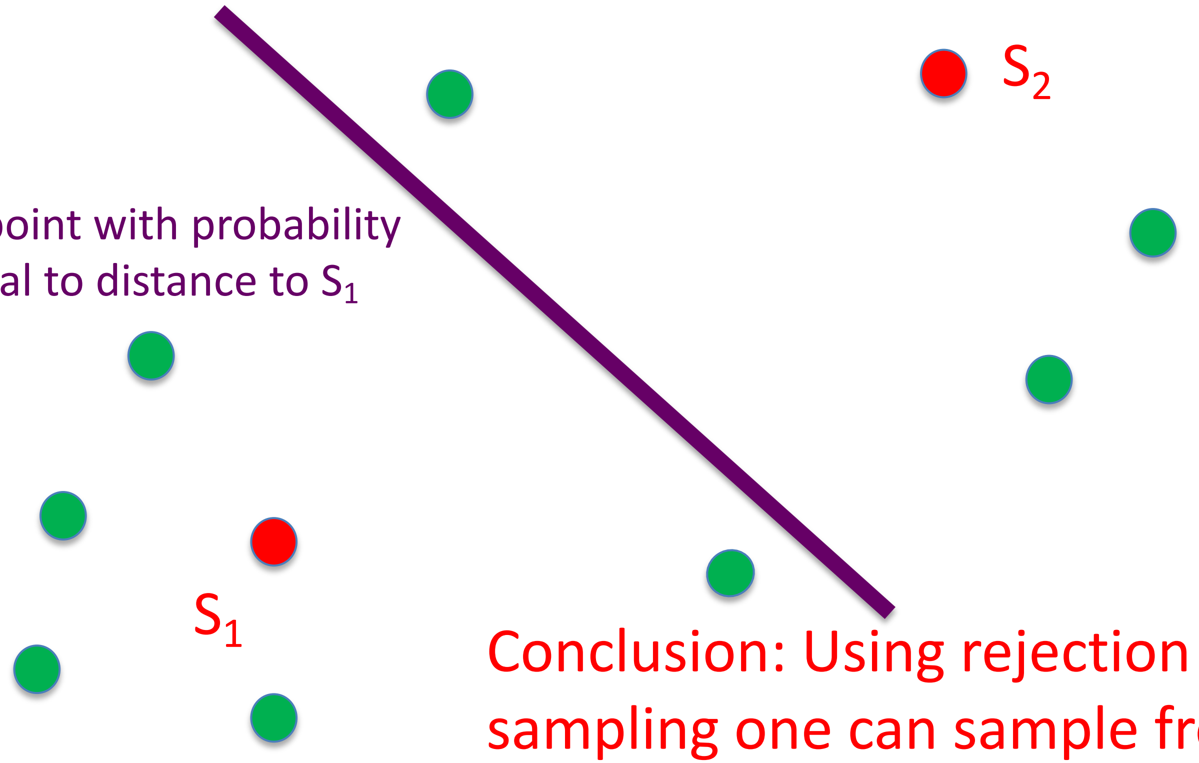
- $f_c(x) =$
  - $(x-S_2(c))^2$ if LB_Box2(c) < x < UB_Box2(c)
  - $(x-S_1(c))^2$ if otherwise

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

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 $\sum_c g_c(M_{rc}) <= R$
- 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 hypershere centered at origin
  - $\sum_r \prod_c 1$
  - For those r where $\sum_c M_{rc}^2 <= R$

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 lg 1/q + (1-q) log 1/(1-q)
  - q= probability label is 0
- This example: H(T) = (2/6)(lg 6/3) + (4/6)(log 6/4)

Table T				
U	V	W	X	Label
1	6	1	6	1
2	5	3	4	1
3	4	5	2	1
4	3	2	1	0
5	2	4	3	1
6	1	6	5	0

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 | C)
    - H(T | C) = Prob(C=0) H(T | C=0) + Prob(C=1) H(T | C=1)

Core of ID3 Algorithm

- Consider C is $U \leq 4$
- $H(T | C) = (2/3)\ H(T \mid U \leq 4) + (1/3)\ H(T \mid U > 4) =$
  - $(2/3)\ (\ 1/4 \lg 4 + 3/4 \lg 4/3) + (1/3)\ (1/2 \lg 2 + ½ \lg 2)$

Table T				
U	V	W	X	Label
1	6	1	6	1
2	5	3	4	1
3	4	5	2	1
4	3	2	1	0
5	2	4	3	1
6	1	6	5	0

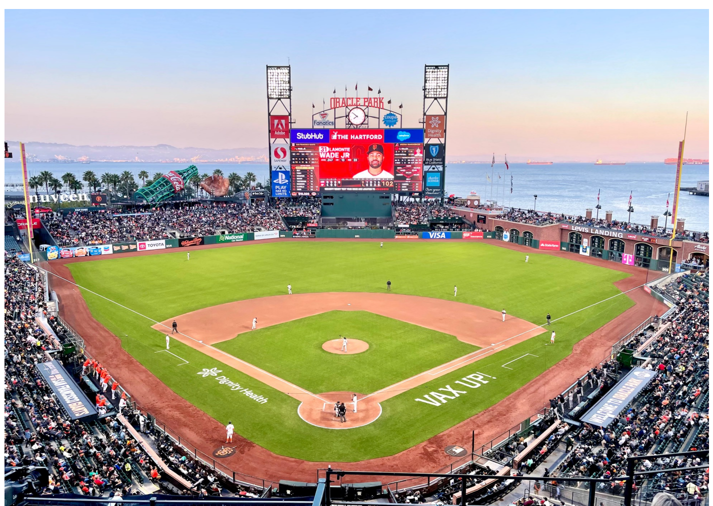
Core of ID3 Algorithm

- Find comparison C of the form:
  - attribute ≤ value
  - that gives maximum information
    - Equivalent to minimizing the resulting conditional entropy H(T | C)
    - H(T | C) = Prob(C=0) H(T | C=0) + Prob(C=1) H(T | C=1)

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

Workshop Outing

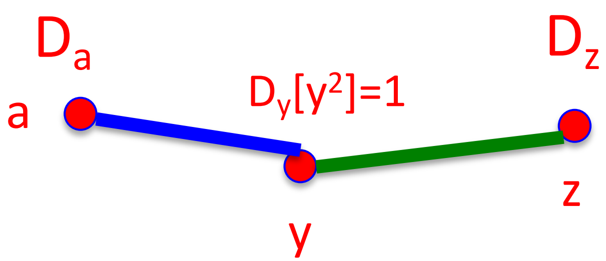
- San Francisco Giants baseball game Wednesday evening
- It is not important that you understand/like baseball
- Contact me if you are interested in joining




Thank you for listening
PSYCHIATRIC HELP 7¢
THE DOCTOR IS IN
SCHULZ
© PNTS

Dynamic Programming

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



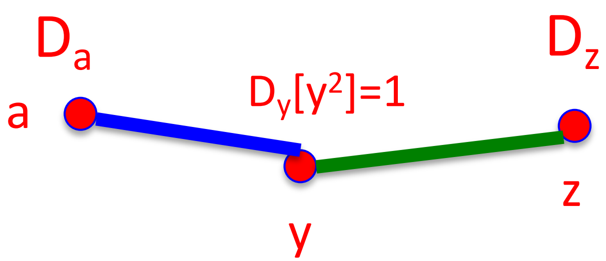
r^{th} entry of

$D_z = D_z \oplus (D_a \otimes D_y)$

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

Counting Points in Hypershere Centered at the Origin

- Dynamic Programming Semiring
  - $D_a \oplus D_b$ = 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^{th} entry of

$D_z = Dz \oplus (D_a \otimes D_y)$

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