Sparse Computing, Databases, and Semirings

Birds of a feather flock together

Altan Haan

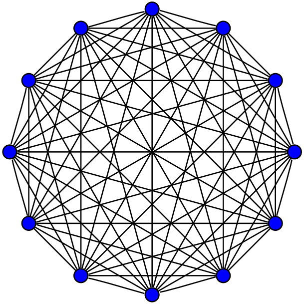
UC Berkeley

April 24, 2024

Outline

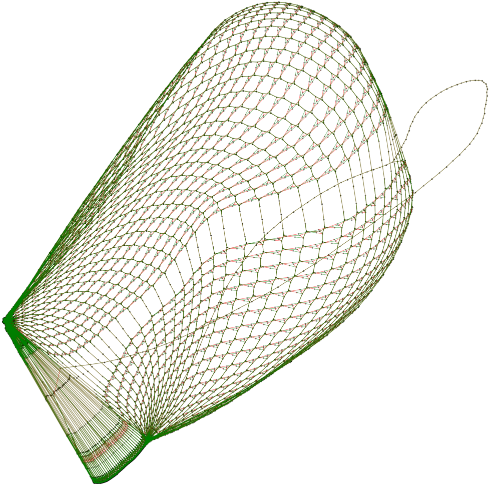
Table of Contents

Examples of sparsity

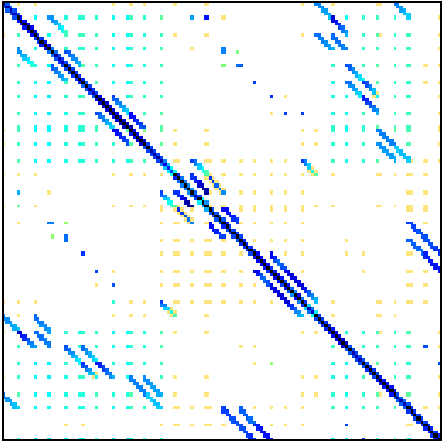


Complete graph on 12 vertices, not sparse!

https://commons.wikimedia.org/wiki/File:11-simplex_graph.svg



Examples of sparsity



`https://sparse.tamu.edu/Norris/torso1`

Examples of sparsity

name: char[32]	**salary**: uint32
bob	80000
alice	85000

	0	1	...	80000	...	85000	...	4294967296
a	0	0	0	0	0	0	0	0
⋮								
alice	0	0	0	0	0	1	0	0
⋮								
bob	0	0	0	1	0	0	0	0
⋮								
z...z	0	0	0	0	0	0	0	0

Navigating sparsity: graph traversal

Graph: $R(\mathsf{src}, \mathsf{dst})$ or adjacency matrix A. Find all pairs of nodes reachable via paths of length 3.

Navigating sparsity: graph traversal

Graph: $R(\text{src}, \text{dst})$ or adjacency matrix A. Find all pairs of nodes reachable via paths of length 3.

$$\text{relabel: } R_1(\text{src}, X_1), R_2(X_1, X_2), R_3(X_2, \text{dst}),$$

$$Q = \prod_{\text{src,dst}} R_1 \bowtie R_2 \bowtie R_3.$$

Navigating sparsity: graph traversal

Graph: $R(\text{src}, \text{dst})$ or adjacency matrix A. Find all pairs of nodes reachable via paths of length 3.

$$\text{relabel: } R_1(\text{src}, X_1), R_2(X_1, X_2), R_3(X_2, \text{dst}),$$

$$Q = \prod_{\text{src,dst}} R_1 \bowtie R_2 \bowtie R_3.$$

Using matrix A:

$$(A^2)_{ij} = \sum_k A_{ik} A_{kj} = \#\text{ of ways } i \text{ can reach } j \text{ in 2 hops}$$

$$Q_{i\ell} = (A^3)_{ij} = \sum_j (A^2)_{ij} A_{j\ell} = \sum_{k,j} A_{ik} A_{kj} A_{j\ell}.$$

Navigating sparsity: graph traversal

Find min cost of all paths of length (exactly) 3 between pairs of nodes.

Edge weights: $R(\mathsf{src}, \mathsf{dst}, W)$, or $A_{ij} \in \mathbb{R}$.

Navigating sparsity: graph traversal

Find min cost of all paths of length (exactly) 3 between pairs of nodes.

Edge weights: $R(\mathsf{src}, \mathsf{dst}, W)$, or $A_{ij} \in \mathbb{R}$.

$$Q(\mathsf{src}, \mathsf{dst}, W) = \gamma_{\{\mathsf{src}, \mathsf{dst}\}, \min(W_1+W_2+W_3)}(R_1 \bowtie R_2 \bowtie R_3).$$

Navigating sparsity: graph traversal

Find min cost of all paths of length (exactly) 3 between pairs of nodes.

Edge weights: $R(\mathsf{src}, \mathsf{dst}, W)$, or $A_{ij} \in \mathbb{R}$.

$$Q(\mathsf{src}, \mathsf{dst}, W) = \gamma_{\{\mathsf{src},\mathsf{dst}\},\min(W_1+W_2+W_3)}(R_1 \bowtie R_2 \bowtie R_3).$$

With A,

$$Q_{ij} = \min_j \left(\min_k A_{ik} + A_{kj} \right) + A_{j\ell} = \min_{j,k} A_{ik} + A_{kj} + A_{j\ell}.$$

$+$ *distributes* over min.

Suspicious...

Table of Contents

One algebra to rule them all

Definition

A *semiring* is a tuple $(S, +, \cdot, 0, 1)$ where

- 0 is the identity for $+$,
- 1 is the identity for $\cdot$,
- 0 is an annihilator for $\cdot$,
- $+$ and $\cdot$ are associative, and $+$ is commutative,
- $a \cdot (b + c) = a \cdot b + a \cdot c$,
- $(b + c) \cdot a = b \cdot a + c \cdot a$.

One algebra to rule them all

Definition

A *semiring* is a tuple $(S, +, \cdot, 0, 1)$ where

- 0 is the identity for $+$,
- 1 is the identity for $\cdot$,
- 0 is an annihilator for $\cdot$,
- $+$ and $\cdot$ are associative, and $+$ is commutative,
- $a \cdot (b + c) = a \cdot b + a \cdot c$,
- $(b + c) \cdot a = b \cdot a + c \cdot a$.

Examples

- $(\mathbb{B}, \vee, \wedge, \bot, \top)$,
- $(\mathbb{N}, +, \cdot, 0, 1)$,
- $(\mathbb{R}, \min, +, \infty, 0)$.

One algebra to rule them all

Definition

A *schema* Γ is a product of sets $A_1 \times \cdots \times A_n$. Each A_i is an *attribute*.

Definition

An S-*relation* is a function $R : \Gamma \to S$ with *finite support*, where S is a semiring. The support of R is the set $\{\mathbf{a} \in \Gamma \mid R(\mathbf{a}) \neq 0\}$.

One algebra to rule them all

Definition

A *schema* Γ is a product of sets $A_1 \times \cdots \times A_n$. Each A_i is an *attribute*.

Definition

An S-*relation* is a function $R : \Gamma \to S$ with *finite support*, where S is a semiring. The support of R is the set $\{\mathbf{a} \in \Gamma \mid R(\mathbf{a}) \neq 0\}$.

Examples

- $\mathbb{B}$-relations: database tables under set semantics,
- $\mathbb{N}$-relations: ... bag semantics,
- Trop$^+$-relations: shortest paths,
- a tensor $T \in \mathbb{R}^{d_1 \times \cdots \times d_n} \cong T : [d_1] \times \cdots \times [d_n] \to \mathbb{R}$.

S-relations were introduced by [GKT07] for provenance tracking.

S-relational algebra

Equijoin. $\Gamma(R_1) = \mathbf{A} \times \mathbf{A}_1$, $\Gamma(R_2) = \mathbf{A} \times \mathbf{A}_2$.

$$[\![R_1 \bowtie R_2]\!](\mathbf{a}, \mathbf{a}_1, \mathbf{a}_2) = R_1(\mathbf{a}, \mathbf{a}_1) \otimes R_2(\mathbf{a}, \mathbf{a}_2).$$

Union. Just entry-wise addition.

Projection. $\Gamma(R) = \mathbf{A} \times \mathbf{A}'$.

$$[\![\prod_{\mathbf{A}'} R]\!](\mathbf{a}') = \bigoplus_{\mathbf{a} \in \mathbf{A}} R(\mathbf{a}, \mathbf{a}').$$

S-relational algebra

Equijoin. $\Gamma(R_1) = \mathbf{A} \times \mathbf{A}_1$, $\Gamma(R_2) = \mathbf{A} \times \mathbf{A}_2$.

$$[\![R_1 \bowtie R_2]\!](\mathbf{a}, \mathbf{a}_1, \mathbf{a}_2) = R_1(\mathbf{a}, \mathbf{a}_1) \otimes R_2(\mathbf{a}, \mathbf{a}_2).$$

Union. Just entry-wise addition.

Projection. $\Gamma(R) = \mathbf{A} \times \mathbf{A}'$.

$$[\![\prod_{\mathbf{A}'} R]\!](\mathbf{a}') = \bigoplus_{\mathbf{a} \in \mathbf{A}} R(\mathbf{a}, \mathbf{a}').$$

This is well-defined because R has finite support!

Brief note on datalog over semirings

What about datalog over semirings? Relevant for iterative computations: shortest paths, BFS, betweenness centrality, PageRank, eigensolvers, ...

We need to figure out:

- Some notion of (partial) order?
- Monotonicity with respect to this order?
- Infinite Herbrand universe, how to prove termination?
- Generalization of subtraction? if we want semi-naïve.

Brief note on datalog over semirings

See the datalog$^\circ$ paper by Mahmoud and others [AKNP$^+$22].

Basic takeaways:

- Separate partial order from semiring natural order: $a \sqsubseteq b \not\equiv \exists c.\ a \oplus c = b$,
- Be careful about $\perp$ in poset vs 0 in semiring,
- Semiring needs certain algebraic properties (stability) for convergence,
- Semi-naïve needs $+$ to be idempotent, poset must form complete distributive lattice.

Table of Contents

Approaches to optimization

Traditionally, (DB) queries and tensor programs are optimized differently.

Query optimization. High-level, fixed set of operators e.g. $\sigma, \bowtie, \prod$. E-graphs good! Opt's include: physical operator choice (nested loop, hash, merge); join reordering; aggregate push-down. Cost estimation is sophisticated. See [Gra95].

Tensor optimization. Lower-level, nested loops over variables. Variables are harder to handle, not impossible but different techniques like polyhedral analysis show up.

Factorization - thinking in terms of variables

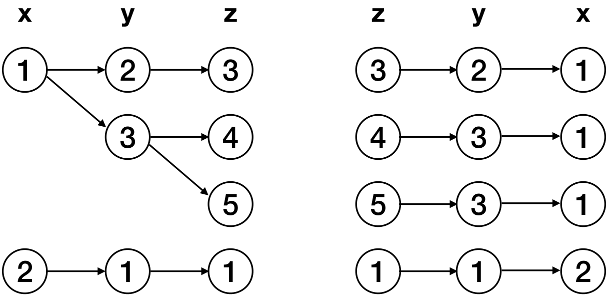
x	y	z
1	2	3
1	3	4
1	3	5
2	1	1

Table: Listing representation of $R(x, y, z)$.

Factorization - thinking in terms of variables

x	y	z
1	2	3
1	3	4
1	3	5
2	1	1

Table: Listing representation of $R(x, y, z)$.



Factorization - thinking in terms of variables

As sets,

$$\begin{aligned} R = &\{1\} \times (\{2\} \times \ \{3\} \\ &\quad \cup \ \{3\} \times (\{4\} \\ &\qquad \cup \ \{5\})) \\ \cup\, &\{2\} \times \ \{1\} \times \ \{1\}. \end{aligned}$$

Distributivity again! $\times$ distributes over $\cup$.

Factorization

Factorization is not new in sparse computing.

- adjacency list
- compressed sparse row/column (CSR/CSC) format
- ELL format (row $\rightarrow$ fixed $\#$ of nonzero cols) - bounded degree graph

Factorizing saves time, not just space

Assume $M = A = B$, but this query is valid in general.

$$C = \sum_{i,j,k} M(i,j) \cdot A(j,k) \cdot B(i,k).$$

Typical DB: *binary join plan*, e.g.

$$C = \text{COUNT}\ (M \bowtie A) \bowtie B.$$

Factorizing saves time, not just space

Assume $M = A = B$, but this query is valid in general.

$$C = \sum_{i,j,k} M(i,j) \cdot A(j,k) \cdot B(i,k).$$

Typical DB: *binary join plan*, e.g.

$$C = \text{COUNT } (M \bowtie A) \bowtie B.$$

All binary plans take worst-case $\Omega(N^2)$ time, where N is the $\#$ of edges.

Exercise: prove that $C = O(N^{1.5})$.

Factorizing saves time, not just space

What if we tried to write dense tensor algebra code?

```
C = 0
for i in range(I):
    for j in range(J):
        for k in range(K):
            C += M[i,j] * A[j,k] * B[i,k]
```

Factorizing saves time, not just space

Now let's try to make it sparse.

```
C = 0
for i in M.i ∩ B.i:
    for j in M.j ∩ A.j:
        for k in B.k ∩ A.k:
            C += M[i,j] * A[j,k] * B[i,k]
```

Intuition: the loop body expression is only nonzero when all inputs have values for the given i, j, k.

Factorizing saves time, not just space

Now let's try to make it sparse.

```
C = 0
for i in M.i ∩ B.i:
    for j in M.j ∩ A.j:
        for k in B.k ∩ A.k:
            C += M[i,j] * A[j,k] * B[i,k]
```

Intuition: the loop body expression is only nonzero when all inputs have values for the given i, j, k.

Important: if $i_0 \notin M.i \cap B.i$, then don't iterate over any tuples (i_0, j, k)! $M.j$ and $B.k$ depend on the current value of i.

Factorizing saves time, not just space

```
# M: i -> j -> value
# A: j -> k -> value
# B: i -> k -> value
C = 0
for i in M:
    M_j = M[i]; B_k = B[i]?
    for j in M_j:
        A_k = A[j]?
        for k in B_k:
            A_k[k]?
            C += M[i][j] * A[j][k] * B[i][k]
```

Factorizing saves time, not just space

```
# M: i -> j -> value
# A: j -> k -> value
# B: i -> k -> value
C = 0
for i in M:
    M_j = M[i]; B_k = B[i]?
    for j in M_j:
        A_k = A[j]?
        for k in B_k:
            A_k[k]?
            C += M[i][j] * A[j][k] * B[i][k]
```

If we make sure to iterate over the smaller set in every intersection, **we obtain the worst-case optimal runtime** $O(N^{1.5})$. This is the Generic Join algorithm [NRR13].

Factorizing saves time, not just space

Lots of exciting research in last decade from DB theory folks:

- Factorized Database (FDB) research: Dan Olteanu's group [OZ12]
- Functional Aggregate Queries (FAQ): RelationalAI folks [KNR23]

Table of Contents

Where do we go next?

In no particular order,

- Bring high-level query planning and low-level loop optimization closer together. Build a unified optimizer. See Free Join [WWS23], SQDLite [SSS23] for initial steps.
- Figure out how to efficiently **parallelize** factorized joins. Build/exploit factorized statistics? Fine-grained parallelism? Need to handle skew.
- Support more "exotic" tensor programs. Allow affine arithmetic on variables. How does this affect optimality guarantees? How do we optimize this?

Thank you!

Questions?

References I

Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang, *Convergence of datalog over (pre-) semirings*, Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (New York, NY, USA), PODS '22, Association for Computing Machinery, 2022, p. 105–117.

Todd J. Green, Grigoris Karvounarakis, and Val Tannen, *Provenance semirings*, Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (New York, NY, USA), PODS '07, Association for Computing Machinery, 2007, p. 31–40.

Goetz Graefe, *The cascades framework for query optimization.*, IEEE Data(base) Engineering Bulletin **18** (1995), 19–29.

Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra, *Faq: Questions asked frequently*, 2023.

References II

Hung Q. Ngo, Christopher Re, and Atri Rudra, *Skew strikes back: New developments in the theory of join algorithms*, 2013.

Dan Olteanu and Jakub Závodný, *Factorised representations of query results: size bounds and readability*, Proceedings of the 15th International Conference on Database Theory (New York, NY, USA), ICDT '12, Association for Computing Machinery, 2012, p. 285–298.

Maximilian Schleich, Amir Shaikhha, and Dan Suciu, *Optimizing tensor programs on flexible storage*, Proc. ACM Manag. Data **1** (2023), no. 1.

Yisu Remy Wang, Max Willsey, and Dan Suciu, *Free join: Unifying worst-case optimal and traditional joins*, Proc. ACM Manag. Data **1** (2023), no. 2.