EmptyHeaded: Worst-case Optimal Join Processing on Graphs

Chris Aberger, Andres Nötzli,
Kunle Olukotun, and Christopher Ré
Stanford University

EMPTYHEADED

Specialized Graph Engines

- Green-Marl
- PowerGraph
- Graph-X
- Snap
- FlockDB
- StingerGraph
- Pregel
- Galois
- Ligra
- Neo4j
- SociaLite
- Tao
- PGX
- Giraph

EmptyHeaded is not another graph engine!

EMPTYHEADED

Rethinking Join Processing

Shock: For Joins and Graph Patterns, **Relational Algebra** is suboptimal, but **Boolean Algebra** can be optimal.

Part 1. Theory. Optimality Claim

- Worst-case optimal Join Processing. [PODS12,SIGMODR14]

Part 2. Hardware. Optimize Boolean Algebra [Arxiv15]

- Modern processors **love** Boolean Algebra
  - SIMD is the new Moore's Law!
  - 8x over last 4 generations, more soon.
    - Linear (hardware) scaling.
- Challenge: Cope with ***Skew***!

EMPTYHEADED

Joins Joins Joins!

R(A,B) JOIN T(B,C)

A	B
0	1
2	1

B	C
1	3
1	4

=

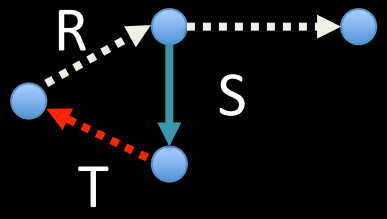
A	B	C
0	1	3
0	1	4
2	1	3
2	1	4

Reminder: The worst case output size a join on two tables of size N is N^2.

Join Queries on Graphs

Data: R(A,B), S(A,C), and T(B,C)

Today: graph where edges colored R,S,T.



Nodes are data values.

$Q_1 = \text{Join}(R,S)$

Acyclic query!

"triples of nodes on a path of length 2 that goes via R then S"

$Q_2 = \text{Join}(R,S,T)$

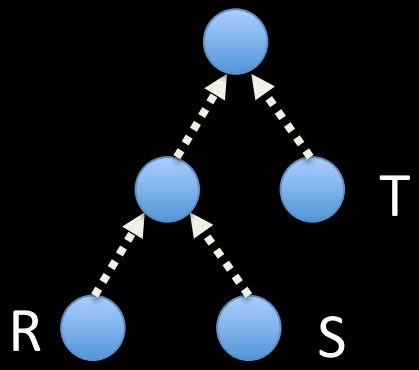
Cyclic query!

"triples of nodes that form R-S-T triangles"

Joins Since System R

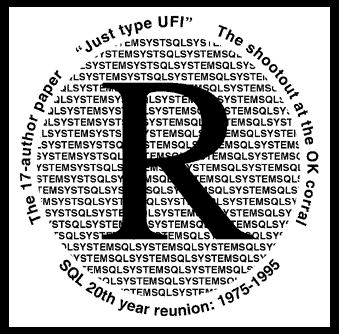
R(A,B), S(A,C), T(B,C)

Join(R,S,T) = { (a,b,c) : (a,b) in R, (a,c) in S, (b,c) in T}



Join(R,S,T) = Join(Join(R,S),T)

Left deep query plan!



System R searches through **pairwise** joins

For 40+ years, major commercial database use System-R style optimizer.

Nugget: *"DBs have been asymptotically suboptimal for the last 4 decades…"*

Background: Triangles [Alon 80, Loomis-Whitney 49]

Data: R(A,B), S(A,C), and T(B,C)

Let Q be Join(R,S,T) = "R-S-T triangles"

If R,S,T contain ≤ N tuples, how big can |Q| be?

R(A,B), S(A,C)

$|\text{Join}(R,S)| \leq N^2$

T(B,C)

R covers B, S covers C

$|\text{Join}(R,S,T)| \leq N^2$

Correct asymptotic: $|Q|$ in $\Theta(N^{3/2})$

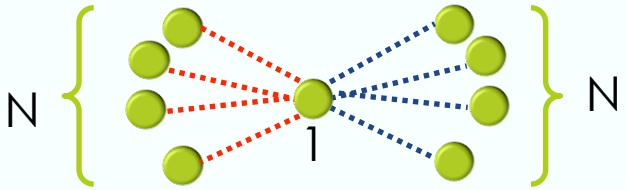
Can we compute Q in time $O(N^{3/2})$?

Pairwise Joins are Suboptimal

R(A,B), S(B,C),T(A,C)

Data

R=[N] x {1}
S ={1} x [N]
T = {1} x [N]



Data in
R and S

| R | = | S | = | T | =N

[N]={1,…,N}

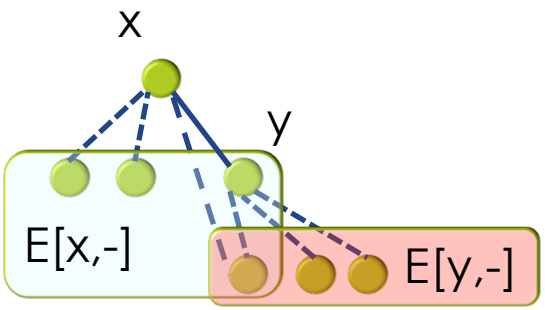
JOIN(R,S) = [N]x {1} x [N]
| Join(R,S) | = N^2
DB is *toast*!

Panic! A simple modification:
any pairwise join plan takes $\Omega(N^2)$

Simple algorithm.

Input (V,E)

For each x in V
  For each y in E[x,-]
    Q = Q ∪ {(x,y)} x Intersect(E[x,-],E[y,-])



Intersect(X,Y) **can be done** in time min{|X|,|Y|}.

Running time can be bounded by…

$$\sum_{x \in V} \sum_{y \in V} \min\{|E[x,-]|, |E[y,-]|\} |E[x,y]|$$

Let's upper bound this by $|E|^{3/2} = N^{3/2}$

This proof is simple!

$$\sum_{x \in V} \sum_{y \in V} \min\{|E[x,-]|, |E[y,-]|\} |E[x,y]|$$

Use geometric mean inequality.
min{|X|, |Y|} ≤ |X|$^{1/2}$ |Y|$^{1/2}$

$$\leq \sum_{x \in V} \sum_{y \in V} |E[x,-]|^{1/2} |E[y,-]|^{1/2} |E[x,y]|$$

Just rearrange terms…

$$= \sum_{x \in V} |E[x,-]|^{1/2} \sum_{y \in V} |E[y,-]|^{1/2} |E[x,y]|$$

Show no larger than $N^{3/2}$ in which $N = |E|$

$$\sum_{x \in V} |E[x,-]|^{1/2} \sum_{y \in V} |E[y,-]|^{1/2} |E[x,y]|$$

Using Cauchy-Schwarz

$$\sum_{i=1}^{n} x_i y_i \leq \sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{j=1}^{n} y_j^2}$$

$$\leq \sum_{x \in V} |E[x,-]|^{1/2} \sqrt{\sum_{y \in V} |E[y,-]|} \sqrt{\sum_{y \in V} |E[x,y]|}$$

Definitions of E[x,-] and $|E| = N$

$$= \sqrt{N} \sum_{x \in V} |E[x,-]|^{1/2} |E[x,-]|^{1/2}$$

$$= \sqrt{N} \sum_{x \in V} |E[x, -]|^{1/2} |E[x, -]|^{1/2}$$

Simplifying.

$$= \sqrt{N} \sum_{x \in V} |E[x, -]| = N^{3/2}$$

For each node x in V
For each y in E[x,-]
Q = Q ∪ {(x,y)} x Intersect(E[x,-],E[y,-])

Optimality: Complete graph on $N^{1/2}$ Nodes has ~ N edges and ~ $N^{3/2}$ triangles

Remarkable Observation

For the Triangle query,

Boolean algebra+ "map" is faster than relational algebra (theory now, empirical next)

Optimality bound followed as long as "min property":

Intersect(X,Y) can be done in time min{|X|,|Y|}

This is the key issue in each one of these algorithms
Galloping in the 70s, or
LeapFrog Trie Join (LogicBlox) in the 2010s.

How do we generalize to joins?

Fractional Hypergraph Covers

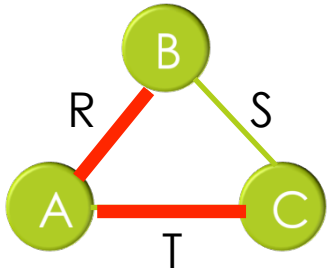
Given a hypergraph H=(V,E) a **fractional edge cover** is x : E →R such that x ≥ 0 and for each v in V we have $\Sigma_{e : v \text{ in } e} x(e) \geq 1$

Ex: R(A,B),S(B,C),T(A,C).

x(R,S,T)=(1,0,1) … or…
x(R,S,T)=(0.5, 0.5, 0.5)

x(R) + x(T) ≥ 1 // cover for A
x(R) + x(S) ≥ 1 // cover for B
x(S) + x(T) ≥ 1 // cover for C

We think of a **query** as hypergraph to cover.



Size bounds [GM05, AGM08]

Fix a query Q=(V,E).
Let **N** be a tuple of $|E|$ positive integers.

Define S(Q, **N**) be the maximum size of Q subject to $|R_e| \leq N_e$

Thm [Atserias, Grohe, Marx FOCS08]: Given any hypergraph cover x for (V,E) then

$$S(Q,\mathbf{N}) \leq \Pi_{e \text{ in } E} |R_e|^{x(e)}$$

Triangle: $|R| = |S| = |T| \leq N$,
$x(R)=x(S)=x(T)=0.5 \rightarrow N^{1.5}$

One more example.

R(A,B,C),S(A,B,D),T(A,C,D),U(B,C,D)

$$x(R) = x(S) = x(T) = x(U) = 1/3$$

Output size is $O(N^{4/3})$.
More joins needs to be done faster?!?

Known since
Loomis-Whitney (1940s Geometers!)

AGM's result.

Atserias, Grohe, and Marx (**AGM**) allow one to write a linear program that ***tightly bounds*** the output size of ***any*** join query.

Proof using Han/Shearer's lemma
(non constructive)

Compute the output in
upper bound time?

We would call this ***worst-case optimal***

Ngo, Porat, Ré, and Rudra *(PODS 2012)*

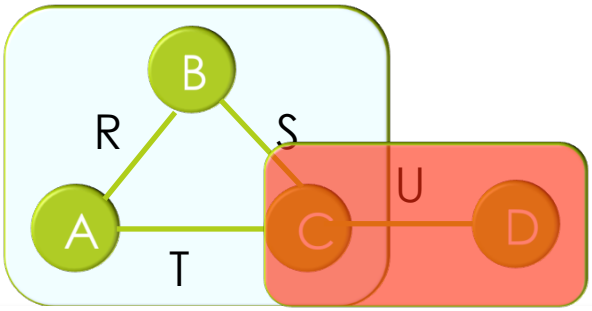
1st algorithm for joins with optimal worst-case runtime
(experts: optimal data complexity)

We show **AGM's** fractional cover inequality is equivalent to the **Bollabás-Thomason** inequality from geometry.

Algorithm: Only Map and Boolean Algebra. [Ngo, Ré, Rudra SIGMOD Record14]

One Example

R(A,B),S(B,C),T(A,C),U(C,D).



x(R,S,T,U) = (1,0,0,1) is optimal N^2 bound

Can "factor the Ds" to get : $N^{3/2}$ + Output.
(i.e., Generalized Hypertree Decomposition)

|Output| could be N^2 but could be much smaller!
EmptyHeaded uses this optimization.

Key Theory Ideas

Relational algebra is suboptimal but **Boolean algebra** can be optimal.

- "min property" for intersections.

EMPTYHEADED

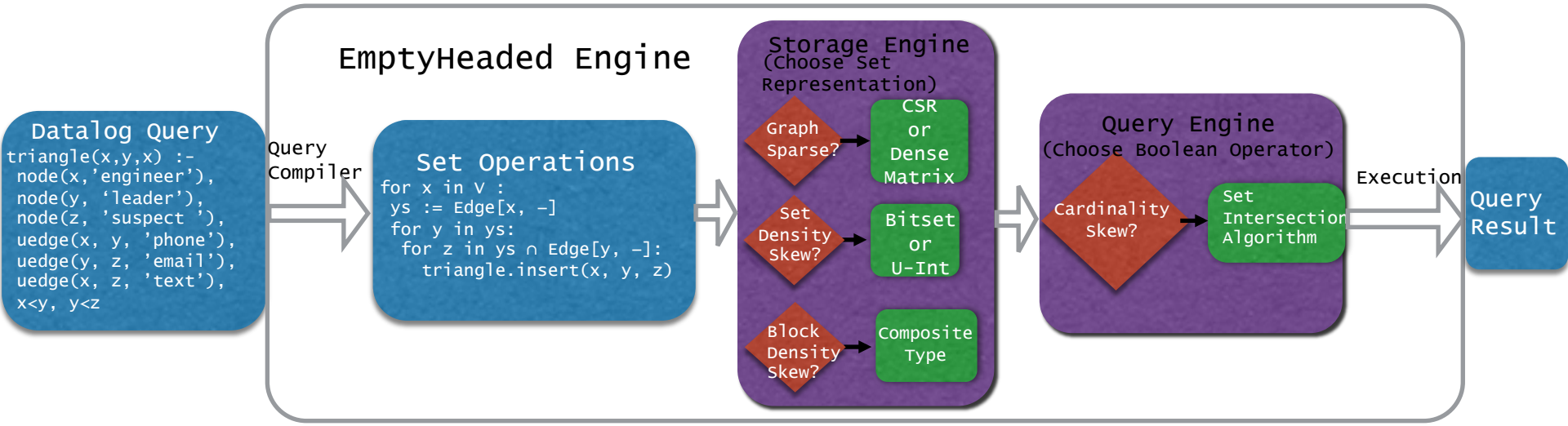
Generalized hyper-graph decompositions (GHDs) yield even better runtimes.

- A generalization of tree decompositions.
  - *"Run NPRR on Bags, Yannakakis between them"*
- With Afrati, Ullman, Joglekar, and Salihoglu recover some "optimal" parallelism results in multiround setting.

EMPTYHEADED

Part 2: The engine. It's fast.

EmptyHeaded



EmptyHeaded

EH uses the following operations.

Operation	Description
Edge[x,-]	{ y | (x,y) in Edge }
Edge[-,y]	{ x | (x,y) in Edge }
for x in xs	Iterates through the elements x of set xs
Intersect(xs,ys)	The intersection of xs and ys

```
For x in V
    For y in E[x,-]
        For z in Intersect(E[x,-],E[y,-])
            Out += (x,y,z)
```

SIMD, and the trend behind it.

Coping with **skew** is key challenge.

Performance #s. It's fast.

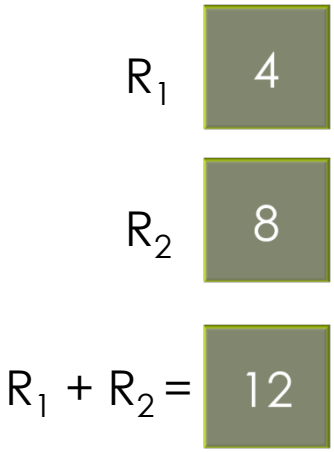
SIMD Processing

Single Instruction Multiple Data (SIMD)

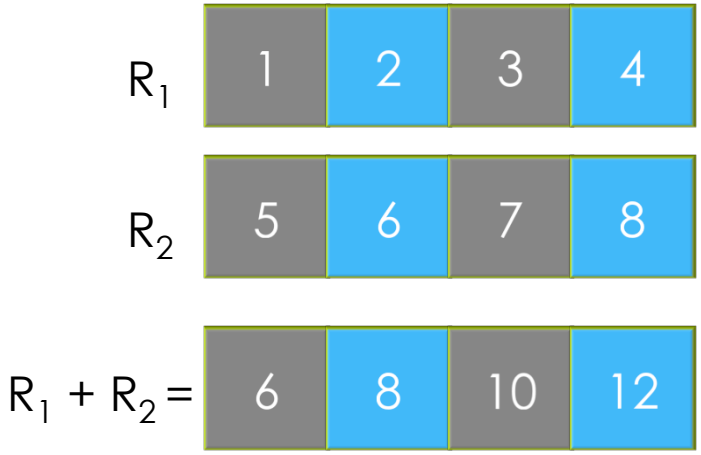
Multiple processing elements that perform the same operation on multiple data items ***simultaneously***

- Exploits **data-level** parallelism, not concurrency
- Happens in one clock cycle (more or less)

Scalar Register Addition



SIMD Register Addition – width 4



Almost 4x faster!

SIMD Trend

Evolution of multi-core:

- Single Core
- Hyper Threading (2002)
- Dual Core(2006)
- Multi Core (2008)
- Many Core (2011)

Future processors: many cores, wider SIMD.

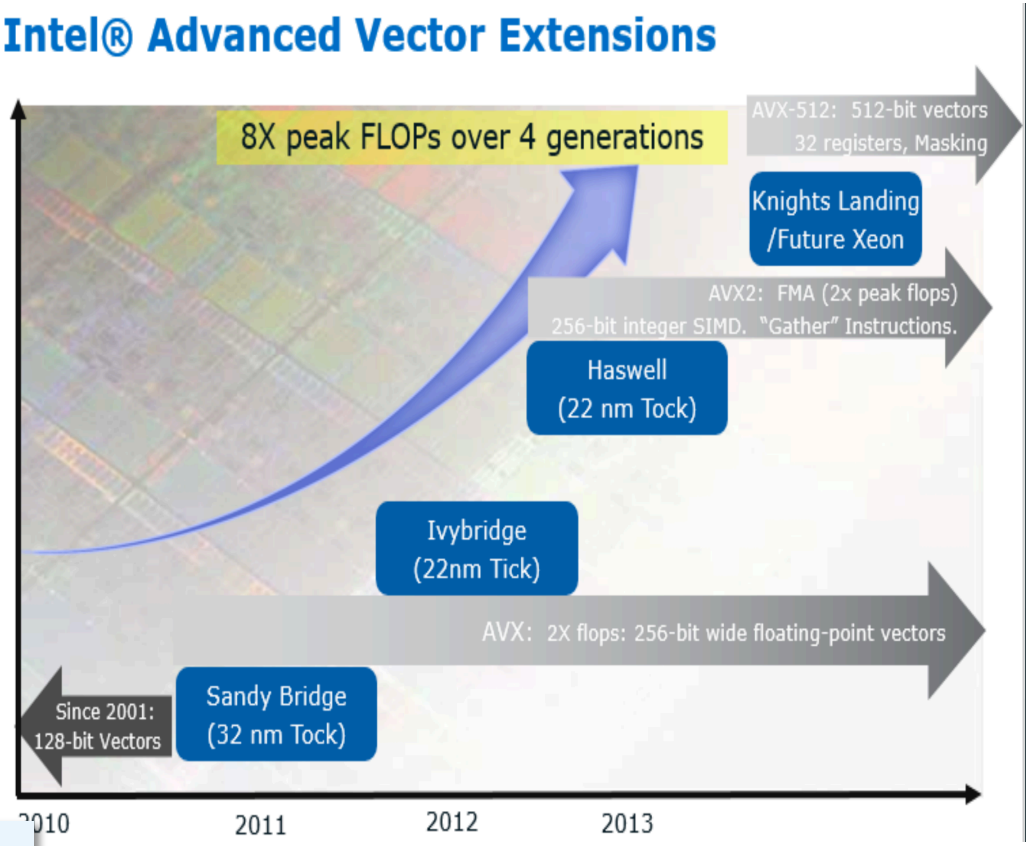
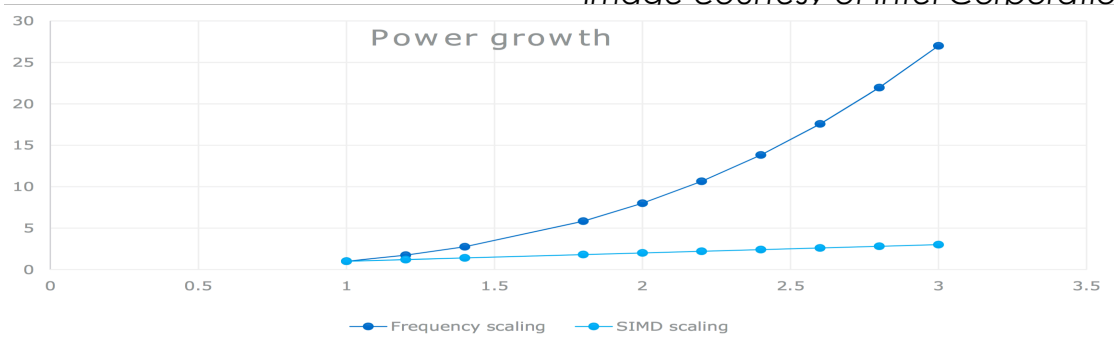


Image courtesy of Intel Corporation

8x FLOPS over last 4 generations due to SIMD!

Why is SIMD likely to be a trend?

Image courtesy of Intel Corporation

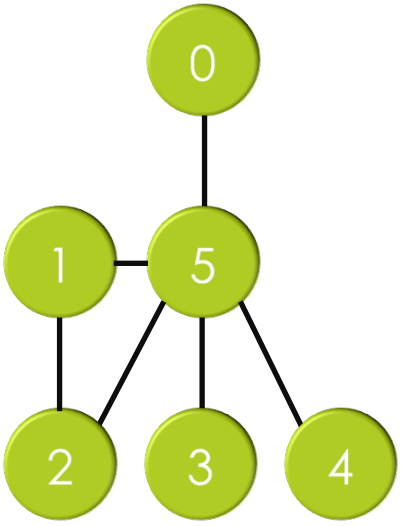


Wider SIMD = **Linear** increase in area and power
Wider superscalar = **Quadratic** increase in a&p
Higher Frequency = **Cubic** increase in a&p

SIMD performance is linear in # transistors, and likely to continue with Moore's law!

Representing Graphs

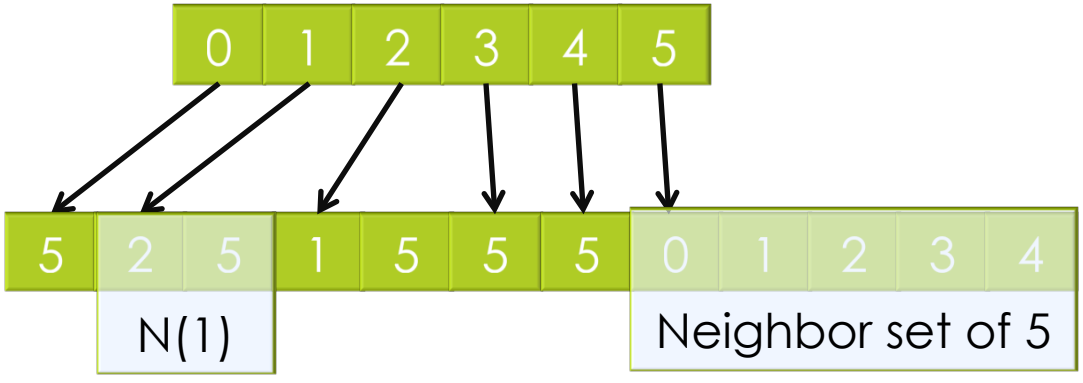
Graph Representation



Dictionary Encoded ID's for each node

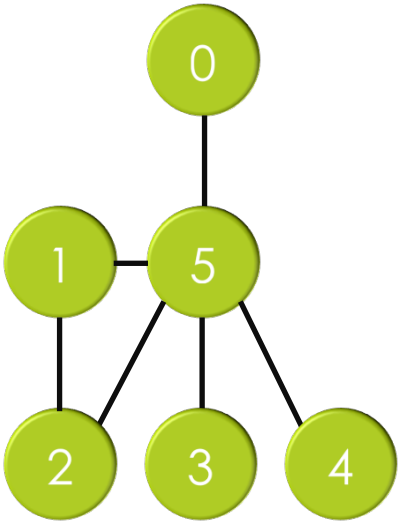
"Compressed Sparse Row"

- Standard in-memory representation for graphs
- Each neighborhood is a set of dictionary encoded id's



Neighbor sets could be Dense, sparse, or in between

Trie Representation

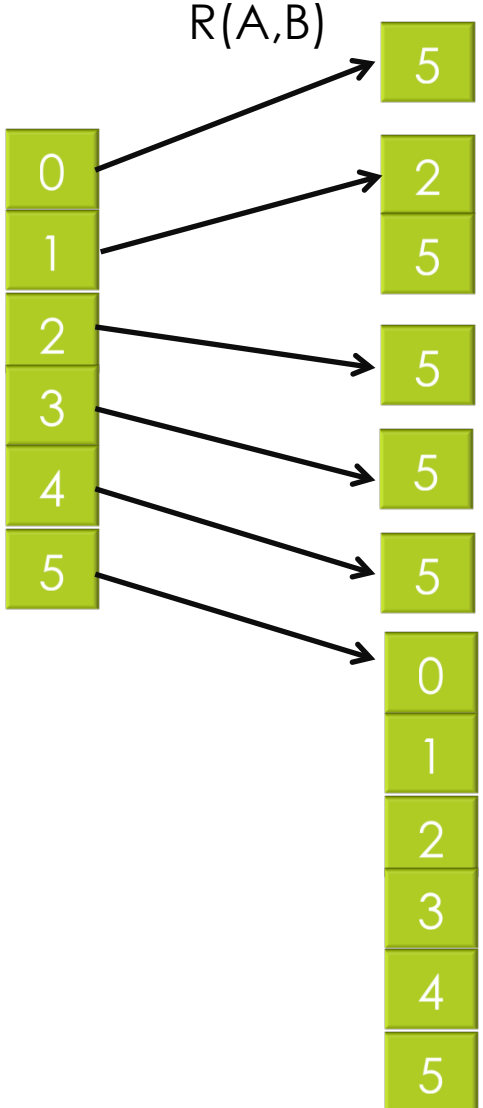


Dictionary Encoded ID's for each node

Table
R(A,B)

A	B
0	5
1	5
1	2
2	5
3	5
4	5
2	5
3	5
4	5

Trie
R(A,B)



EmptyHeaded

Design a Boolean algebra execution engine that exploits SIMD parallelism.

Challenge: cope with **skew** in data.

- *Cardinality Skew*
  - *Skew in the sizes of the sets.*
  - EH selects SIMD algorithms for set intersection
- *Density Skew*
  - *Skew in the density of the sets.*
    - *(subtle property of encoding)*
  - EH selects different representations
- Block Skew
  - Skew within a set!

Skew calculated using
Pearson's first coefficient of skew
3(mean-mode)/stddev

Two types of Set Representation

Full paper contains fancier types…

Unsigned Integer (uint)

- Standard CSR representation
- Works well with sparse data
  - 4 comparisons per cycle using SSE intrinsic
    - Next processor generations even faster!

Bitvector

- Works well with dense data
  - Can provide compression or increase memory usage
  - 256 comparisons per cycle
    - Intersections to use wide AVX AND instructions
    - Next processor generation 2x faster

Unsigned Integer Intersections

Uint Intersection Algorithms

Shuffling: shuffle through arrays SIMD comparing blocks. **No min property**

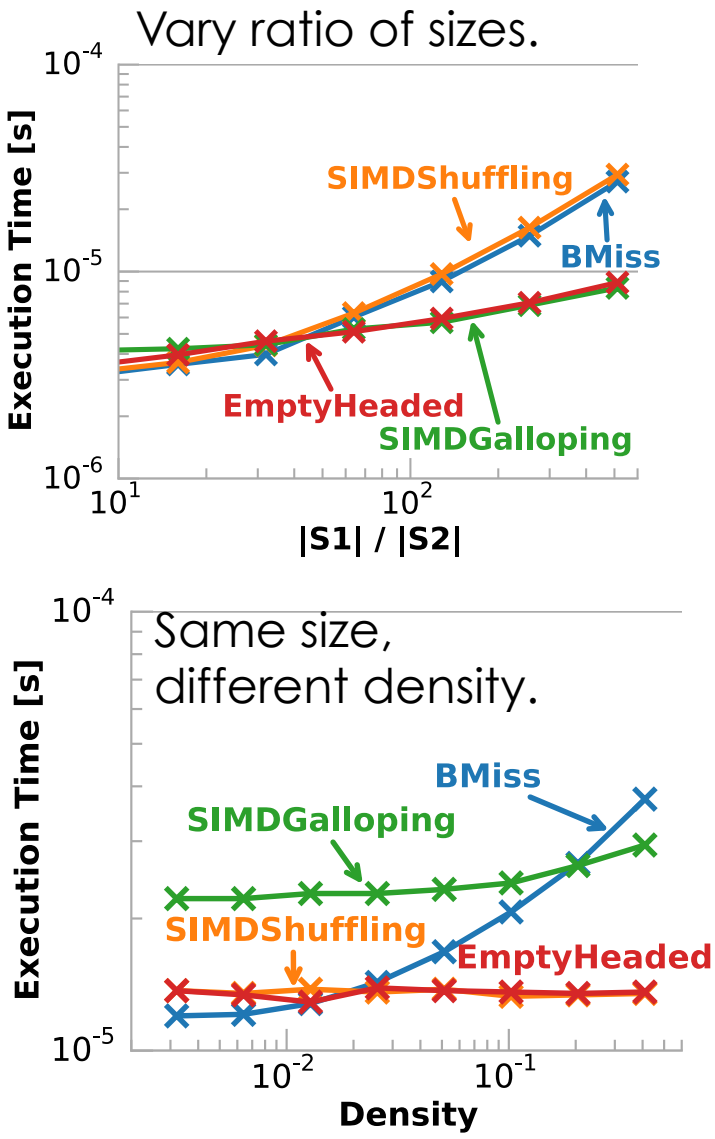
Galloping: gallop over large gaps that do not contribute to the output. ***Has Min property***

Bmiss: filter out unnecessary comparisons with a SIMD check.

To cope with cardinality skew, EH **dynamically** selects galloping when

$\max\{|S1|/|S2|, |S2|/|S1|\} > 32$

Preserves min property.



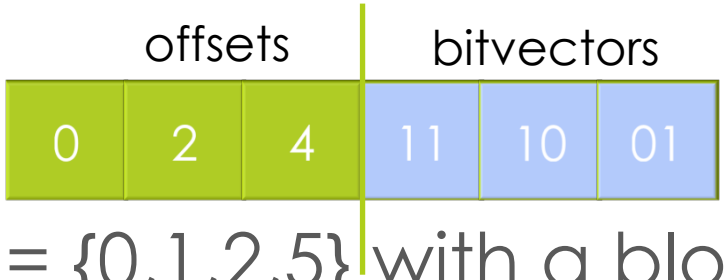
Bitset

EH uses a variant of bitvectors called bitsets.

Stores a set of pairs *(offset,bitvector)*

- Offset is the index of the smallest value in the bitvector

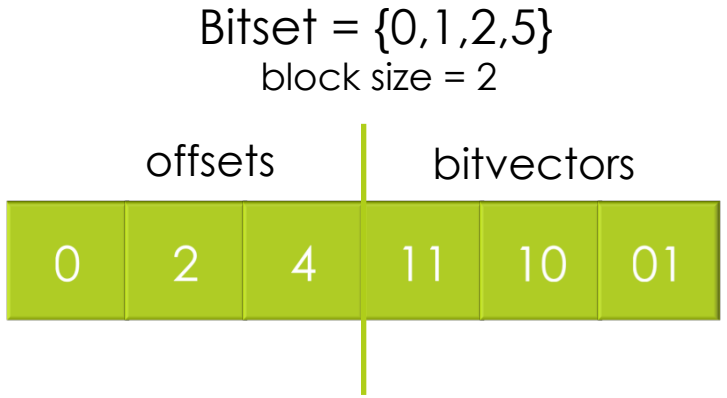
Block size: number of bits in the bitvector

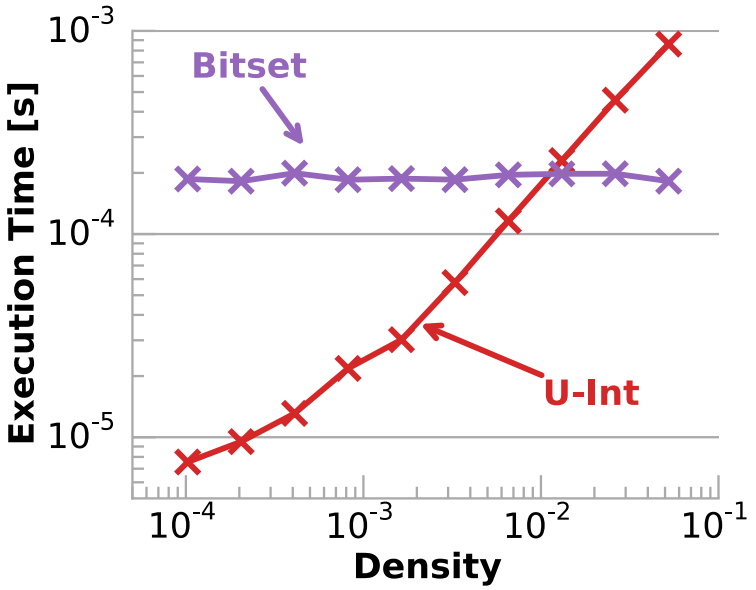


Example: $S = \{0,1,2,5\}$ with a block size of 2.

Bitset Intersections

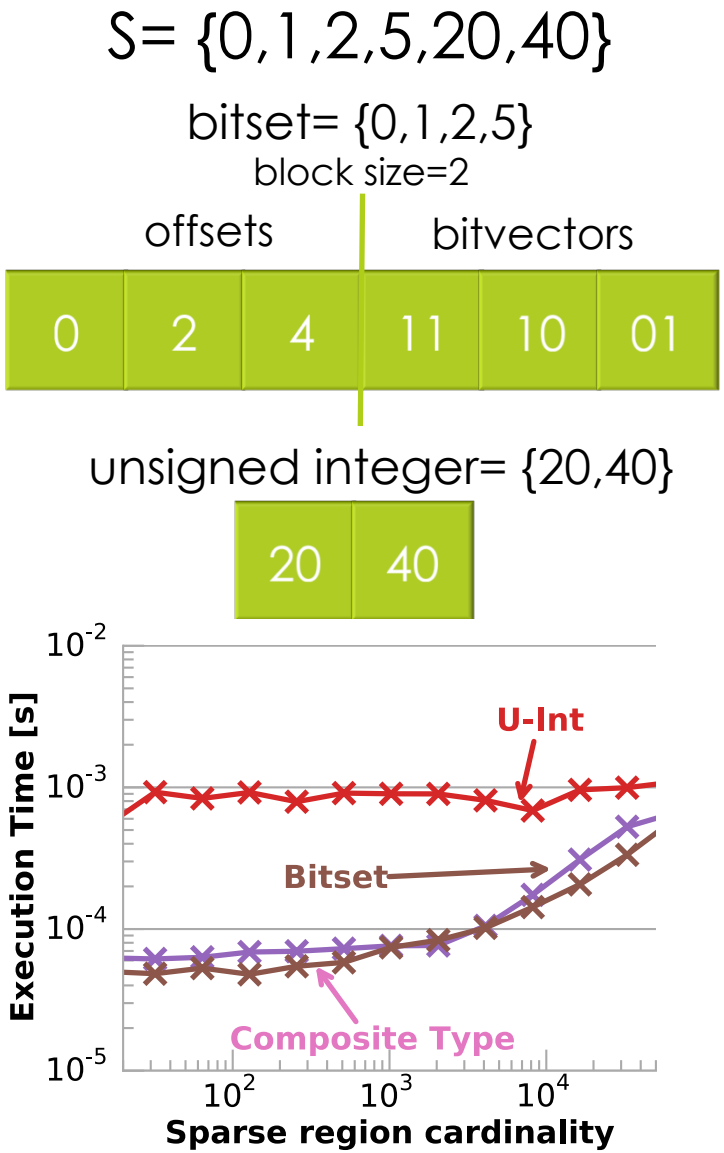
- Use SIMD unsigned integer intersection to intersect offsets.
  - Produces blocks with potential matches
- Use an AVX AND instruction to intersect the bitvectors
  - Can compute the intersection of up to 256 elements in one cycle





Block Level

- Introduce a *composite type* which pushes the decision of representation to a block (within each set) level
  - Each block represented using a bitset or unsigned integer representation
  - Each set contains a bitset with all the bitset blocks and a unsigned integer with the unsigned integer blocks
- Intersection: the cross product of the two representations intersected using previously described techniques, add a merge for the unsigned integer outputs



Representation Decisions

At what granularity do we make the representation decision?

- Graph Level *Edge set Density is Skewed*
  - All sets in graph stored using the same representation
  - No Overhead!
- Set Level *Set Densities are Skewed*
  - Each set stored using a fixed representation
  - Overhead to check set type
- Block Level *Block Densities are Skewed*
  - Each block uses a different representation.
  - Overhead of merging results of blocks within sets

Scene Missing

EMPTYHEADED

See paper for many more experiments and layouts…

High-level Experiments

Triangle Counting

Skew calculated using Pearson's first coefficient of skew
3(mean-mode)/stddev

	Nodes [Million]	Undirected Edges [Millions]	Density Skew	Cardinality Skew
Google+	0.11	12.2	1.17	1.17
Higgs	0.4	12.5	0.23	0.46
LiveJournal	4.8	43.4	0.09	0.97
Orkut	3.1	117.2	0.08	1.46
Patents	3.8	16.5	0.09	2.22
Twitter	41.7	757.8	0.12	0.07

Recall: *Cardinality Skew* handled by Set Algorithms, *Density Skew* handled by Representations

- Google+ Highly Skewed in density & cardinality.
- Patents no Density Skew, High Cardinality Skew.

Triangle Counting

	-SIMD	-Rep	-Both
Google+	1.0x	3.0x	7.5x
Higgs	1.5x	3.9x	4.8x
LiveJournal	1.6x	1.0x	1.6x
Orkut	1.8x	1.1x	2.0x
Patents	1.3x	0.9x	1.1x

High Skew

Low Skew

SIMD and representation optimizations

Triangle Counting

Runtime in seconds for EmptyHeaded.

Relative performance for other systems.

	EmptyHeaded	PowerGraph	Snap-R	LogicBlox
Google+	0.055s	55x	23.7x	862.7x
Higgs	0.057s	8.9x	15.4x	126.1x
LiveJournal	0.24s	10.8x	21.6x	92.4x
Orkut	1.36s	5.5x	7.1x	46.4x
Patents	0.048s	30.0x	63.8x	88.7x
Twitter	24.67s	11.0x	5.1x	129.2x

48 used threads for each system.

Against highly tuned systems, EH approach is faster.

4-clique and Lollipops!

		EmptyHeaded	-SR	-Y	LogicBlox
Google+	K_4	1.66s	19.1x	-	t/o
	$L_{3,1}$	6.03s	3.5x	653.9x	t/o
Higgs	K_4	0.25s	1.7x	-	230.4x
	$L_{3,1}$	0.89s	19.7x	834.9x	t/o
LiveJournal	K_4	1.04s	1.5x	-	1276.4x
	$L_{3,1}$	1.76s	1.3x	79.9x	t/o
Orkut	K_4	3.40s	1.4x	-	299.9x
	$L_{3,1}$	8.88s	1.5x	216.9x	t/o
Patents	K_4	0.10s	0.8x	-	37.2x
	$L_{3,1}$	0.31s	0.9x	1.8x	52.5x

Similarity Queries

		EmptyHeaded	-SR	LogicBlox
Google+	CN	0.01s	6.1x	851.9x
	Symb	4e-5s	0.9x	6.6e4x
Higgs	CN	0.01s	44.6x	255x
	Symb	0.01s	1.0x	834.7x
LiveJournal	CN	0.03s	39.7x	37.9x
	Symb	0.03s	0.9x	604.0x
Orkut	CN	0.04s	51.1x	141.4x
	Symb	0.03s	0.9x	742.1x
Patents	CN	0.04s	1.6x	10.0x
	Symb	0.01s	0.9x	291.6x

More Complex Queries

- On K4 or Lollipop, only LogicBlox runs between 37x to 1276x+ improvement.
  - Part of this due to the GHD trick I mentioned
    - 80x-850x in the cases when it helps.
    - Up to 20x due to SIMD across datasets.

- Also ran BFS and some other workloads
  - Single threaded performance beats all systems
  - Ligra scales better on BFS
  - "Similarity" workloads also much faster.

Conclusion

Shock: For Joins and Graph Patterns, **Relational Algebra** is suboptimal, but **Boolean Algebra** can be optimal.

Part 1. Theory. Optimality Claim

- Simplified worst-case optimal compilation

Part 2. Hardware. Optimize Boolean Algebra [Arxiv15]

- Modern processors **love** Boolean Algebra
  - SIMD is the new Moore's Law!
- Challenge: Cope with ***Skew***!

EMPTYHEADED

To Infinity and Beyond

The Team

- Adam Perelman + Susan Tu
  - DunceCap Query Compiler
  - REPL
- Rohan Puttagunta
  - Incremental algorithms
  - Worst-case theory and beyond!

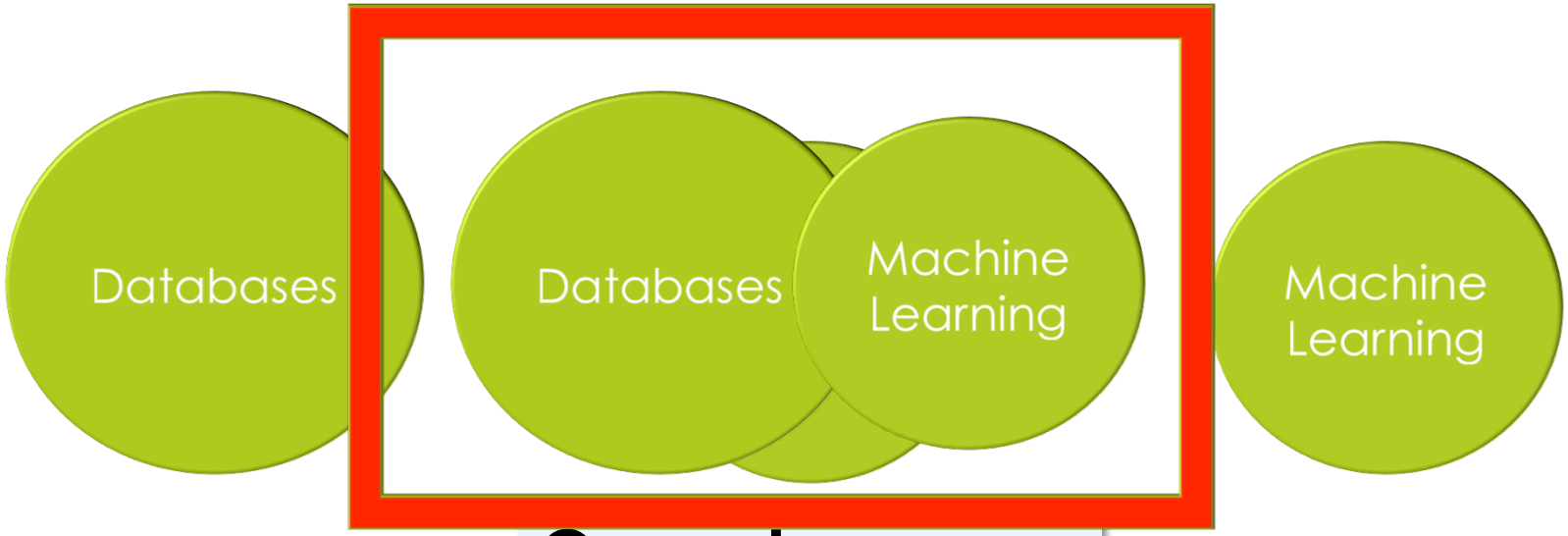
EMPTYHEADED

Future Work

Is everything a join? Taking advantage of pockets of density certainly extends beyond joins.

- Applications -Matrix Multiply, Inference, Fast-Fourier transform
- Distributed computation-"If your system doesn't perform well on one billion things, you don't yet have the credibility to claim it's going to work much better on a trillion things."

One System to Rule Them All?

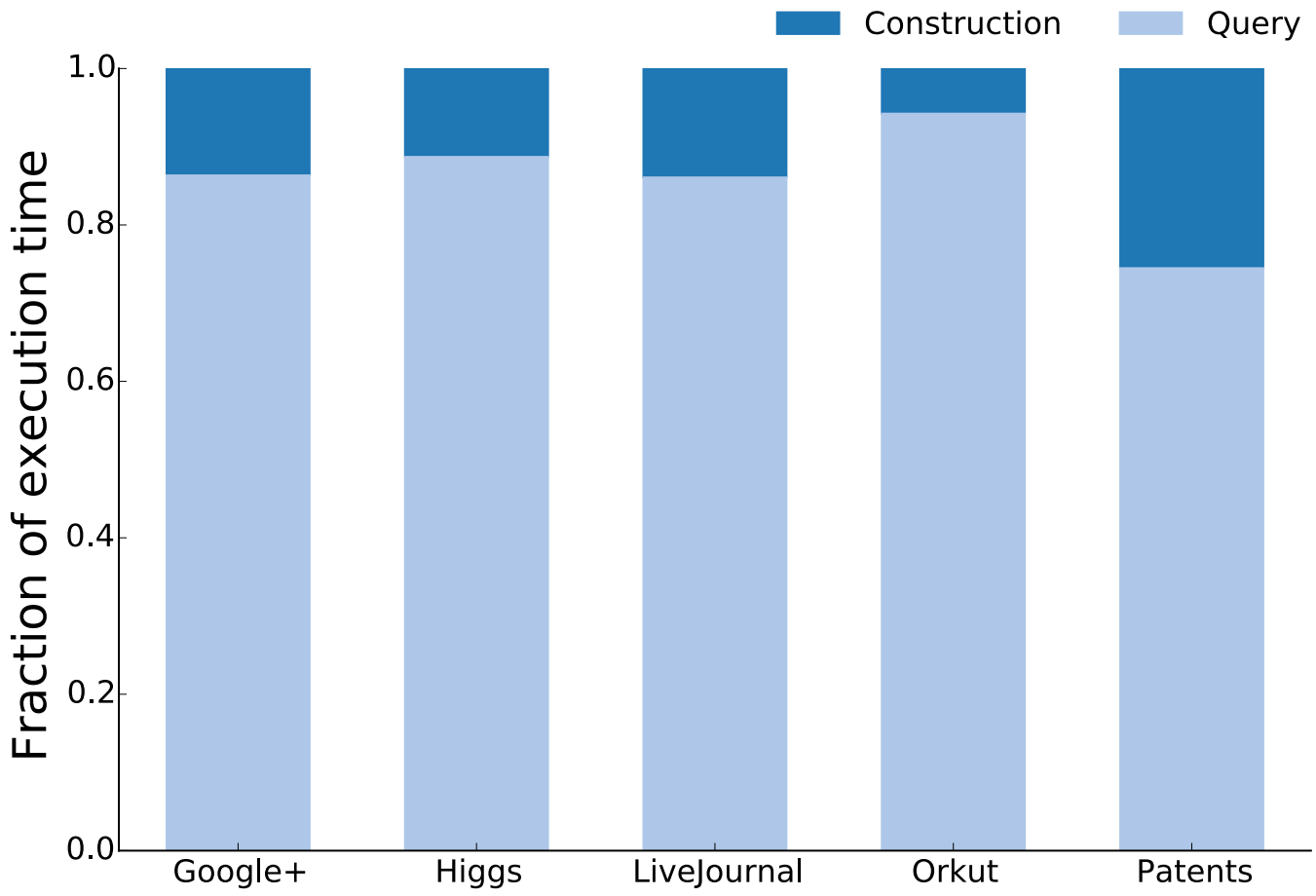


Operators

Map
Reduce
Filter
GroupBy
Intersect
Union
Difference

Lets collaborate!

Triangle Counting



Triangle Counting

	Graph Level	Set Level	Block Level
Google+	7.3x	1.1x	3.2x
Higgs	1.6x	1.4x	2.4x
LiveJournal	1.3x	1.4x	2.0x
Orkut	1.4x	1.4x	2.0x
Patents	1.2x	1.6x	1.9x

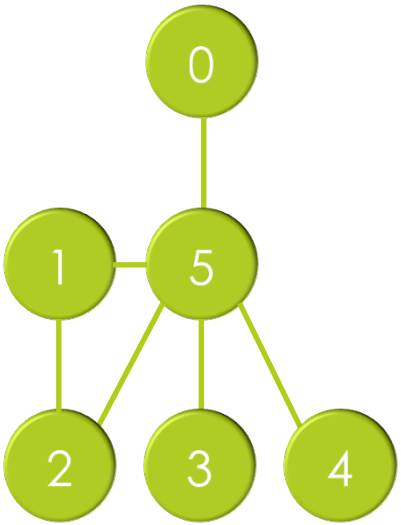
Relative time of the EmptyHeaded optimizers compared to an oracle. **Smaller is better.**

Triangle Counting

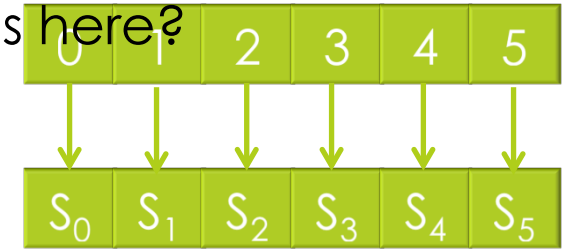
	Set Optimizer	Block Optimizer
Google+	4%	5%
Higgs	1%	6%
LiveJournal	4%	12%
Orkut	3%	8%
Patents	10%	24%

Set and block level optimizer overheads.

Graph Representation

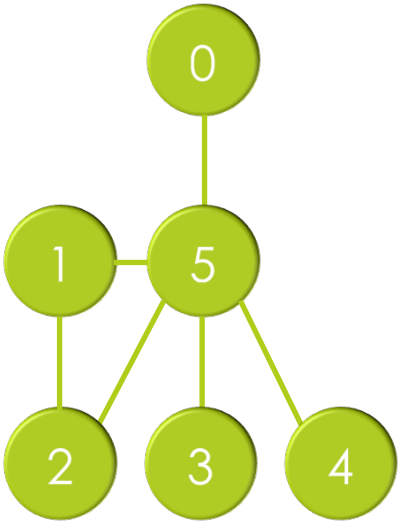


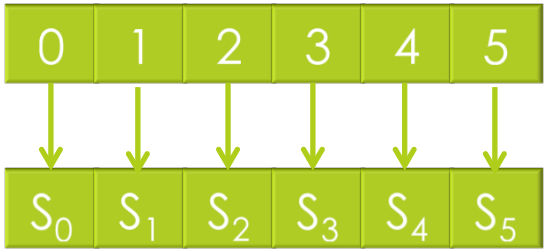
- Logically regard each neighborhood as sets.
- Not fundamentally different from Compressed Sparse Row but now an abstraction is in place to choose representations at will.
  - S_0 is sparse…storing the 32-bit ID probably works OK
  - S_5 is dense…maybe we can do better than storing the 32-bit ID's here?



Graph Level

- Whole graph encoded using unsigned integer or bitset representations





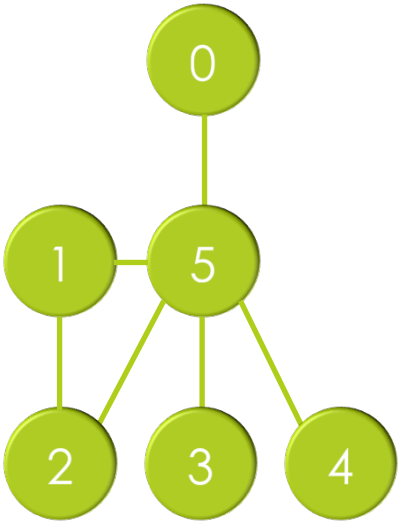
$\{S_0,S_1,S_2,S_3,S_4,S_5\}$ = unsigned integers (CSR)

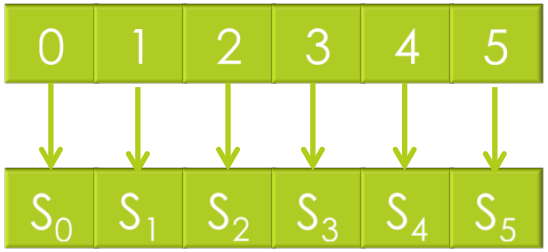
or

$\{S_0,S_1,S_2,S_3,S_4,S_5\}$ = bitsets

Set Level

- Each set encoded using unsigned integer or bitset representations





$\{S_0,S_2,S_3,S_4\}$ = unsigned integers (CSR)

$\{S_1,S_5\}$ = bitsets

Heterogeneous Intersections

- Shift the unsigned integer values right by $log_2(block\ size)$
  - Restrict ourselves to block sizes which are powers of 2
- Intersect offsets with shifted values, keeping track of the positions in both sets that match
  - e.g. yields {0,2,4}
- Probe the bitvector blocks yielded from the intersection with the corresponding unsigned integer values that matched
  - e.g. probe blocks with offsets of {0,2,4} with values of {1,3,5}
- Output result in a unsigned integer representation
  - e.g. result = {1,5}