It's All a Matter of Degree: Using Degree Information to Optimize Multiway Joins

Manas R. Joglekar[1] and Christopher M. Ré[2]

1 Department of Computer Science, Stanford University, Stanford, CA, USA
manasrj@stanford.edu

2 Department of Computer Science, Stanford University, Stanford, CA, USA
chrismre@stanford.edu

Abstract

We optimize multiway equijoins on relational tables using degree information. We give a new bound that uses degree information to more tightly bound the maximum output size of a query. On real data, our bound on the number of triangles in a social network can be up to 95 times tighter than existing worst case bounds. We show that using only a constant amount of degree information, we are able to obtain join algorithms with a running time that has a smaller exponent than existing algorithms – *for any database instance*. We also show that this degree information can be obtained in nearly linear time, which yields asymptotically faster algorithms in the serial setting and lower communication algorithms in the MapReduce setting.

In the serial setting, the data complexity of join processing can be expressed as a function $O(\mathrm{IN}^x + \mathrm{OUT})$ in terms of input size IN and output size OUT in which x depends on the query. An upper bound for x is given by fractional hypertreewidth. We are interested in situations in which we can get algorithms for which x is strictly smaller than the fractional hypertreewidth. We say that a join can be processed in subquadratic time if $x < 2$. Building on the AYZ algorithm for processing cycle joins in quadratic time, for a restricted class of joins which we call 1-series-parallel graphs, we obtain a complete decision procedure for identifying subquadratic solvability (subject to the 3-SUM problem requiring quadratic time). Our 3-SUM based quadratic lower bound is tight, making it the only known tight bound for joins that does not require any assumption about the matrix multiplication exponent ω. We also give a MapReduce algorithm that meets our improved communication bound and handles essentially optimal parallelism.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Joins, Degree, MapReduce

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.11

1 Introduction

We study query evaluation for natural join queries. Traditional database systems process joins in a pairwise fashion (two tables at a time), but recently a new breed of multiway join algorithms have been developed that satisfy stronger runtime guarantees. In the sequential setting, worst-case-optimal sequential algorithms such as NPRR [16, 17] or LFTJ [18] process the join in runtime that is upper bounded by the largest possible output size, a stronger guarantee than what traditional optimizers provide. In MapReduce settings[1], the Shares algorithm [2, 13] processes multiway joins with optimal communication complexity on skew

[1] A description of Background material including MapReduce, as well as proofs of all our results, can be found in the full version of the paper [12]

Table 1 Triangle bounds on various social networks.

Network	MO Bound	AGM Bound	$\frac{\text{AGM}}{\text{MO}}$
Twitter	$225M$	$3764M$	17
Epinions	$33M$	$362M$	11
LiveJournal	$6128M$	$573062M$	95

free data. However, traditional database systems have developed sophisticated techniques to improve query performance. One popular technique used by commercial database systems is to collect "statistics": auxiliary information about data, such as relation sizes, histograms, and counts of distinct different attribute values. Using this information helps the system better estimate the size of a join's output and the runtimes of different query plans, and make better choices of plans. Motivated by the use of statistics in query processing, we consider how statistics can improve the new breed of multiway join algorithms in sequential and parallel settings.

We consider the first natural choice for such statistics about the data: the degree. The degree of a value in a table is the number of rows in which that value occurs in that table. We describe a simple preprocessing technique to facilitate the use of degree information, and demonstrate its value through three applications: i) An improved output size bound ii) An improved sequential join algorithm iii) An improved MapReduce join algorithm. Each of these applications has an improved exponent relative to their corresponding state-of-the-art versions [5, 8, 16, 18].

Our key technique is what we call *degree-uniformization*. Assume for the moment that we know the degree of each value in each relation, we then partition each relation by degree of each of its attributes. In particular, we assign each degree to a bucket using a parameter L: we create one bucket for degrees in $[1, L)$, one for degrees in $[L, L^2)$, and so on. We then place each tuple in every relation into a partition based on the degree buckets for each of its attribute values. The join problem then naturally splits into smaller join problems; each smaller problem consisting of a join using one partition from each relation. Let IN denote the input size, if we set $L = \text{IN}^c$ for some constant c, say $\frac{1}{4}$, the number of smaller joins we process will be exponential in the number of relations – but constant with respect to the data size IN. Intuitively, the benefit of joining partitions separately is that each partition will have more information about the input and will have reduced skew. We show that by setting L appropriately this scheme allows us to get tighter AGM-like bounds.

Now we consider a concrete example. Suppose we have a d-regular graph with N edges; the number of triangles in the graph is bounded by $\min(Nd, \frac{N^2}{d})$ by our degree-based bound and by $N^{3/2}$ by the AGM bound. In the worst case, $d = \sqrt{N}$ and our bound matches the AGM bound. But for other degrees, we do much better; better even than simply "summing" the AGM bounds over each combination of partitions. Table 1 compares our bound (MO) with the AGM bound for the triangle join on social networks from the SNAP datasets [14]. 'M' in the table stands for millions. The last column shows the ratio of the AGM bound to our bound; our bound is tighter by a factor of $11x$ to $95x$. We could not compare the bounds on the Facebook network, but if the number of friends per user is ≤ 5000, our bound is at least $450x$ tighter than the AGM bound.

We further use degree uniformization as a tool to develop algorithms that satisfy stronger runtime and communication guarantees. Degree uniformization allows us to get runtimes with a better exponent than existing algorithms, while requiring only linear time preprocessing on the data. We demonstrate our idea in both the serial and parallel (MapReduce) setting, and we now describe each in turn.

Serial Join Algorithms: We use our degree-uniformization to derive new cases in which one can obtain subquadratic algorithms for join processing. More precisely, let IN denote the size of the input, and OUT denote the size of the output. Then the runtime of an algorithm on a query Q can be written as $O(\text{IN}^x + \text{OUT})$ for some x. Note that $x \geq 1$ for all algorithms and queries in this model as we must read the input to answer the query. If the query is α-acyclic, Yannakakis' algorithm [19] achieves $x = 1$. If the query has fractional hypertree width (fhw), a recent generalization of tree width [10], equal to 2, then we can achieve $x = 2$ using a combination of algorithms like NPRR and LFTJ with Yannakakis' algorithm. In this work, we focus on cases for which $x < 2$, which we call *subquadratic algorithms*. Subquadratic algorithms are interesting creatures in their own right, but they may provide tools to attack the common case in join processing in which OUT is smaller than IN.

Our work builds on the classical AYZ algorithm [4], which derives subquadratic algorithms for cycles using degree information. This is a better result than the one achieved by the fhw result since the fhw value of length ≥ 4 cycles is already $= 2$. This result is specific to cycles, raising the question: *"Which joins are solvable in subquadratic time?"* Technically, the AYZ algorithm makes use of properties of cycles in their result and of "heavy and light" nodes (high degree and low degree, respectively). We show that degree-uniformization is a generalization of this method, and that it allows us to derive subquadratic algorithms for a larger family of joins. We devise a procedure to upper bound the processing time of a join, and an algorithm to match this upper bound. Our procedure improves the runtime exponent x relative to existing work, for a large family of joins. Moreover, for a class of graphs that we call 1-series-parallel graphs,[2] we completely resolve the subquadratic question in the following sense: For each 1-series-parallel graph, we can either solve it in subquadratic time, or we show that it cannot be solved subquadratically unless the 3-SUM problem [6] can be solved in subquadratic time. Note that 1-series-parallel graphs have fhw equal to 2. Hence, they can all be solved in quadratic time using existing algorithms; making our 3-SUM based lower bound tight. There is a known 3-SUM based lower bound of $N^{\frac{4}{3}}$ on triangle join processing, which only has a matching upper bound under the assumption that the matrix multiplication exponent $\omega = 2$. In contrast, our quadratic lower bound can be matched by existing algorithms without any assumptions on ω. To our knowledge, this makes it the only known tight bound on join processing time for small output sizes.

We also recover our sequential join results within the well-known GHD framework [10]. We do this using a novel notion of width, which we call m-width, that is no larger than fhw, and sometimes smaller than submodular width [12, 15]. While we resolve the subquadratic problem on 1-series-parallel graphs, the general subquadratic problem remains open. In the full version [12], we show that known notions of widths, such as submodular width and m-width do not fully characterize subquadratically solvable joins.

Joins on MapReduce: Degree information can also be used to improve the efficiency of joins on MapReduce. Previous work by Beame et al. [8] uses knowledge of heavy hitters (values with high degree) to improve parallel join processing on skewed data. It allows a limited range of parallelism (number of processors $p \leq \sqrt{\text{IN}}$), but subject to that achieves optimal communication for 1-round MapReduce algorithms. We use degree information to allow all levels of parallelism ($p \geq 1$) while processing the join. We also obtain an improved degree-based upper bound on output size that can be significantly better than the

[2] A 1-series-parallel graph consists of a source vertex s, a target vertex t, and a set of paths of any length from s to t, which do not share any nodes other than s and t.

AGM bound even on simple queries. Our improved parallel algorithm takes three rounds of MapReduce, matches our improved bound, and out-performs the optimal 1-round algorithm in several cases. As an example, our improved bound lets us correctly upper bound the output of a sparse triangle join (where each value has degree $O(1)$) by IN instead of $\text{IN}^{\frac{3}{2}}$ as suggested by the AGM bound. Moreover, we can process the join at maximum levels of parallelism (with each processor handling only $O(1)$ tuples) at a total communication cost of $O(\text{IN})$; in contrast to previous work which requires $\theta(\text{IN}^{\frac{3}{2}})$ communication. Furthermore, previous work [8] uses edge packings to bound the communication cost of processing a join. Edge packings have the paradoxical property that adding information on the size of subrelations by adding the subrelations into the join can make the communication cost larger. As an example suppose a join has a relation R, with an attribute A in its schema. Adding $\pi_A(R)$ to the set of relations to be joined does not change the join output. However, adding a weight term for subrelation $\pi_A(R)$ in the edge packing linear program increases its communication cost bound. In contrast, if we add $\pi_A(R)$ into the join, our degree based bound does not increase, and will in fact decrease if $|\pi_A(R)|$ is small enough.

Computing Degree Information: In some cases, degree information is not available beforehand or is out of date. In such a case, we show a simple way to compute the degrees of all values in time linear in the input size. Moreover, the degree computation procedure can be fully parallelized in MapReduce. Even after including the complexity of computing degrees, our algorithms outperform state of the art join algorithms.

Our paper is structured as follows:

- In Section 2, we describe related work.
- In Section 3, we describe a process called *degree-uniformization*, which mitigates skew. We show the MO bound on join output size that strengthens the exponent in the AGM bound, and describe a method to compute the degrees of all attributes in all relations.
- In Section 4, we present DARTS, our sequential algorithm that achieves tighter runtime exponents than state-of-the-art. We use DARTs to process several joins in subquadratic time. Then we establish a quadratic runtime lower bound for a certain class of queries modulo the 3-SUM problem. Finally we recover the results of DARTS within the familiar GHD framework, using a novel notion of width (m-width) that is tighter than fhw.
- In Section 5, we present another bound with a tighter exponent than AGM (the DBP bound), and a tunable parallel algorithm whose communication cost at maximum parallelism equals the input size plus the DBP bound. The algorithm's guarantees work on all inputs independent of skew.

2 Related Work

We divide related work into four broad categories.

New join algorithms and implementation: The AGM bound [5] is tight on the output size of a multiway join in terms of the query structure and sizes of relations in the query. Several existing join algorithms, such as NPRR [16], LFTJ [18], and Generic Join [17], have worst case runtime equal to this bound. However, there exist instances of relations where the output size is significantly smaller than the worst-case output size (given by the AGM bound), and the above algorithms can have a higher cost than the output size. We demonstrate a bound on output size that has a tighter exponent than the AGM bound by taking into account information on degrees of values, and match it with a parallelizable algorithm.

On α-acyclic queries, Yannakakis' algorithm [19] is instance optimal up to a constant multiplicative factor. That is, its cost is $O(\text{IN} + \text{OUT})$ where IN is the input size. For cyclic queries, we can combine Yannakakis' algorithm with the worst-case optimal algorithms like NPRR to get a better performance than that of NPRR alone. This is done using Generalized Hypertree decompositions (GHDS) [9, 10] of the query to answer the query in time $O(\text{IN}^{\mathsf{fhw}} + \text{OUT})$ where fhw is a measure of cyclicity of the query. A query is α-acyclic if and only if its fhw is one. Our work allows us to obtain a tighter runtime exponent than fhw by dealing with values of different degrees separately.

Parallel join algorithms: The Shares [2] algorithm is the optimal one round algorithm for skew free databases, matching the lower bound of Beame et al. [7]. But its communication cost can be much worse than optimal when skew is present. Beame's work [8] deals with skew and is optimal among 1-round algorithms when skew is present. The GYM [1] algorithm shows that allowing $\log(n)$ rounds of MapReduce instead of just one round can significantly reduce cost. Allowing n rounds can reduce it even further. Our work shows that merely going from one to three rounds can by itself significantly improve on existing 1-round algorithms. Our parallel algorithm can be incorporated into Step 1 of GYM as well, thereby reducing its communication cost.

Using Database Statistics: The cycle detection algorithm by Alon, Yuster and Zwick [4] can improve on the fhw bound by using degree information in a sequential setting. Specifically, the fhw of a cycle is two but the AYZ algorithm [4] can process a cycle join in time $O(\text{IN}^{2-\epsilon} + \text{OUT})$ where $\epsilon > 0$ is a function of the cycle length. We generalize this, obtaining subquadratic runtime for a larger family of graphs, and develop a general procedure for upper bounding the cost of a join by dealing with different degree values separately.

Beame et al.'s work [8] also uses degree information for parallel join processing. Specifically, it assumes that all heavy hitters (values with high degree) and their degrees are known beforehand, and processes them separately to get optimal 1-round results. Their work uses edge packings to bound the cost of their algorithm. Edge packings have the counterintuitive property that adding more constraints, or more information on subrelation sizes, can worsen the edge packing cost. This suggests that edge packings alone do not provide the right framework for taking degree information into account. Our work remedies this, and the performance of our algorithm improves when more constraints are added. In addition, Beame et al. [8] assume that $M > p^2$ where M is relation size and p is the number of processors. Thus, their algorithm cannot be maximally parallelized. In contrast, our algorithm can work at all levels of parallelism, ranging from one in which each processor gets only $O(1)$ tuples to one in which a single processor does all the processing.

Degree Uniformization: The partitioning technique of Alon et al. [3] is similar to our *degree-uniformization* technique, but has stronger guarantees at a higher cost. It splits a relation into 'parts' where the maximum degree of any attribute set A in each part P is within a constant factor of the average degree of A in P. In contrast, degree-uniformization lets us upper bound the maximum degree of A in P in absolute terms, but not relative to the average degree of A in P.

Marx's work [15] uses a stronger partitioning technique to fully characterize the fixed-parameter tractability of joins in terms of the *submodular width* of their hypergraphs. Marx achieves degree-uniformity within all small projections of the output, while we only achieve uniform degrees within relations. Marx's preprocessing is expensive; the technique as written in Section 4 of his paper [15] takes time $\Omega(\text{IN}^{2c})$ where c is the submodular width of the join hypergraph. This preprocessing is potentially more expensive than the join processing itself. Our algorithms run in time $O(\text{IN}^{\mathsf{MW}})$ with $\mathsf{MW} < c$ for several joins. Marx did not attempt to minimize this exponent, as his application was concerned with fixed parameter tractability. We were unable to find an easy way to achieve $O(\text{IN}^{c})$ runtime for Marx's technique.

3 Degree Uniformization

We describe our algorithms for degree-uniformization and counting, as well as our improved output size bound. Section 3.1 introduces our notation. Section 3.2 gives a high-level overview of our join algorithms. Then, we describe the degree-uniformization which is a key step in our algorithms. In Section 3.3, we describe the MO bound, an upper bound on join output size that has a tighter exponent than the AGM bound. We provide realistic examples in which the MO bound is much tighter than the AGM bound. Finally, in Section 3.4 we describe a linear time algorithm for computing degrees.

3.1 Preliminaries and Notation

Throughout the paper we consider a multiway join. Let $\mathcal{R}$ be the set of relations in the join and $\mathcal{A}$ be the set of all attributes in those relations' schemas. For any relation R, we let $\mathsf{attr}(R)$ denote the set of attributes in the schema of R. We wish to process the join $\Join_{R\in\mathcal{R}} R$, defined as the set of tuples t such that $\forall R \in \mathcal{R} : \pi_{\mathsf{attr}(R)}(t) \in R$. $|R|$ denotes the number of tuples in relation R. For any set of attributes $A \subseteq \mathcal{A}$, a *value* in attribute set A is defined as a tuple from $\bigcup_{R\in\mathcal{R}:A\subseteq\mathsf{attr}(R)} \pi_A(R)$. For any $A \subseteq \mathsf{attr}(R)$, the *degree* of a value v in A in relation R is given by the number of times v occurs in R i.e. $\mathsf{deg}(v, R, A) = |\{t \in R \mid \pi_A(t) = v\}|$. For all values v of A in R, we must have $\mathsf{deg}(v, R, A) \geq 1$.

In Section 4, we denote a join query with a hypergraph G; the vertices in the graph correspond to attributes and the hyperedges to relations. We use $R(X_1, X_2, \dots, X_k)$ to denote a relation R having schema $(X_1, X_2, \dots, X_k)$. IN denotes the input size i.e. sum of sizes of input relations, while OUT denotes the output size. Our output size bounds, computation costs, and communication costs will be expressed using O notation which hides polylogarithmic factors i.e. $\log^c(\text{IN})$, for some c not dependent on number of tuples IN (but possibly dependent on the number of relations/attributes). All ensuing logarithms in the paper, unless otherwise specified, will be to the base IN.

AGM Bound: Consider the following linear program:

▶ **Linear Program 1.**

$$\text{Minimize} \sum_{R\in\mathcal{R}} w_R \log(|R|) \text{ such that } \forall a \in \mathcal{A} : \sum_{R\in\mathcal{R}:a\in\mathsf{attr}(R)} w_R \geq 1$$

A valid assignment of weights w_R to relation R in the linear program is called a *fractional cover*. If $\rho*$ is the minimum value of the objective function, then the AGM bound on the join output size is given by $\text{IN}^{\rho*}$. In general, for any set of relations $\mathcal{R}$, we use $\mathsf{AGM}(\mathcal{R})$ to denote the AGM bound on $\Join_{R\in\mathcal{R}} R$.

3.2 Degree Uniformization

We describe our high level join procedure in Algorithm 1. In Step 1, we compute the degree of each value in each attribute set A, in each relation R. If the degrees are available beforehand,

```
Algorithm 1: High level join algorithm
  Input: Set of relations $\mathcal{R}$, Bucket range parameter $L$
  Output: $\bowtie_{R\in\mathcal{R}} R$
  1. Compute $\mathsf{deg}(v, R, A)$ for each $R \in \mathcal{R}, A \subseteq \mathsf{attr}(R), v \in \pi_A(R)$
  2. Compute the set of all $L$-degree configurations $\mathcal{C}_L$
  foreach $c \in \mathcal{C}_L$ do
      3.1. Compute partition $R(c)$ of each relation $R$
      3.2. Compute $\mathcal{R}(c) = \{R(c) \mid R \in \mathcal{R}\}$
      4. Compute join $J_c = \bowtie_{R\in\mathcal{R}(c)} R$
  5. return $\bigcup_{c\in\mathcal{C}_L} J_c$
```

due to being maintained by the database, then we can skip this step. We further describe this step in Section 3.4.

Steps $2, 3$ together constitute *degree-uniformization*. In these steps, we partition each relation R by degree. In particular, we assign each value in a relation to a bucket based on its degree: with one bucket for degrees in $[1, L)$, one for degrees in $[L, L^2)$, and so on. Then we process the join using one partition from each relation, for all possible combinations of partitions. Each such combination is referred to as a *degree configuration*. We use c to denote any individual degree configuration, $\mathcal{C}_L$ to denote the set of all degree configurations, $R(c)$ to denote the part of relation R being joined in configuration c, and $\mathcal{R}(c)$ to denote $\{R(c) \mid R \in \mathcal{R}\}$. Step 2 consists of enumerating all degree configurations, and Step 3 consists of finding the partition of each relation corresponding to each degree configuration.

In Step 4, we compute $J_c = \bowtie_{R\in\mathcal{R}(c)} R$ for each degree configuration c. Section 4 describes how to perform Step 4 in a sequential setting, while Section 5 describes it for a MapReduce setting. Step 5 combines the join outputs for each c to get the final output.

Steps 1, 2, 3 and 5 can be performed efficiently in MapReduce as well as sequential settings; thus the cost of Algorithm 1 is determined by Step 4. Step 4 is carried out differently in sequential and MapReduce settings. Its cost in the sequential setting is lower than the cost in a MapReduce setting. Steps 1, 2, and 3 have a cost of $O(\text{IN})$, while Step 5 has cost $O(\text{OUT})$. Since reading the input and output always has a cost of $O(\text{IN} + \text{OUT})$, the only extra costs we incur are in Step 4 when we actually process the join. Costs for Step 4 will be described in Sections 4 and 5.

Degree-uniformization: Now we describe degree-uniformization in detail. We pick a value for a parameter L which we call 'bucket range', and define buckets $B_l = [L^l, L^{l+1})$ for all $l \in \mathbb{N}$. Let $\mathcal{B} = \{B_0, B_1, \ldots, \}$. For any two buckets $B_i, B_j \in \mathcal{B}$, we say $B_i \leq B_j$ iff $i \leq j$. A degree configuration specifies a unique bucket for each relation and set of attributes in that relation. Formally:

▶ **Definition 1.** Given a parameter L, we define a degree configuration c to be a function that maps each pair (R, A) with $R \in \mathcal{R}, A \subseteq \mathsf{attr}(R)$ to a unique bucket in $\mathcal{B}$ denoted $c(R, A)$, such that

$$\forall R, A, A' : A' \subseteq A \subseteq \mathsf{attr}(R) \Rightarrow c(R, A) \leq c(R, A')$$

$$\forall R : c(R, \mathsf{attr}(R)) = B_0 \text{ and } c(R, \emptyset) = B_{\lfloor \log_L(|R|) \rfloor}$$

▶ **Example 2.** If a join has relations $R_1(X, Y), R_2(Y)$, then a possible configuration is $(R_1, \emptyset) \mapsto B_3$, $(R_1, \{X\}) \mapsto B_1$, $(R_1, \{Y\}) \mapsto B_2$, $(R_1, \{X, Y\}) \mapsto B_0$, $(R_2, \emptyset) \mapsto B_1$, $(R_2, \{Y\}) \mapsto B_0$.

▶ **Definition 3.** Given a degree configuration c for a given L, and a relation $R \in \mathcal{R}$, we define $R(c)$ to be the set of tuples in R that have degrees consistent with c. Specifically:

$$R(c) = \{t \in R \mid \forall A \subseteq \mathsf{attr}(R) : \mathsf{deg}(\pi_A(t), R, A) \in c(R, A)\} \ .$$

We define $\mathcal{C}_L$ to be the set of all degree configurations with parameter L.

▶ **Example 4.** For a tuple $(a, b) \in R$, where $L^2 \leq |R| < L^3$, with the degree of a in B_1, and that of b in B_2, the tuple would be in $R(c)$ if $c(R, \emptyset) = B_2, c(R, \{A\}) = B_1, c(R, \{B\}) = B_2, c(R, \{A, B\}) = B_0$. On the other hand, it would not be in $R(c)$ if $c(R, \{A\}) = B_0$, even if we had $c(R, \{A, B\}) = B_0$, $c(R, \{B\}) = B_2$.

A degree configuration also bounds degrees of values in sub-relations, as stated below:

▶ **Lemma 5.** *For all $R \in \mathcal{R}, A' \subseteq A \subset \mathsf{attr}(R), L > 1, c \in \mathcal{C}_L, v \in \pi_{A'}(R), j \geq i \geq 0$:*

$$c(R, A) = B_i \wedge c(R, A') = B_j \Rightarrow \mathsf{deg}(v, \pi_A(R(c)), A') \leq L^{j+1-i} \ .$$

Choosing L: The optimal value of parameter L depends on our application. L has three effects : (i) For the DBP/MO bounds (Sections 3.3, 5) and sequential algorithm (Section 4), the error in output size estimates is exponential in L (with the exponent depending only on the number of attributes) (ii) The load per processor for the parallel algorithm (Section 5) is $O(L)$ (iii) the number of rounds for the parallel algorithm is $\log_L(\text{IN})$. As a result, we choose a small $L(= 2)$ for the sequential algorithm and DBP/MO bounds, and a larger L ($=$ load capacity $= \text{IN}^\gamma$ for some $\gamma < 1$) for the parallel algorithm.

3.3 Beyond AGM: The MO Bound

We now use degree-uniformization to tighten our upper bound on join output size.

▶ **Definition 6.** Let $\mathcal{R}$ be a set of relations, with attributes in $\mathcal{A}$. For each $R \in \mathcal{R}, A \subseteq \mathsf{attr}(R)$, let $d_{R,A} = \max_{v \in \pi_A(R)} \mathsf{deg}(v, R, A)$. If $A = \emptyset$ then $d_{R,\emptyset} = |R|$. And for any $A \subseteq B \subseteq \mathsf{attr}(R)$, let $d(A, B, R)$ denote $\log(d_{\pi_B(R),A})$. Then consider the following linear program for L.

▶ **Linear Program 2.**

$$\text{Maximize } s_{\mathcal{A}} \text{ s. t. } \ \text{(i) } s_\emptyset = 0 \ \text{(ii) } \forall A, B \text{ s.t. } A \subseteq B : s_A \leq s_B$$
$$\text{(iii) } \forall A, B, E, R \text{ s.t. } R \in \mathcal{R}, E \subseteq \mathcal{A}, A \subseteq B \subseteq \mathsf{attr}(R) : s_{B \cup E} \leq s_{A \cup E} + d(A, B, R)$$

We define $m_{\mathcal{A}}$ to be the maximum objective value of the above program.

▶ **Proposition 7.** The output size $\bowtie_{R \in \mathcal{R}} R$ is in $O(\text{IN}^{m_{\mathcal{A}}})$.

Intuitively, for any $A \subseteq \mathcal{A}$, s_A stands for possible values of $\log(|\pi_A(\bowtie_{R \in \mathcal{R}} R)|)$. This explains the first two constraints (projecting onto the empty set gives size 1, and the projection size over A is monotone in A). For the third constraint, we use the fact that each value in A has at most $\text{IN}^{d(A,B,R)}$ values in B, thus each tuple in $\pi_{A \cup E}(\bowtie_{R \in \mathcal{R}} R)$ can give us at most $\text{IN}^{d(A,B,R)}$ tuples in $\pi_{B \cup E}(\bowtie_{R \in \mathcal{R}} R)$. The linear program attempts to maximize the total output size ($\text{IN}^{s_{\mathcal{A}}}$) while still satisfying the constraints.

We now define the MO bound.

▶ **Definition 8.** Let $\mathsf{MO}(\mathcal{R})$ denote the value $m_{\mathcal{A}}$ for any join query consisting of relations $\mathcal{R}$. Then the MO bound is given by $\sum_{c \in \mathcal{C}_2} \text{IN}^{\mathsf{MO}(\mathcal{R}(c))}$.

▶ **Theorem 9.** *The* MO *bound is in* $O(AGM(\mathcal{R}))$.

The constant in the $O()$ notation depends on the number of attributes in the query, but not on the number of tuples. This result is proved in two steps. Theorem 26 states that the DBP bound (introduced in Section 5) is smaller than the AGM bound, while Theorem 23 implies that the MO bound is smaller than the DBP bound times a constant.

▶ **Example 10.** Let $L = 2$ for this example. Consider a triangle join $R(X, Y) \bowtie S(Y, Z) \bowtie T(Z, X)$. Let $|R| = |S| = |T| = N$. The AGM bound on this is $N^{3/2}$. Let the degree of each value x in X in both R and T be h. For different values of h we will find an upper bound on $m_{\{X,Y,Z\}}$ and hence on the output size.

Case 1. $h < \sqrt{N}$**:** Then $s_{\{X\}} \leq s_\emptyset + d(\emptyset, \{X\}, R) = \log(N/h)$. Thus, $s_{\{X,Y\}} \leq s_{\{X\}} + d(\{X\}, \{X, Y\}, R) \leq \log(N/h) + \log(h) = \log(N)$. Finally, $s_{\{X,Y,Z\}} \leq s_{\{X,Y\}} + d(\{X\}, \{X, Z\}, T) \leq \log(N) + \log(h)$. Thus the MO bound is $\leq Nh < N^{3/2}$.

Case 2. $h > \sqrt{N}$**:** Since there can be at most N/h distinct X values, we have $d(\{Y\}, \{X, Y\}, R) \leq \log(N/h))$. More if the degree of Y in S in a degree configuration is g, then $s_{\{Y,Z\}} \leq s_{\{Y\}} + d(\{Y\}, \{Y, Z\}, S) \leq \log(N/g) + \log(g) = \log(N)$. Finally, $s_{\{X,Y,Z\}} \leq s_{\{Y,Z\}} + d(\{Y\}, \{X, Y\}, R) \leq \log(N) + \log(N/h) = \log(N^2/h) < N^{3/2}$.

The MO bound has a strictly smaller exponent than AGM unless $h \approx \sqrt{N}$. Computing the AGM bound individually over each degree configuration does not help us do better, as the above example can have all tuples in a single degree configuration.

▶ **Example 11.** Consider a matching database [7], where each attribute has the same domain of size N, and each relation is a matching. Thus each value has degree 1, and $d(A, B, R)$ equals 0 when $A \neq \emptyset$ and 1 if $A = \emptyset$. The MO bound on such a database trivially equals N, which can have an unboundedly smaller exponent than the AGM bound.

The full version similarly compares the DBP and AGM bounds, showing that DBP (and hence MO) has a strictly smaller exponent than AGM for 'almost all' degrees.

3.4 Degree Computation

If we do not know degrees in advance we can compute them on the fly, as stated below:

▶ **Lemma 12.** *Given a relation* R*,* $A \subseteq \mathsf{attr}(R)$*, and* $L > 1$*, we can find* $\mathsf{deg}(v, R, A)$ *for each* $v \in \pi_A(R)$ *in a MapReduce setting, with* $O(|R|)$ *total communication, in* $O(\log_L(|R|))$ *MapReduce rounds, and at* $O(L)$ *load per processor. In a sequential setting, we can compute degrees in time* $O(|R|)$.

To perform degree-uniformization, we compute degrees for all relations R, and all $A \subseteq \mathsf{attr}(R)$. The number of such (R, A) pairs is exponential in the number and size of relations, but is still constant with respect to the input size IN.

4 Sequential Join Processing

We present our results on sequential join processing. Section 4.1 describes our problem setting. In Section 4.2 we present our sequential join algorithm, *DARTS* (for **D**egree-based **A**ttribute-**R**elation **T**ransform**s**). DARTS handles queries consisting of a join followed by a projection. A join alone is simply a join followed by projection onto all attributes. We pre-process the input by performing degree-uniformization, and then run DARTS on each degree configuration. DARTS works by performing a sequence of *transforms* on the join problem; each transform reduces the problem to smaller problems with fewer attributes or relations. We describe each of the transforms in turn. We then show that DARTS can be used to recover (while potentially improving on) known join results such as those of the NPRR algorithm, Yannakakis' algorithm, the fhw algorithm, and the AYZ algorithm.

In Section 4.3, we apply DARTS to the subquadratic joins problem; presenting cases in which we can go beyond existing results in terms of the runtime exponent. For a family of joins called 1-series-parallel graphs, we obtain a full dichotomy for the subquadratic joins problem. That is, for each 1-series-parallel graph, we can either show that DARTS processes its join in subquadratic time, or that no algorithm can process it in subquadratic time modulo the 3-SUM problem. Note that 1-series-parallel graphs have treewidth 2, making them easily solvable in quadratic time. Thus, our 3-SUM based quadratic lower bound on some of the graphs is tight making it, to our knowledge, the only tight bound for join processing time with small output sizes. In contrast, there is a $N^{\frac{4}{3}}$ lower bound (using 3-SUM) for triangle joins, but its matching upper bound depends on the additional assumption that the matrix multiplication exponent equals two.

In Section 4.4, we show that most results of the DARTS algorithms can be recovered using the well known framework of Generalized Hypertree Decompositions (GHDs), along with a novel notion of width we call m-width. m-width is no larger than fhw, and sometimes smaller than submodular width [12].

4.1 Setting

In this section, we focus on a sequential join processing setting. We are especially interested in the subquadratic joins problem stated below:

▶ **Problem 1.** For any graph G, we let each node in the graph represent an attribute and each edge represent a relation of size N. Then we want to know, for what graphs G can we process a join over the relations in *subquadratic* time, i.e. $O(N^{2-\epsilon} + \mathrm{OUT})$ for some $\epsilon > 0$?

Performing a join in subquadratic time is especially important when we have large datasets being joined, and the output size is significantly smaller than the worst case output size. Note that we define subquadratic to be a poly(N) factor smaller than N^2, so for instance a $\frac{N^2}{\log N}$ algorithm is not subquadratic by our definition.

As an example, if a join query is α-acyclic, then Yannakakis' algorithm can answer it in time $O(N + \mathrm{OUT})$, which is subquadratic. More generally, if the fractional hypertree width (fhw) of a query is $\rho*$, the join can be processed in time $O(N^{\rho*} + \mathrm{OUT})$ using a combination of the NPRR and Yannakakis' algorithms. The fhw of an α-acyclic query is one. For any graph with $\mathsf{fhw} < 2$, we can process its join in subquadratic time. The AYZ algorithm allows us to process joins over length n cycles in time $O(N^{2-\frac{1}{1+\lceil \frac{n}{2} \rceil}} + \mathrm{OUT})$, even though cycles of length ≥ 4 have $\mathsf{fhw} = 2$. To the best of our knowledge, this is the only previous result that can process a join with fhw ≥ 2 in subquadratic time.

The DARTS algorithm is applicable to any join-project problem and not just those with equal relation sizes like in Problem 1. Applying DARTS to Problem 1 lets us process several joins in subquadratic time despite having fhw ≥ 2. Section 4.4 recovers the subquadratic runtimes of DARTS using GHDs that have m-width < 2.

4.2 The DARTS algorithm

We now describe the DARTS algorithm. The problem that DARTS solves is more general than a join. It takes as input a set of relations $\mathcal{R}$, and a set of attributes $\mathcal{O}$ (which stands for **O**utput), and computes $\pi_{\mathcal{O}} \bowtie_{R\in\mathcal{R}} R$. When $\mathcal{O} = \mathcal{A}$, the problem reduces to just a join. We first pre-process the inputs by performing degree-uniformization. Then each degree configuration is processed separately by DARTS. The L parameter for degree-uniformization is set to be very small ($O(1)$). The total computation time is the sum of the computation times over all degree configurations. Let $G = (c, \mathcal{R}(c), \mathcal{O})$. That is, G specifies the query relations, output attributes, and degrees for each attribute set in each relation according to the degree configuration. We let $c_G, \mathcal{R}_G, \mathcal{O}_G$ denote to degree configuration of G, the relations in G, and the output attributes of G. We define two notions of runtime complexity for the join-project problem on G:

▶ **Definition 13.** $Q(G)$ is the smallest value such that a join-projection with query structure, degrees, and output attributes given by those in G can be processed in time $O(Q(G)+\mathrm{OUT})$. $P(G)$ is the smallest value such that a join-projection with query structure, degrees, and output attributes given by those in G can be processed in time $O(P(G))$.

▶ **Example 14.** As an example of the difference between P and Q, consider a chain join G with relations $R_1(X_1, X_2)$, $R_2(X_2, X_3)$, $R_3(X_3, X_4)$, and $\mathcal{O} = \{X_1, X_2, X_3, X_4\}$. All relations have size N, and the degree of each attribute in each relation is $\sqrt{N}$. Then $P(G)$ would be N^2, the worst case size of the output (where all attributes have $\sqrt{N}$ values and each relation is a full cartesian product). $Q(G)$ on the other hand would be N because the join is α-acyclic, and Yannakakis' algorithm lets us process the join in time $O(N + \mathrm{OUT})$.

4.2.1 Heavy, Light and Split

The DARTS algorithm performs a series of *transforms* on G, each of which reduces it to a smaller problem. In each step, it chooses one of three types of transforms, which we call *Heavy*, *Light* and *Split*. Each transform takes as input G itself and either an attribute or a set of attributes in the relations of G. Then it reduces the join-project problem on G to a simpler problem via a *procedure*. This reduction gives us a *bound* on $P(G)$ and/or $Q(G)$ in terms of the P and Q values of simpler problems. We describe each of these transforms in turn, along with their input, procedure, and bound.

Heavy

Input: G, An attribute X

Procedure: Let $\mathcal{R}_X = \{R \in \mathcal{R}(c) \mid X \in \mathsf{attr}(R)\}$. Then we compute the values of $x \in X$ that lie in all relations in $\mathcal{R}_X$ i.e. $\mathsf{vals}(X) = \bigcap_{R\in\mathcal{R}_X} \pi_X R$. Then for each $x \in \mathsf{vals}(X)$, we marginalize on x. That is, we solve the *reduced problem*:

$$J_x = \pi_{\mathcal{O}\setminus\{X\}} \left(\bowtie_{R\in(\mathcal{R}(c)\setminus\mathcal{R}_X)} R \bowtie_{R\in\mathcal{R}_X} (\pi_{\mathcal{A}\setminus\{X\}}\sigma_{X=x}R) \right) .$$

Our final output is $\bigcup_{x\in\mathsf{vals}(X)} (\pi_{\mathcal{O}} x) \times J_x$. For each relation $R \in \mathcal{R}_X$, let d_R be the maximum value in bucket $c(R, \{X\})$. So $|\mathsf{vals}(X)| \le \min_{R\in\mathcal{R}_X} \frac{|R|}{d_R}$. Secondly, in each reduced problem J_x, the size of each reduced relation $\pi_{\mathcal{A}\setminus\{X\}}\sigma_{X=x}R$ for $R \in \mathcal{R}_X$ reduces to at most d_R. Let G' denote the reduced relations, degrees, and output attributes for J_x. This gives us:

Bound: $Q(G) \le \left(\min_{R\in\mathcal{R}_X} \frac{|R|}{d_R}\right) Q(G')$, $P(G) \le \left(\min_{R\in\mathcal{R}_X} \frac{|R|}{d_R}\right) P(G')$

Light

Input: G, An attribute set X

Procedure: The light transform reduces the number of relations in G. Define $\mathcal{R}_X = \{R \in \mathcal{R}(c) \mid \mathsf{attr}(R) \subseteq X\}$. We compute $R_X = \bowtie_{R\in\mathcal{R}(c)} \pi_X R$. This subjoin is computed using a sequential version of the parallel technique in Section 5. Hence it takes time equal to the DBP bound on that join. Then we delete relations in $\mathcal{R}_X$ from G, and add R_X into $\mathcal{R}_G$. The degrees for attributes in R_X can be computed in terms of degrees in the relations from $\mathcal{R}_X$. As long as $|\mathcal{R}_X| > 1$, this gives us a reduced problem G'. $\mathcal{O}$ stays unchanged for the reduced problem. The size of relation R_X can be upper bounded using the DBP bound as well. Let $\mathsf{DBP}(G, X)$ denote this bound.

Bound: $Q(G) \leq \mathsf{DBP}(G,X) + Q(G')$, $P(G) \leq \mathsf{DBP}(G,X) + P(G')$

Split

Input: G, An *articulation set* S of attributes [11] such that there are joins G_1, G_2 whose attribute sets have no attribute outside S in common, and $\mathcal{R}_G \subseteq \mathcal{R}_{G_1} \cup \mathcal{R}_{G_2}$. Also, S satisfies either (i) $S \subseteq \mathcal{O}$, or (ii) $\mathcal{O} \subseteq \bigcup_{R\in\mathcal{R}_{G_2}} \mathsf{attr}(R)$.

Procedure: We compute $R_S = \pi_S \left(\bowtie_{R\in\mathcal{R}_{G_1}} R\right)$. This takes time $P(G_1')$, where G_1' is like G_1 but with $\mathcal{O}_{G_1'} = S$. Let $J_2 = \left(\bowtie_{R\in\mathcal{R}_{G_2}} R\right) \bowtie R_S$. If $\mathcal{O} \subseteq \bigcup_{R\in\mathcal{R}_{G_2}} \mathsf{attr}(R)$, then we compute and output $\pi_{\mathcal{O}} J_2$, and we are done. This step costs $P(G_2)$. Otherwise, $S \subseteq \mathcal{O}$. We compute $O_2 = \pi_{\mathcal{O}} J_2$. Each tuple in O_2 has a matching output tuple for G. Then we set $R_S = R_S \cap \pi_S O_2$ and compute $O_1 = \pi_{\mathcal{O}}(\bowtie_{R\in\mathcal{R}_{G_1}} R \bowtie R_S)$. Then for each tuple $t \in R_S$, we take each pair of matching tuples $t_1 \in O_1$, $t_2 \in O_2$ and output $t_1 \bowtie t_2$. Let G_1'' be like G_1, but with $\mathcal{O}_{G_1''} = \mathcal{O} \cap \left(\bigcup_{R\in\mathcal{R}_{G_1}} \mathsf{attr}(R)\right)$, and G_2'' be defined similarly. This gives us:

Bound: If $S \subseteq \mathcal{O}$, then $Q(G) \leq P(G_1') + Q(G_1'') + Q(G_2'')$. If $\mathcal{O} \subseteq \bigcup_{R\in\mathcal{R}_{G_2}} \mathsf{attr}(R)$, then $P(G) \leq P(G_1') + P(G_2)$.

4.2.2 Combining the Transforms

Once we know the transforms, the DARTS algorithm is quite straightforward. It considers all possible sequences of transforms that can be used to solve the problem, and picks the one that gives the smallest upper bound on $Q(G)$. The number of such transform sequences is exponential in the number of attributes and relations, but constant with respect to data size. The P and Q values of various Gs can be computed recursively given a degree configuration. The G' obtained in each recursive step itself specifies a degree configuration, over a smaller problem. The degrees in G' can be computed in terms of degrees in G. Note that in some cases, we do not have cost bounds available e.g. we do not have a P bound for the Split transform when $S \subseteq \mathcal{O}$. This is a part of the DARTS algorithm. DARTS only considers performing a transform when it can upper bound the resulting cost.

We show that DARTS can be used to recover existing results on sequential joins.

▶ **Proposition 15.** If we compute the join using a single Light transform, our total cost is $\leq$ the AGM bound, thus recovering the result of the NPRR algorithm [16].

▶ **Proposition 16.** If we successively apply the Split transform on an α-acyclic join, with G_1 being an ear of the join in each step, then the total cost of our algorithm becomes $O(\mathrm{IN} + \mathrm{OUT})$, recovering the result of Yannakakis' algorithm [19].

▶ **Proposition 17.** If a query has fractional hypertree width equal to fhw, then using a combination of Split and Light transforms, we can bound the cost of running DARTS by $O(\text{IN}^{fhw} + \text{OUT})$, recovering the fractional hypertree width result.

▶ **Proposition 18.** A cycle join of length n with all relations having size N, can be processed by DARTS in time $O(N^{2-\frac{1}{1+\lceil \frac{n}{2} \rceil}} + \text{OUT})$, recovering the result of the AYZ algorithm [4].

In the next subsection, we present a few of the cases in which we can go *beyond* existing results. Since we are primarily interested in joins, the output attribute set $\mathcal{O}$ below is always assumed to be $\mathcal{A}$.

4.3 Subquadratic Joins

Now we consider applications of DARTS to the subquadratic joins problem. Analyzing a run of DARTS on a join graph allows us to obtain a subquadratic runtime upper bound in several cases. We now define a set of graphs for which we have a complete decision procedure to determine if they can be solved in subquadratic time modulo the 3-SUM problem.

1-series-parallel graphs

▶ **Definition 19.** A 1-series-parallel graph is one that consists of :
- A source node X_S
- A sink node X_T
- Any number of paths, of arbitrary length, from X_S to X_T, having no other nodes in common with each other

Equivalently, a 1-series-parallel graph is a series parallel graph that can be obtained using any number of series transforms (which creates paths) followed by exactly one parallel transform, which joins the paths at the endpoints. A cycle is a special case of a 1-series-parallel graph.

▶ **Theorem 20.** *For* 1*-series-parallel graphs, the following decision procedure determines whether or not the join over that graph can be processed in sub-quadratic time:*
1. *If there is a direct edge (path of length one) between X_S and X_T, then the join can be processed in sub-quadratic time. Else:*
2. *Remove all paths of length two between X_S and X_T, as they do not affect the sub-quadratic solvability of the join problem. Then*
3. *If the remaining number of paths (obviously all having length ≥ 3) is ≥ 3, then the join cannot be processed in subquadratic time (modulo 3-SUM). If the number of remaining paths is < 3, then the graph can be solved in sub-quadratic time.*

Theorem 20 establishes the decision procedure for subquadratic solvability of 1-series-parallel graphs. The full version gives an example of a subquadratic solution for a specific 1-series-parallel graph, namely $K_{2,n}$, followed by an example on the general bipartite graph $K_{m,n}$. In both these examples, DARTS achieves a better runtime exponent than previously known algorithms. We now make three statements that together imply Theorem 20 (formally stated and proved in the full version).

- If we have a 1-series-parallel graph, which has a direct edge from X_S to X_T (i.e. a path of length 1), then a join on that graph can be processed in subquadratic time.
- Suppose we have a 1-series-parallel graph G, which does not have a direct edge from X_S to X_T, but has a vertex X_U such that there is an edge from X_S to X_U and from X_U to X_T (i.e. a path of length 2 from X_S to X_T). Let G' be the graph obtained by deleting

the vertex X_U and edges X_SX_U and X_UX_T. Then the join on G can be processed in subquadratic time if and only if that on G' can be processed in subquadratic time.

- Let G be any 1-series-parallel graph which does not have an edge from X_S to X_T, but has ≥ 3 paths of length at ≥ 3 each, from X_S to X_T. Then a join over G can be processed in subquadratic time only if the 3-SUM problem can be solved in subquadratic time.

4.4 A new notion of width (m-width)

We demonstrate a way to formulate the DARTS algorithm for joins in terms of GHDs.

For each $A \in \mathcal{A}$, we define m_A similarly to how we defined $m_{\mathcal{A}}$ in Section 3.3. Specifically, for each A, we use the same constraints as in linear program 2, but the objective is set to Maximize s_A instead of Maximize $s_{\mathcal{A}}$. m_A is then defined as the value of this objective function. We let $\mathsf{Prog}(A)$ denote the above linear program for finding m_A. Then the size $|\pi_A(\bowtie_{R\in\mathcal{R}} R)|$ must be bounded by IN^{m_A} for all $A \subseteq \mathcal{A}$. Moreover, for any GHD $D = (\mathcal{T}, \chi)$ of query $\mathcal{R}$, we can define $\mathsf{MW}(D, \mathcal{R})$ to be $\max_{t\in\mathcal{T}}(m_{\chi(t)})$. And $\mathsf{MW}(\mathcal{R})$ is simply the minimum value of $\mathsf{MW}(D, \mathcal{R})$ over all GHDs D. Thus we have:

▶ **Definition 21.** The m-width of a join query $\bowtie_{R\in\mathcal{R}} \mathcal{R}$ (possibly with non-uniform degrees), is given by $\max_{c\in\mathcal{C}_2} \mathsf{MW}(\mathcal{R}(c))$.

▶ **Theorem 22.** *A query with m-width* MW *can be answered in time* $O(\mathrm{IN}^{\mathsf{MW}} + \mathrm{OUT})$.

This theorem lets us recover all our subquadratic joins results as well. That is, for the 1-series-parallel graphs that have a subquadratic join algorithm (as per Theorem 20), we can construct a GHD that has m-width less than 2.

The MO bound is tighter than the DBP bound (and consequently, the AGM bound, as stated in Theorem 9 earlier).

▶ **Theorem 23.** *For any join query $\mathcal{R}$, and any degree configuration $c \in \mathcal{C}_2$,* $\mathsf{MO}(\mathcal{R}(c)) \leq \mathsf{DBP}(\mathcal{R}(c), 2) + |C|\log(2)$*, where C is the cover used in the DBP bound.*

Note that since logarithms are to the base IN, the $|C|\log(2)$ term is negligible even though it goes in the exponent of the bound i.e. its exponent is a constant. Theorems 22 and 23 let us recover all the results of the DARTS algorithm.

The theorems also imply that our new notion of width (m-width) is tighter than fhw. The full version [12] compares m-width to submodular width (which, barring m-width, is the tightest known notion of width applicable to general joins), showing examples where m-width is tighter than submodular width. But we do not know in general if m-width is tighter than submodular width.

The full version also shows that while m-width < 2 implies subquadratic solvability, the converse is not true; we show an example join which has m-width and submodular width $= 2$ but can be solved in subquadratic time. Thus known notions of width do not fully characterize subquadratically solvable graphs.

5 Parallel Join Processing

Like in sequential settings, degree-uniformization can be applied in a MapReduce setting. We first present the DBP bound, which is a bound on output size that is tighter than AGM bound (but not tighter than MO), and characterizes the complexity of our parallel algorithm. Then we present a 3-round MapReduce algorithm whose cost equals the DBP bound at the highest level of parallelism.

The DBP Bound

We start by defining a quantity called the Degree-based packing (DBP).

▶ **Definition 24.** Let $\mathcal{R}$ be a set of relations, with attributes in $\mathcal{A}$. Let C denote a cover i.e. a set of pairs (R, A) such that $R \in \mathcal{R}$, $A \subseteq \mathsf{attr}(R)$, and $\bigcup_{(R,A)\in C} A = \mathcal{A}$. Let $L > 1$. Then, consider the following linear program for C, L.

▶ **Linear Program 3.**

$$\text{Minimize} \sum_{a\in\mathcal{A}} v_a \text{ such that } \forall(R,A)\in C, \forall A' \subseteq A : \sum_{a\in A'} v_a \geq \log\left(\frac{d_{\pi_A(R),A\setminus A'}}{L}\right)$$

If $O_{C,L}$ is the maximum objective value of the above program, then we define $\mathsf{DBP}(\mathcal{R}, L)$ to be $\min_C O_{C,L}$ where the minimum is taken over all covers C.

▶ **Proposition 25.** Let $L > 1$ be a constant. Then the output size of $\bowtie_{R\in\mathcal{R}} R$ is in $O(\mathrm{IN}^{\mathsf{DBP}(\mathcal{R},L)})$.

We implicitly prove this result by providing a parallel algorithm whose complexity equals the output size bound at the maximum parallelism level. We can now define the DBP bound. We arbitrarily set $L = 2$ for this definition (choosing another constant value only changes the bound by a constant factor). Thus, we define the *DBP bound* to be $\sum_{c\in\mathcal{C}_2} \mathrm{IN}^{\mathsf{DBP}(\mathcal{R}(c),2)}$. As a simple corollary, the output size of the join is $\leq$ the DBP bound.

▶ **Theorem 26.** *For each degree configuration $c \in \mathcal{C}_L$, $\mathrm{IN}^{\mathsf{DBP}(\mathcal{R}(c),L)} \leq \mathsf{AGM}(\mathcal{R}(c))$.*

We prove this theorem using a sequence of linear program transformations, starting with the AGM bound, and ending with the DBP bound, which each transformation decreasing the objective function value. The key transform is the fifth one, where we switch from a cover-based program to a packing-based program. We show in the full version that the DBP bound has a strictly better exponent than AGM for 'almost all' degrees.

Parallel Join Algorithm

We present our parallel 3-round join algorithm. The algorithm works at all levels of parallelism specified by load level L. Its communication cost matches the DBP bound when $L = O(1)$. We formally state the result, and then provide an example of its performance.

▶ **Theorem 27.** *For any value of L, we can process a join in $O(\log_L(\mathrm{IN}))$ rounds (three rounds if degrees are already known) with load $O(L)$ per processor and a communication cost of $O(\mathrm{IN} + \mathrm{OUT} + \max_{c\in\mathcal{C}_L} L \cdot \mathrm{IN}^{\mathsf{DBP}(\mathcal{R}(c),L)})$.*

We briefly sketch the algorithm here, and provide the full proof in the full version. We start by performing degree uniformization. Now consider any configuration c. We solve Linear Program 3 over all covers. Let C be the optimal cover, and v_a the values in the optimal solution to Linear Program 3 with cover C. We join $\bowtie_{(R,A)\in C} \pi_A(R)$ using the Shares algorithm with share IN^{v_a} for attribute a. Finally, we semijoin relations not in C with the result. The following lemma gives us our required communication cost and load bounds.

▶ **Lemma 28.** *The shares algorithm, where each attribute a has share IN^{v_a}, where v_a is from the solution to Linear Program 3, has a load of $O(L)$ per processor with high probability, and a communication cost of $O(\max_{c\in\mathcal{C}_L} L \cdot \mathrm{IN}^{\mathsf{DBP}(\mathcal{R}(c),L)})$.*

▶ **Example 29.** Consider the sparse triangle join, with $\mathcal{R} = \{R_1(X,Y), R_2(Y,Z), R_3(Z,X)\}$. Each relation has size N, and each value has degree $O(1)$. When the load level is $L < N$, the join requires $\mathsf{DBP}(\mathcal{R}, L) = \frac{N}{L}$ processors. Equivalently, when we have p processors, the load per processor is $\frac{N}{p}$, which means it decreases as fast as possible as a function of p.

In contrast the vanilla shares algorithm allocates a share of $p^{\frac{1}{3}}$ to each attribute, and the load per processor is $Np^{-\frac{2}{3}}$. Current state of the art work [8] has a load of $Np^{-\frac{2}{3}}$ as well.

We further explore and generalize this example in the full version, where we also show an example where our parallel algorithm operating at maximum parallelism still has lower total cost than existing state-of-the-art sequential algorithms.

6 Conclusion and Future Work

We demonstrated that using degree information for a join can let us tighten the exponent of our output size bound. We presented a parallel algorithm that works at all levels of parallelism, and whose communication cost matches a tightened bound at the maximum parallelism level. We proposed the question of deciding which joins can be processed in subquadratic time, and made some progress towards answering it. We showed a tight quadratic lower bound for a family of joins, making it the only known tight bound that makes no assumptions about the matrix multiplication exponent. We presented an improved sequential algorithm, namely DARTS, that generalizes several known join algorithms, while outperforming them in several cases. We recovered the results of DARTS in the GHD framework, using a novel notion of width that is tighter than fhw and sometimes tighter than submodular width as well.

We presented several cases in which DARTS outperforms existing algorithms, in the context of subquadratic joins. However, it is likely that DARTS outperforms existing algorithms on joins having higher treewidths as well. A fuller exploration of the improved upper bounds achieved by DARTS is left to future work. The full version shows a join that can be performed in subquadratic time despite its m-width/submodular width being $= 2$. Thus the problem of precisely characterizing which joins can be performed in subquadratic time remains open. Moreover, we focused entirely on using degree information for join processing; using other kinds of information stored by databases to improve join processing is a promising direction for future work.

Acknowledgements. The authors would like to thank Atri Rudra for pointing out the connection to submodular width. CR gratefully acknowledges the support of the Defense Advanced Research Projects Agency (DARPA) XDATA Program under No. FA8750-12-2-0335 and DEFT Program under No. FA8750-13-2-0039, DARPAs MEMEX program under No. FA8750-14-2-0240, the National Science Foundation (NSF) under CAREER Award No. IIS-1353606, Award No. CCF-1356918 and EarthCube Award under No. ACI-1343760, the Office of Naval Research (ONR) under awards No. N000141210041 and No. N000141310129, the Sloan Research Fellowship, the Moore Foundation Data Driven Investigator award, and gifts from American Family Insurance, Google, Lightspeed Ventures, and Toshiba.

References

1 F. Afrati, M. Joglekar, C. Ré, S. Salihoglu, and J. Ullman. GYM: A multiround join algorithm in mapreduce. *CoRR*, abs/1410.4156, 2014. URL: `http://arxiv.org/abs/1410.4156`.

2 F. N. Afrati and J. D. Ullman. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE TKDE*, 23, 2011.

3 N. Alon, I. Newman, A. Shen, G. Tardos, and N. Vereshchagin. Partitioning multi-dimensional sets in a small number of “uniform” parts. *Eur. J. Comb.*, 28, 2007.

4 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles (extended abstract). In *Proceedings of the Second Annual European Symposium on Algorithms*, ESA’94, pages 354–364, London, UK, 1994. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=647904.739463`.

5 A. Atserias, M. Grohe, and D. Marx. Size Bounds and Query Plans for Relational Joins. *SIAM J. Comput.*, 42, 2013.

6 Ilya Baran, ErikD. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3sum. In F. Dehne, A. Lopez-Ortiz, and J. Sack, editors, *Algorithms and Data Structures*, volume 3608 of *Lecture Notes in Computer Science*, pages 409–421. Springer Berlin Heidelberg, 2005. `doi:10.1007/11534273_36`.

7 P. Beame, P. Koutris, and D. Suciu. Communication Steps for Parallel Query Processing. In *PODS*, 2013.

8 P. Beame, P. Koutris, and D. Suciu. Skew in Parallel Query Processing. In *PODS*, 2014.

9 C. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. *TCS*, 239, 2000.

10 G. Gottlob, M. Grohe, M. Nysret, S. Marko, and F. Scarcello. Hypertree Decompositions: Structure, Algorithms, and Applications. In *WG*, 2005.

11 J. Gross, J. Yellen, and P. Zhang. *Handbook of Graph Theory, Second Edition.* Chapman & Hall/CRC, 2nd edition, 2013.

12 M. Joglekar and C. Ré. It’s all a matter of degree: Using degree information to optimize multiway joins. *CoRR*, abs/1508.01239, 2015. URL: `http://arxiv.org/abs/1508.01239`.

13 P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS’11, pages 223–234, New York, NY, USA, 2011. ACM. `doi:10.1145/1989284.1989310`.

14 J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014. URL: `http://snap.stanford.edu/data`.

15 D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60, 2013.

16 H. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms: [extended abstract]. In *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS’12, pages 37–48, New York, NY, USA, 2012. ACM. `doi:10.1145/2213556.2213565`.

17 H. Ngo, C. Ré, and A. Rudra. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *SIGMOD*, 42, 2014.

18 T. Veldhuizen. Leapfrog triejoin: a worst-case optimal join algorithm. *CoRR*, abs/1210.0481, 2012. URL: `http://arxiv.org/abs/1210.0481`.

19 M. Yannakakis. Algorithms for Acyclic Database Schemes. In *VLDB*, 1981.