Query Evaluation by Circuits

Yilei Wang Ke Yi

The Hong Kong University of Science and Technology

(ywanggq, yike)@cse.ust.hk

Abstract

In addition to its theoretical interest, computing with circuits has found applications in many other areas such as secure multi-party computation and outsourced query processing. Yet, the exact circuit complexity of query evaluation had remained an unexplored topic. In this paper, we present circuit constructions for conjunctive queries under degree constraints. These circuits have polylogarithmic depth and their sizes match the polymatroid bound up to polylogarithmic factors. We also propose a definition of output-sensitive circuit families and obtain such circuits with sizes matching their RAM counterparts.

1 Introduction

A classical result in database theory states that (the decision version of) any relational algebra query, when the query size is taken as a constant, is in NC, i.e., it can be solved by a circuit of polylogarithmic depth and polynomial size (see e.g. Section 17.1 of [1]). While polynomial time/size used to be considered "efficient", many recent efforts have taken a more fine-grained approach, trying to determine the exact polynomial $\tilde{O}(N^k)$ for a given query, where N is the total size of all the relations. In the past two decades, there has been tremendous progress in efficient RAM algorithms for *conjunctive queries (CQs)*, and the optimal k can be achieved in many cases [28, 31, 3, 20].

Besides theoretical elegance, a circuit result is stronger than its counterparts in other models of computation: By Brent's Theorem [12], a circuit of size W and depth D can be evaluated on a P-processor PRAM in $O(W/P + D)$ steps for any P, which in turn can be simulated in $O(W)$ steps on a single RAM. The opposite direction of both simulations is possible with a polynomial overhead [29], but it is a major open problem in complexity theory whether the overhead can be reduced to polylogarithmic in either case.

We see a gap here. The classical circuit construction [1] is actually quite naive, using $\tilde{O}(N^m)$ gates where m is the number of relations in the query; on the other hand, the recent worst-case efficient RAM algorithms have $k < m$, but they are all sequential algorithms, at least as written. There are some intuitive arguments that these algorithms have high levels of parallelism, but there is no formal proof. Notably, there have been a series of papers on query evaluation in the *massively parallel computation (MPC)* model [26, 24, 30]. Indeed, any MPC algorithm can be simulated on the PRAM, but we have not been able to match the best RAM running times with MPC algorithms for all CQs. In fact, a recent negative result [22] shows that, for some CQs, MPC is polynomially weaker than the RAM.

We close this gap in this paper. Our main result is a circuit whose depth is polylogarithmic and whose size matches the *polymatroid bound* [20, 3], which is the best known efficiently computable query size upper bound, up to polylogarithmic factors. By plugging this circuit into the *generalized hypertree decomposition* [19] framework, together with a new circuit version of the *Yannakakis algorithm* [34], we also obtain output-sensitive circuits with sizes matching their RAM counterparts. Compared with the classical result [1], our new results more quantitatively formalize the intuition that CQs can be evaluated efficiently in parallel.

Beyond complexity theory, circuits of small depth and size have found applications in many other areas. We mention a few below.

Query evaluation by hardware Computations that require the best performance are often carried out using hardware. While traditionally it was expensive to build custom chips, new programmable hardware, such as PAL and FPGA, has made it much more economical, so one can build customized hardware for frequently asked queries. Here, the circuit size directly translates into the cost (both the fabrication cost and power consumption) of the hardware, while the depth corresponds to the time for query evaluation.

Secure multi-party query evaluation In the *secure multi-party computation* model [14], two or more parties each hold some private data, and they would like to collectively evaluate a function on their joint data, without revealing their own data to others. Nearly all generic protocols in this model, such as *Yao's garbled circuits* [35], *GMW* [18], and *BGW* [11], are based on expressing the function as a circuit. The communication and computation costs of these protocols are proportional to the size of the circuit, while the depth corresponds to the number of communication rounds.

Outsourced query processing Suppose a client uploads her private data to a service provider (e.g., Amazon), and later asks queries on them. To protect her data, the client can encrypt them before uploading. To evaluate a query, the service provider needs to run a program on the encrypted data, which is now possible by *homomorphic encryption* [15, 16]. As homomorphic encryption handles data in a black-box fashion and the program cannot and should not see the plaintext, it cannot make any branching that depends on the actual data, namely, the program's access pattern must be *oblivious* to the input. There are two approaches towards obliviousness: The first is to use *oblivious RAM (ORAM)* [17], which translates each logical access into a polylogarithmic number of physical accesses to random locations of the memory. However, this translation must be done by the client (who knows the logical address to access), resulting in repeated interaction between the service provider and the client. On the other hand, if the program can be expressed as a circuit, which is oblivious by definition, then the process can be made much easier. In particular, evaluating the circuit requires no interaction, except sending the query at the beginning and receiving the query results (in encrypted form) in the end. Note that this requires the service provider to be able to generate the circuit from the given query and the input size, a property known as *uniformity*.

2 Related Work

A number of worst-case efficient RAM algorithms for CQs have been proposed in the literature [28, 31, 3]; among them, we choose to "circuitize" the PANDA algorithm [3] because it can be turned into a *relational circuit*, as we show in Section 4.3. This allows us to focus on designing circuits for individual relational operators. Other algorithms are tuple based, and heavily rely on indexing to achieve the optimal running time, which has no corresponding circuit implementation. Another nice property of PANDA is that it naturally supports general degree constraints as opposed to only cardinality constraints.

Arasu and Kaushik [5] describe oblivious algorithms for common relational operators, including acyclic joins. However, as mentioned above, circuits directly translate to oblivious algorithms but not necessarily vice versa. Their model assumes the existence of a *trusted module (TM)* with $\tilde{O}(1)$ memory size, which allows the client to run some code on private data. Thus, ORAM simulation also works with a TM. Their only theoretical improvement over ORAM simulation is one logarithmic factor. But with the recent discovery of an optimal ORAM [6], this log-factor improvement disappears. ObliDB [13] provides the first end-to-end oblivious query processing system for general queries under the TM assumption, which also uses ORAM to further reduce the cost of some operators. We eliminate the need of a TM (more generally speaking, repeated interaction with the client) by expressing the query evaluation algorithm as a circuit.

SMCQL [10] is the first database system that supports query evaluation under the secure two-party computation model using Yao's garbled circuit. However, the circuit they use is the naive one of the size $\tilde{O}(N^m)$. Secure Yannakakis [32] improves the complexity to linear in the input size plus output size, when the query is a free-connex join-aggregate query. However, their protocol, in particular the one for joins, is not based on garbled circuits. Senate [33] is a query processing system under secure multi-party computation. Their protocol is not purely circuit-based, either.

3 Conjunctive Queries

3.1 Definition

Let $[n] = \{1, \ldots, n\}$. A *conjunctive query* (CQ) Q is defined by a hypergraph $\mathcal{H} \stackrel{\text{def}}{=} ([n], \mathcal{E})$, $\mathcal{E} \subseteq 2^{[n]}$ and a non-negative integer $k \leq n$. Its variables are $\{A_i\}_{i=1}^{n}$. For each $F \in \mathcal{E}$, let R_F be a relation of arity $|F|$. The query is:

$$Q(A_1, \cdots, A_k) \leftarrow \exists(A_{k+1}, \cdots, A_n) \bigwedge_{F \in \mathcal{E}} R_F(\mathbf{A}_F),$$

where $\mathbf{A}_F$ denotes the tuple $(A_j)_{j \in F}$. $A_1, \ldots, A_k$ are called *free variables* and $A_{k+1}, \cdots, A_n$ are the *bound variables.* When $k = n$, i.e., the set of bound variables is empty, we call it a *full query (FCQ).* When $k = 0$, the query becomes a *Boolean query (BCQ)*, which returns a Boolean value that indicates whether the result of FCQ is empty (false) or not (true).

We adopt the notion of *data complexity*, where the query size (i.e., the size of the hypergraph $\mathcal{H}$) is considered a constant, and measure the complexity in terms of the data size (i.e., the relation cardinalities $|R_F|$ for $F \in \mathcal{E}$). We assume that the tuples in the relations draw their values from the integer domain $[u]$. For simplicity we often ignore $\text{poly}(\log(\sum_F |R_F|), \log u)$ factors, signified by the $\tilde{O}(\cdot)$ notation.

In addition to the relation cardinalities, we also use *degree constraints* [3] to measure the complexity, which would yield much tighter bounds when available. For $X \subseteq Y \in \mathcal{E}$ and any $t \in [u]^{|X|}$, define $\deg(Y|t) \stackrel{\text{def}}{=} |\sigma_{X=t}(R_Y)|$ and $\deg(Y|X) \stackrel{\text{def}}{=} \max_t \deg(Y|t)$. Then the 3-tuple $(X, Y, N_{Y|X})$ represents the degree constraint $\deg(Y|X) \leq N_{Y|X}$. Note that a cardinality constraint $|R_Y| \leq N_Y$ can be considered as a special degree constraint $(\emptyset, Y, N_{Y|\emptyset} = N_Y)$, while a functional dependency $\mathbf{A}_X \to \mathbf{A}_Y$ is also a special degree constraint $(X, Y, N_{Y|X} = 1)$. Our definition is slightly different from that in [3]. In their definition, they allow $Y \subseteq F \in \mathcal{E}$. Our restriction $Y = F$, where $F \in \mathcal{E}$, is without loss of generality: If there is a degree constraint on some $Y \subset F \in \mathcal{E}$, we just pre-compute $R_Y = \Pi_Y(R_F)$, and add it to the input relations. We denote by DC the given set of degree constraints. All the degree constraints apply to the input relations; during our development, we may create new relations, and we use $\deg_{R'_Y}(X) \stackrel{\text{def}}{=} \max_t |\sigma_{X=t}(R'_Y)|$ to denote the maximum degree from X to Y in some new relation R'_Y.

3.2 Output Size Bounds for FCQs

A central problem in database theory is to derive upper bounds for the output size of FCQs. The AGM bound is tight when there are only cardinality constraints [7]. However, when the database instance is subject to general degree constraints, the AGM bound can be further reduced. In particular, the following output size bounds have been introduced to take degree constraints into consideration:

Entropic bound Given a query Q defined by hypergraph $\mathcal{H} = ([n], \mathcal{E})$, a function $h : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is said to be *entropic* if there is a joint distribution on $[n]$ such that $h(F)$ is the marginal entropy on F for any $F \subseteq [n]$. Let Γ_n^* be the set of entropic functions. Under degree constraints DC, we have the additional constraint $h(Y|X) \stackrel{\text{def}}{=} h(Y) - h(X) \leq n_{Y|X} \stackrel{\text{def}}{=} \log N_{Y|X}$ for each $(X, Y, N_{Y|X}) \in \mathsf{DC}$. Denote

$$\mathsf{HDC} = \left\{ h : 2^{[n]} \to \mathbb{R}_{\geq 0} \;\middle|\; \bigwedge_{(X,Y,N_{Y|X}) \in \mathsf{DC}} h(Y|X) \leq n_{Y|X} \right\}.$$

Let $|Q(\mathbf{D})|$ be the output size of Q on database instance $\mathbf{D}$. The following is bound is known as the *entropic bound* [20], which is tight in the worst case [3]:

$$\log |Q(\mathbf{D})| \leq \max_{h \in \Gamma_n^* \cap \mathsf{HDC}} h([n]).$$

Polymatroid bound Although the entropic bound is tight, it is not known how to compute it. To address this issue, Gottlob et al. [20] replace entropic functions with polymatroids. A polymatroid is a set function $h : 2^{[n]} \to \mathbb{R}_{\geq 0}$ with $h(\emptyset) = 0$ that satisfies the following two rules (define $h(Y|X) = h(Y) - h(X)$ as before):

(R1) *(Monotonicity):* $h(Y) \geq h(X)$ for any $X \subseteq Y \subseteq [n]$;

(R2) *(Submodularity):* $h(I|I \cap J) \geq h(I \cup J|J)$ for any $I, J \subseteq [n]$.

Let Γ_n be the set of polymatroid functions, then the *polymatroid bound* is defined as $\max_{h \in \Gamma_n \cap \mathsf{HDC}} h([n])$. Any entropic function is a polymatroid, so $\Gamma_n^* \subseteq \Gamma_n$, and the following inequalities hold:

$$\log|Q(\mathbf{D})| \leq \max_{h \in \Gamma_n^* \cap \mathsf{HDC}} h([n]) \leq \max_{h \in \Gamma_n \cap \mathsf{HDC}} h([n]).$$

For simplicity we denote the (degree-aware) polymatroid bound as

$$\mathsf{LOGDAPB}(Q) \overset{\text{def}}{=\!=} \max_{h \in \Gamma_n \cap \mathsf{HDC}} h([n]),\ \mathsf{DAPB}(Q) = 2^{\mathsf{LOGDAPB}(Q)},$$

so that the polymatroid bound can be written as $|Q(\mathbf{D})| \leq \mathsf{DAPB}(Q)$.

The polymatroid bound degenerates into the AGM bound when there are only cardinality constraints, so it is tight in this case. However, Abo Khamis et al. [3] show that it is not tight under general degree constraints. Still, it is the best output size bound under degree constraints, and more importantly, the PANDA algorithm [3], to be reviewed in more detail in Section 4.4, has running time matching this bound up to polylogarithmic factors.

3.3 Shannon-flow Inequality

We call the following inequality a *Shannon-flow inequality* if it holds for any polymatroid function $h \in \Gamma_n$:

$$\sum_{X \subseteq Y \subseteq [n]} \delta_{Y|X} h(Y|X) \geq \sum_{Y \subseteq [n]} \lambda_Y h(Y), \tag{1}$$

where $\{\delta_{Y|X}\}$ and $\{\lambda_Y\}$ are some non-negative real numbers. By representing $\{\delta_{Y|X}\}$, $\{\lambda_X\}$, $\{h(Y|X)\}$ as vectors over (X, Y) pairs (recall that $h(Y) = h(Y|\emptyset)$, so $\lambda_{Y|X} = 0$ for any $X \neq \emptyset$), (1) can be concisely written as $\langle \boldsymbol{\delta}, \boldsymbol{h} \rangle \geq \langle \boldsymbol{\lambda}, \boldsymbol{h} \rangle$, where $\langle \cdot, \cdot \rangle$ denotes dot product. The following theorem states that the polymatroid bound must be the LHS of (1) for some $\boldsymbol{\delta}$.

Theorem 1 ([3])**.** *For any FCQ Q, there exists a non-negative vector $\boldsymbol{\delta}$ such that $\langle \boldsymbol{\delta}, \boldsymbol{h} \rangle \geq h([n])$ is a Shannon-flow inequality, and $\sum_{(X,Y) \in \mathsf{DC}} \delta_{Y|X} \cdot n_{Y|X} = \mathsf{LOGDAPB}(Q)$.*

3.4 Proof Sequence

A main result from [3] is that any Shannon-flow inequality, in particular the one whose LHS corresponds to the polymatroid bound, can be proved by just applying the monotonicity and submodularity rules above. To formalize a *proof sequence*, they also introduce the following two additional rules, which simply follow from the definition $h(Y|X) = h(Y) - h(X)$:

(R3) *(Composition):* $h(X) + h(Y|X) \geq h(Y)$;

(R4) *(Decomposition):* $h(Y) \geq h(X) + h(Y|X)$.

For example, the following Shannon-flow inequality, which corresponds to the AGM bound for the triangle query,

$$h(AB) + h(BC) + h(AC) \geq 2h(ABC), \tag{2}$$

can be proved by applying the four rules as follows:

$$\begin{aligned}
& h(AB) + h(BC) + h(AC) & \\
\geq\ & h(ABC|C) + h(BC) + h(AC) & \text{(submodularity)} \\
\geq\ & h(ABC|C) + h(C) + h(BC|C) + h(AC) & \text{(decomposition)} \\
\geq\ & [h(ABC|C) + h(C)] + [h(ABC|AC) + h(AC)] & \text{(submodularity)} \\
\geq\ & h(ABC) + h(ABC) = 2h(ABC). & \text{(composition)}
\end{aligned}$$

Such a proof can be considered as a sequence of transformations converting the LHS of (1), represented by the vector $\boldsymbol{\delta}$, to its RHS, represented by $\boldsymbol{\lambda}$ (for any Shannon-flow inequality corresponding to a polymatroid bound of an FCQ, we have $\lambda_Y = 1$ if $Y = [n]$ and 0 otherwise). Thus, each rule effectively adds a "rule vector" to $\boldsymbol{\delta}$, eventually converting it to a vector no less than $\boldsymbol{\lambda}$. We denote these rule vectors as $\boldsymbol{s}_{IJ}, \boldsymbol{m}_{XY}, \boldsymbol{c}_{XY}, \boldsymbol{d}_{YX}$, which stand for <u>s</u>ubmodularity, <u>m</u>onotonicity, <u>c</u>omposition, and <u>d</u>ecomposition, respectively. For example, in $\boldsymbol{d}_{YX}$, the component for $(\emptyset, Y)$ is -1, the components for $(\emptyset, X)$ and (X, Y) are $+1$, while the other components are 0. This way, the proof above can be encoded as the following proof sequence:

$$\mathsf{ProofSeq} = (\boldsymbol{s}_{AB,C}, \boldsymbol{d}_{BC,C}, \boldsymbol{s}_{BC,AC}, \boldsymbol{c}_{C,ABC}, \boldsymbol{c}_{AC,ABC}). \tag{3}$$

The steps in a proof sequence may be weighted, so a more general definition is $\mathsf{ProofSeq} = (w_1\boldsymbol{f}_1, \ldots, w_\ell \boldsymbol{f}_\ell)$, where

1. each $\boldsymbol{f}_i \in \{\boldsymbol{s}_{IJ}, \boldsymbol{m}_{XY}, \boldsymbol{c}_{XY}, \boldsymbol{d}_{YX}\}$ is called a *proof step*;
2. the vectors $\boldsymbol{\delta}_0 \stackrel{\text{def}}{=} \boldsymbol{\delta}, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_\ell$ defined by $\boldsymbol{\delta}_i = \boldsymbol{\delta}_{i-1} + w_i \boldsymbol{f}_i$ are non-negative; and
3. $\boldsymbol{\delta}_\ell \geq \boldsymbol{\lambda}$ (element-wise comparison).

The following theorem is a direct result from Theorem B.12 and Proposition B.13 in [25] (the full version of [3]).

Theorem 2. *For any Shannon-flow inequality $\langle \boldsymbol{\delta}, \boldsymbol{h} \rangle \geq \langle \boldsymbol{\lambda}, \boldsymbol{h} \rangle$ with $\|\boldsymbol{\lambda}\|_1 = 1$, there exists a proof sequence of length $O(n^4 \cdot 384^n)$.*

Recall that n is the number of variables. Thus, although the proof sequence may be exponentially long in n, it is a constant in terms of data complexity.

3.5 The PANDA Algorithm

PANDA (Algorithm 1 before our modifications) is a RAM algorithm that takes an FCQ Q, the degree constraints DC, and the database instance $\mathbf{D}$ as inputs, and computes $Q(\mathbf{D})$ in time $\tilde{O}(N + \mathsf{DAPB}(Q))$, where $N = \sum_{F \in \mathcal{E}} |R_F|$ is the total size of all the relations. At a high level, the reader can think of PANDA as a query planner that computes $Q(\mathbf{D})$ via a sequence of $\tilde{O}(1)$ operations[1], where each operation is a projection (line 9 and 16), a join (line 24), a decomposition (line 14), or a union (line 19), and each operation is guaranteed to take time $\tilde{O}(N + \mathsf{DAPB}(Q))$. For decomposition, it decomposes a relation R_Y into $k = O(\log N)$ sub-relations $\{R_Y^{(i)}\}_{i=1}^k$ such that

$$\begin{array}{llll} \text{(a)} & \cup_{i=1}^k R_Y^{(i)} = R_Y, & \text{(b)} & \deg_{R_Y^{(i)}}(X) \leq N_{Y|X}^{(i)}, \\ \text{(c)} & |R_X^{(i)}| \leq N_X^{(i)}, & \text{(d)} & N_X^{(i)} \cdot N_{Y|X}^{(i)} \leq N, \end{array} \tag{4}$$

for some $\{N_X^{(i)}\}$ and $\{N_{Y|X}^{(i)}\}$, where $R_X^{(i)} \stackrel{\text{def}}{=} \Pi_X(R_Y^{(i)})$. Another important concept in PANDA is the notion of a *guard*: A relation R_Y is said to *guard*[2] the degree constraint $(X, Y, N_{Y|X})$ if $X \subseteq Y$ and $\deg_{R_Y}(X) \leq N_{Y|X}$.

4 Circuits

4.1 Boolean Circuits

A Boolean circuit is a directed acyclic graph $C = (V, E)$, where the nodes in V are called the *gates*. All nodes have in-degree[3] at most 2. Gates with no incoming edges are called the *inputs*, while gates with no outgoing edges are the *outputs*. Every gate with two incoming edges is associated with a Boolean operator

[1]The exponent in the polylogarithmic factor depends on the length of the proof sequence.

[2]Our definition of a guard is slightly different from [3]. They also consider a relation R_F as a guard if $X \subseteq Y \subseteq F$ and $\deg_{\Pi_Y(R_F)}(X) \leq N_{Y|X}$. We add the extra requirement $Y = F$, which simplifies the design of the circuits. Accordingly, we modify PANDA (e.g., line 16 of Algorithm 1) to ensure that every degree constraint has a guard.

[3]Some circuit models also consider gates with unlimited in-degree, but this will only make a logarithmic-factor difference in the depth of the circuit.

Algorithm 1: PANDA-C$(\mathcal{R}, \mathsf{DC}, (\boldsymbol{\lambda}, \boldsymbol{\delta}), \mathsf{ProofSeq})$

```
   Input: $\langle \boldsymbol{\delta}, \boldsymbol{h} \rangle \geq \langle \boldsymbol{\lambda}, \boldsymbol{h} \rangle$ is a Shannon-flow inequality with proof sequence $\mathsf{ProofSeq}$
   Input: $\mathsf{DC}$ are input degree constraints guarded by input relations $\mathcal{R}$
1  if $R_B \in \mathcal{R}$ for some $B \in \mathcal{B}$ then
2      return $T_B = R$;
3  Let $\mathsf{ProofSeq} = (w \cdot \boldsymbol{f}, \mathsf{ProofSeq}')$;
4  $\boldsymbol{\delta}' \leftarrow \boldsymbol{\delta} + w \cdot \boldsymbol{f}$;
5  if $\boldsymbol{f} = \boldsymbol{s}_{I,J}$ then
6      return PANDA-C$(\mathcal{R}, \mathsf{DC}, (\boldsymbol{\lambda}, \boldsymbol{\delta}'), \mathsf{ProofSeq}')$;
7  else if $\boldsymbol{f} = \boldsymbol{m}_{X,Y}$ then
8      Let $R_Y$ be a guard for $(\emptyset, Y, N_Y) \in \mathsf{DC}$;
9      $\mathcal{R}' \leftarrow \mathcal{R} \cup \{\Pi_X(R_Y)\}$;
10     ~~$\mathsf{DC}' \leftarrow \mathsf{DC} \cup \{(\emptyset, X, N_X \stackrel{\text{def}}{=} |\Pi_X(R)|)\}$~~;
       $\mathsf{DC}' \leftarrow \mathsf{DC} \cup \{(\emptyset, X, N_X \stackrel{\text{def}}{=} N_Y)\}$;
11     return PANDA-C$(\mathcal{R}', \mathsf{DC}', (\boldsymbol{\lambda}, \boldsymbol{\delta}'), \mathsf{ProofSeq}')$;
12 else if $\boldsymbol{f} = \boldsymbol{d}_{Y,X}$ then
13     Let $R_Y \in \mathcal{R}$ be a guard for $(\emptyset, Y, N_Y) \in \mathsf{DC}$;
14     Decompose $R_Y \to R_Y^{(1)} \cup \cdots \cup R_Y^{(k)}$ subject to (4);
15     for $j \leftarrow 1$ to $k$ do in parallel
16         $\mathcal{R}^{(j)} \leftarrow \mathcal{R} \cup \{\Pi_X(R_Y^{(j)}), R_Y^{(j)}\}$;
17         ~~$\mathsf{DC}^{(j)} \leftarrow \mathsf{DC} \cup \{(\emptyset, X, N_X^{(j)}), (X, Y, N_{Y|X}^{(j)})\}$ as in (86) in [25] (the full version of [3])~~;
           $\mathsf{DC}^{(j)} \leftarrow \mathsf{DC} \cup \{(\emptyset, X, N_X^{(j)}), (X, Y, N_{Y|X}^{(j)})\}$ as line 7 in Algorithm 2;
18         $(T_B^{(j)})_{B \in \mathcal{B}} \leftarrow$ PANDA-C$(\mathcal{R}^{(j)}, \mathsf{DC}^{(j)}, (\boldsymbol{\lambda}, \boldsymbol{\delta}'), \mathsf{ProofSeq}')$;
19     return $\left(T_B \stackrel{\text{def}}{=} \cup_{j=1}^{k} T_B^{(j)}\right)_{B \in \mathcal{B}}$;
20 else if $\boldsymbol{f} = \boldsymbol{c}_{X,Y}$ then
21     Let $R_X$ be a guard of $(\emptyset, X, N_X) \in \mathsf{DC}$;
22     Let $R_W$ be a guard of $(Z, W, N_{W|Z}) \in \mathsf{DC}$ supporting $\delta_{Y|X}$;
23     if $N_X \cdot N_{W|Z} \leq \mathsf{DAPB}(Q)$ then
24         Compute $T_Y \leftarrow R_X \bowtie R_W$;
25         $\mathcal{R}' = \mathcal{R} \cup \{T_Y\}$;
26         ~~$\mathsf{DC}' = \mathsf{DC} \cup \{(\emptyset, Y, N_Y \stackrel{\text{def}}{=} |T_Y|)\}$~~;
           $\mathsf{DC}' = \mathsf{DC} \cup \{(\emptyset, Y, N_Y \stackrel{\text{def}}{=} N_X \cdot N_{W|Z})\}$;
27         return PANDA-C$(\mathcal{R}', \mathsf{DC}', (\boldsymbol{\lambda}, \boldsymbol{\delta}'), \mathsf{ProofSeq}')$;
28     else
29         Let $\langle \boldsymbol{\delta}', \boldsymbol{h} \rangle \geq \langle \boldsymbol{\lambda}', \boldsymbol{h} \rangle$ be the truncated Shannon-flow inequality (see Lemma 5.11 in [25]);
30         Recompute a fresh $\mathsf{ProofSeq}$ for $\langle \boldsymbol{\delta}', \boldsymbol{h} \rangle \geq \langle \boldsymbol{\lambda}', \boldsymbol{h} \rangle$;
31         return PANDA-C$(\mathcal{R}, \mathsf{DC}, (\boldsymbol{\lambda}', \boldsymbol{\delta}'), \mathsf{ProofSeq}')$;
```

from $\{\wedge, \vee\}$, and every gate with one incoming edge must be associated with $\neg$. The *truth value* of an input gate is specified by the input, while the truth values of other gates are defined inductively in the natural way: the truth value of a gate v is the result of applying the logical operator on the truth values of the two (or one) gates connected by the incoming edges to v. The *size* of C is $|V|$, while its *depth* is the length of the longest path from any input gate to any output gate.

For certain applications, it is more efficient to use *arithmetic circuits*, where each edge (wire) carries an integer and the gates can perform basic arithmetic operations: addition, subtraction and multiplication. Since each tuple can be represented by $O(\log u)$ bits and we do not care about polylogarithmic factors, we will not make the distinction between Boolean circuits and arithmetic circuits. For convenience, we will allow each wire to carry an integer, a tuple, or a Boolean value, and each gate can perform any standard operation on them.

4.2 Uniform Circuits

In order to use circuits as algorithms, they need to be *uniform*. Formally, a family of circuits is (logspace) *uniform* if there exists a deterministic Turing machine M, such that M outputs a description of C_N on input 1^N for any N in $O(\log N)$ space, where C_N is the circuit handling inputs of size N. For the problem of query evaluation under degree constraints, the "input size" actually consists of all the degree constraints DC, together with the domain size u. For a query Q, the Turing machine that generates the circuit is required to use space $O(\log(\sum_F |R_F|) + \log u)$. In this paper, the circuits we propose are all uniform.

4.3 Relational Circuits

We will design our circuit in two steps. First, for any query Q, we will design a *relational circuit*, in which each wire carries a relation and each gate is one of the standard relational operators: selection, projection, join, and union. The in-degree of the first two gates is one while it is two for the latter two gates. In the second step (Section 5), we replace each relational gate by a Boolean circuit.

A relational circuit is very similar to a traditional query evaluation plan, but with the following differences.

Pure relational circuits Note that every CQ (or union of CQs) can be evaluated by a relational circuit of constant size, by definition. However, the evaluation cost can vary significantly over different circuits. To define the cost of a relational circuit more precisely, we first define the cost of each relational operator:

1. The cost of selection $\sigma_\varphi(R)$ is $|R|$.

2. The cost of projection $\Pi_F(R)$ is $|R|$.

3. The cost of join $R \bowtie S$ is $|R| + |S| + |R \bowtie S|$.

4. The cost of union $R \cup S$ is $|R| + |S|$.

Then, the cost of a relational circuit is defined as the total costs of all its relational gates on the worst-case database instance. Note that standard algorithms for these operators in the RAM model match these costs asymptotically [1].

Extended relational circuits However, no pure relational circuits using only the four relational operators above achieve a worst-case cost of $\tilde{O}(N + \mathsf{DAPB}(Q))$, even for the triangle query $Q_\triangle = R_{AB} \bowtie R_{BC} \bowtie R_{AC}$. Thus, we equip relational circuits with two more operators: aggregation and sorting. They are not defined in the pure relational algebra, but are routinely found in practical database systems. As we shall see, they can also be easily implemented by circuits.

A (group-by) aggregation operator is a natural extension of projection, hence denoted as $\Pi_{F,\mathbf{agg}(A)}(R)$. It first partitions the input relation R according to a set of attributes F, and then computes the aggregate over attribute A for each group. Common aggregations are **count**, **sum**, **max** and **min**. For the **count** aggregation, we simply write $\Pi_{F,\mathbf{count}}(R)$ without having to specify A. Note that the output relation of $\Pi_{F,\mathbf{agg}(A)}(R)$ consists of $|F| + 1$ attributes: all the attributes in F plus the aggregate.

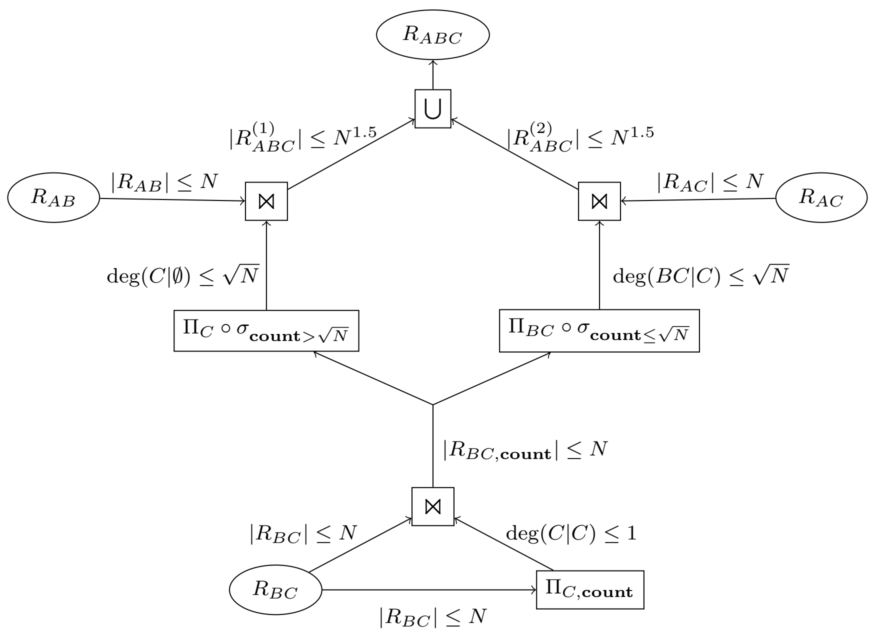

R_{ABC}
$\cup$
$|R^{(1)}_{ABC}| \leq N^{1.5}$
$|R^{(2)}_{ABC}| \leq N^{1.5}$
R_{AB}
$|R_{AB}| \leq N$
$\bowtie$
$\bowtie$
$|R_{AC}| \leq N$
R_{AC}
$\deg(C|\emptyset) \leq \sqrt{N}$
$\deg(BC|C) \leq \sqrt{N}$
$\Pi_C \circ \sigma_{\mathbf{count} > \sqrt{N}}$
$\Pi_{BC} \circ \sigma_{\mathbf{count} \leq \sqrt{N}}$
$|R_{BC,\mathbf{count}}| \leq N$
$\bowtie$
$|R_{BC}| \leq N$
$\deg(C|C) \leq 1$
R_{BC}
$\Pi_{C,\mathbf{count}}$
$|R_{BC}| \leq N$

Figure 1: A relational circuit for the triangle query, where the wires are labeled with cardinality and degree bounds.

Since relations are sets of tuples, sorting takes the form of an ordering operator $\tau_F(R)$. More precisely, $\tau_F(R)$ adds a new **order** column to R, which is the position number of the tuple after sorting R by F, where ties can be broken arbitrarily.

Relational circuits with bounded wires There is another crucial difference between relational circuits and Boolean circuits. For any given query Q, we just design one relational circuit, but with the latter, we have to design a uniform family of circuits, parameterized by the input size (DC for the query evaluation problem). A related, and more technical, issue is that each wire in a relational circuit carries a relation, but each wire in a Boolean circuit can only carry one bit (or one tuple if polylogarithmic factors are ignored). This poses a difficulty in our second step, where we replace each relational gate by a Boolean circuit. In particular, a join gate may produce an output relation of size $|R| \cdot |S|$, so the Boolean circuit for a join must have at least this size in preparation for the worst case.

To resolve the difficulty, we require the relational circuits to have *bounded wires*, i.e., each wire is parameterized by certain degree constraints (including cardinality constraints) and only carries relations conforming to these constraints. As the input wires are bounded, the costs of the relational gates can also be bounded. More precisely, we define their costs as follows:

1. The cost of a selection, projection, aggregation, or sorting gate, which takes an input relation R with the constraint $|R| \leq N$, is N.

2. The cost of a union gate, which takes two input relations R and S with the constraints $|R| \leq N$, $|S| \leq M$, is $M + N$.

3. The cost of a join gate, which takes two input relations R, S with the constraints $|R| \leq M$, $\deg_F(S) \leq N$, $|S| \leq N'$ where F is the set of common attributes between R and S, is $MN + N'$.

Note that the costs defined above only depend on the constraints on the input wires but not the actual instance. In Section 5 we will replace these relational gates by Boolean circuits whose size matches these costs, up to polylogarithmic factors. Thus, the overall size of the Boolean circuit for the query will be exactly the total costs of all the relational gates.

Example 1. *We design an extended relational circuit with bounded wires for for $Q_\triangle$ in Figure 1. For simplicity We consider the case where* DC *only includes cardinality constraints $|R_{AB}| \le N, |R_{BC}| \le N, |R_{AC}| \le N$. In this case,* $\mathsf{DAPB}(Q_\triangle) = O(N^{1.5})$, *and this relational circuit achieves this cost. It uses the now-classical idea of dividing the values in one attribute (C in this example) into heavy (degree $> \sqrt{N}$) and light (degree $\le \sqrt{N}$). This results in bounded wires and the cost of each gate is $O(N^{1.5})$. Note that this relational circuit may produce some false positives, which can be removed by (semi-)joining its output relation R_{ABC} with all input relations. The costs of these extra join gates are all $O(N^{1.5})$ as well. For this particular query under only cardinality constraints, we do not need the sorting operator, which will be needed for general queries under arbitrary degree constraints.*

In the rest of the paper, a *relational circuit* always refers to one with bounded wires and extended with aggregation and sorting gates. Note that since the wires are now parameterized, for a given query Q, we will need to design a family of relational circuits, one for each set of degree constraints DC. To achieve uniformity, the corresponding relational circuit should be generated in log-space.

4.4 PANDA-C: Relational Circuits for FCQs

The relational circuit for the triangle query in Figure 1 has constant size and the optimal cost $O(N^{1.5})$. However, it is an open question whether constant-sized relational circuits exist that achieve $\widetilde{O}(N+\mathsf{DAPB}(Q))$ cost for an arbitrary FCQ, even if there are only cardinality constraints and all cardinality constraints are equal (to N). Note that in this case, $\mathsf{DAPB}(Q) = N^{\rho^*}$ where ρ^* is the minimum fractional edge cover number of the hypergraph $\mathcal{H}$ of the query.

Nevertheless, we are able to design a polylogarithmic-sized relational circuit for an arbitrary FCQ under general degree constraints while achieving cost $\widetilde{O}(N + \mathsf{DAPB}(Q))$, by appropriately modifying PANDA. We call the modified algorithm PANDA-C. There are two differences between PANDA and PANDA-C:

Data independence If we think of PANDA as a query evaluation algorithm, PANDA-C would be a query compiler, i.e., it takes Q and DC as inputs, but not $\mathbf{D}$, and will generate a relational circuit that works for any $\mathbf{D}$ that conforms to DC. In particular, this means that the sequence of operations can only depend on Q and DC. To this end, we modify the original PANDA algorithm as shown in Algorithm 1, where the modified code is marked ~~like this~~.

Decomposition Another missing piece to make PANDA-C a relational circuit is the decomposition operation (see Section 3.5). We design a relational circuit for this operation in Algorithm 2, which has $\widetilde{O}(1)$ size and $\widetilde{O}(N)$ cost. Note that this is where we need the sorting gate.

Algorithm 2: Decomposition Circuit

```
1 $R_{Y,\mathbf{count}} \leftarrow R_Y \bowtie \Pi_{X,\mathbf{count}}(R_Y)$;
2 $k \leftarrow 1 + \lfloor \log N \rfloor$;
3 for $i \leftarrow 1$ to $k$ do in parallel
4     $T_Y^{(i)} \leftarrow \Pi_Y(\sigma_{2^{i-1} \le \mathbf{count} < 2^i}(R_{Y,\mathbf{count}}))$;
5     $R_Y^{(2i-1)} = \Pi_Y(\sigma_{\mathbf{order} \text{ is odd}}(\tau_X(T_Y^{(i)})))$;
6     $R_Y^{(2i)} = \Pi_Y(\sigma_{\mathbf{order} \text{ is even}}(\tau_X(T_Y^{(i)})))$;
7     $N_X^{(2i-1)} = N_X^{(2i)} = \lfloor N/2^{i-1} \rfloor,\ N_{Y|X}^{(2i-1)} = N_{Y|X}^{(2i)} = 2^{i-1}$;
8 return $\{R_Y^{(i)}\}_{i=1}^{2k}$;
```

Correctness Proof to Algorithm 2. We show that the algorithm satisfies the four conditions in (4). Line 1 associates every tuple in R_Y with its degree on X. Then for $i = 1, 2, \ldots, k$, line 4 finds the set of all tuples in R_Y that have degree on X between 2^{i-1} (included) and 2^i (excluded), and denotes it as $T_Y^{(i)}$. Therefore we have $\deg_{T_Y^{(i)}}(X) \le 2^i - 1$ and $|\Pi_X(T_Y^{(i)})| \le N_Y/2^{i-1}$. Line 5–6 further splits $T_Y^{(i)}$ into two equally-sized

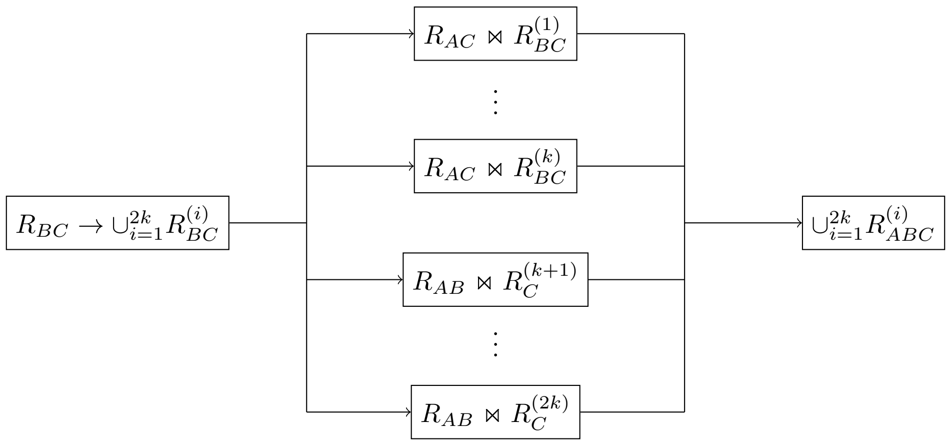

$R_{BC} \to \cup_{i=1}^{2k} R_{BC}^{(i)}$
$R_{AC} \bowtie R_{BC}^{(1)}$
$R_{AC} \bowtie R_{BC}^{(k)}$
$R_{AB} \bowtie R_{C}^{(k+1)}$
$R_{AB} \bowtie R_{C}^{(2k)}$
$\cup_{i=1}^{2k} R_{ABC}^{(i)}$

Figure 2: A relational circuit generated by PANDA-C for the triangle query, where $k = \log N$.

halves based on the parity of the order number, so that the degrees for the same value on X in $R_Y^{(2i-1)}$ and $R_Y^{(2i)}$ differ by at most 1, and hence the maximum degree of two relations are bounded by $\lceil (2^i - 1)/2 \rceil = 2^{i-1}$. Define parameters as in line 7, then the four conditions in (4) can be verified easily. □

Now we are ready to prove the correctness of PANDA-C.

Correctness Proof to Algorithm 1. We follow the proof in [3]. Note that we only change the assignments of DC in Algorithm 1. We only need to prove the following two invariants:

$$\sum_{(X,Y)} n(\delta_{Y|X}) \leq \|\boldsymbol{\lambda}\|_1 \cdot \mathsf{LOGDAPB}, \tag{5a}$$

$$n_{Y|\emptyset} \leq \mathsf{LOGDAPB}. \tag{5b}$$

The other two invariants naturally hold because they do not rely on DC.

The two invariants are satisfied at the beginning as shown in [3]. Now for each proof step $\boldsymbol{f}$:

1. $\boldsymbol{f} = \boldsymbol{s}_{I,J}$ is a submodularity step. The proof for this case is exactly the same with [3].
2. $\boldsymbol{f} = \boldsymbol{m}_{X,Y}$ is a monotonicity step. Now $N_{X|\emptyset} = N_{Y|\emptyset}$ instead of $|\Pi_X(R)|$. However, the inequality $N_{X|\emptyset} \leq N_{Y|\emptyset}$ still holds, so the proof for this case directly follows [3].
3. $\boldsymbol{f} = \boldsymbol{d}_{Y,X}$ is a decomposition step. Recall in Inequalities (4) we have $N_X^{(j)} \cdot N_{Y|X}^{(j)} \leq N_Y$ for each j, so $n_{X|\emptyset}^{(j)} + n_{Y|X}^{(j)} \leq n_{Y|\emptyset}$, which implies invariant (5a). Besides, invariant (5b) holds since $N_X^{(j)} \leq N_Y \leq$ DAPB.
4. $\boldsymbol{f} = \boldsymbol{c}_{X,Y}$ is a composition step. If $N_X \cdot N_{W|Z} \leq$ DAPB, then $N_Y = N_X \cdot N_{W|Z} \leq$ DAPB, so invariant (5b) holds; invariant (5a) holds because $n_{Y|\emptyset} = n_{X|\emptyset} + n_{W|Z}$. If $N_X \cdot N_{W|Z} >$ DAPB, then we also have $n_{X|\emptyset} + n_{W|Z} >$ LOGDAPB, so we prove the two invariants by following the proof of [3].

□

Since PANDA-C has a constant number of recursion levels and each level may produce $O(\log N)$ subproblems due to the decomposition operator, the total size of the relational circuit is polylogarithmic. Also, it is easy to see that it can be ran in log-space. So we conclude the following theorem:

Theorem 3. *There is a log-space Turing machine that, given an FCQ Q and degree constraints* DC*, generates a relational circuit that computes $Q(\mathbf{D})$ for any $\mathbf{D}$ conforming to* DC*. The generated relational circuit has size $\tilde{O}(1)$ and cost $\tilde{O}(N + \mathsf{DAPB}(Q))$.*

Example 2. *Revisiting the triangle query in Example 1, the relational circuit generated by* PANDA-C *is more complicated. For this query, the Shannon-flow inequality is* (2)*, and suppose* PANDA-C *uses the proof sequence* (3)*. Since $\boldsymbol{f}_2 = \boldsymbol{d}_{BC,C}$, it will first decompose R_{BC} into $2\log N$ sub-relations, instead of two as in Example 1. Each sub-relation is joined with either R_{AB} or R_{AC}. Finally, all join results are unioned together, as shown in Figure 2. Again, this relational circuit may produce some false positives, which can be removed by joining the results with the input relations.*

5 From Relational Circuits to Boolean Circuits

In this section, we replace each relational gate in the relational circuit generated from the previous section with a Boolean circuit. Recall that each wire in the PANDA-C relational circuit carries a relation of size bounded by, say, K. In the Boolean circuit, we replace it with Boolean wires that carry exactly K tuples (which require $\tilde{O}(K)$ Boolean wires). Of course, on any actual data instance $\mathbf{D}$, there may not be exactly K tuples. In this case, we simply pad *dummy* tuples. One way to implement this is to add a special Boolean attribute Z to each tuple t, and set $t.Z = 0$ if the tuple t is dummy and $t.Z = 1$ if not. For conciseness, we will only say "set t to dummy" or "check whether t is dummy" while the underlying implementation is obvious.

Some relational gates have straightforward circuits:

- Selection $\sigma_\varphi(R)$ can be trivially implemented by a Boolean circuit of $\tilde{O}(1)$ depth and $\tilde{O}(N)$ size under the constraint $|R| \le N$. Note that in the circuit implementation, all tuples in R will be in the output, except that those that do not pass the filter condition φ will be set to dummy.
- Ordering (sorting) $\tau_F(R)$ can be implemented by a sorting network, such as the Bitonic sorter [9] or the AKS network [4], both of which have $\tilde{O}(N)$ size and $\tilde{O}(1)$ depth. When R has dummy tuples, we ensure that all non-dummy tuples are placed before the dummy tuples after the sorting. Then all the non-dummy tuples will get their correct order number.
- Projection $\Pi_F(R)$ is described in Algorithm 3, where without loss of generality, we consider the projection $\Pi_A(R_{AB})$ subject to $|R| \le N$. The circuit has depth $\tilde{O}(1)$ and size $\tilde{O}(N)$.
- Union $R \cup S$ can be realized by simply applying the projection circuit on all attributes on all tuples in R and S, noting that the projection circuit eliminates duplicates.

Algorithm 3: Projection Circuit

```
1 Remove column $B$ from $R$;
2 Sort $R$ by $A$;
3 Define $t_i$ as the $i$-th tuple of $R$ for any $i$;
4 for $i \leftarrow 2$ to $N$ do in parallel
5     if $t_i.A = t_{i-1}.A$ then
6         Set $t_i$ to dummy;
7 return $R$;
```

Below we describe how to replace other relational gates with Boolean circuits with $\tilde{O}(1)$ depth and size matching the cost of the corresponding relational gate up to polylogarithmic factors. Together these yield the main result of this chapter.

Theorem 4. *For any FCQ Q, there is a uniform family of Boolean circuits computing $Q(\mathbf{D})$ for any $\mathbf{D}$ conforming to given degree constraints* DC*. The circuit has $\tilde{O}(1)$ depth and $\tilde{O}(N + \mathsf{DAPB}(Q))$ size.*

Algorithm 4: $\oplus$-Scan Circuit

```
1 for $i \leftarrow 0$ to $\lceil \log N \rceil - 1$ do
2     for $j \leftarrow 2^i + 1$ to $N$ do in parallel
3         Update $x_j \leftarrow x_{j-2^i} \oplus x_j$;
4 return $(x_j)_{j=1}^N$;
```

5.1 Scan and Segmented Scan

For a sequence $X = (x_i)_{i=1}^N$ and a binary associative operator $\oplus$, a $\oplus$-*scan* on X, also known as the *prefix sums* problem, produces an output sequence $(x_1, x_1 \oplus x_2, \cdots, x_1 \oplus x_2 \oplus \cdots \oplus x_N)$. The classical $\oplus$-*scan circuit* [21] for this problem has depth $\tilde{O}(1)$ and size $\tilde{O}(N)$, as described in Algorithm 4.

For our purpose, we need to extend the scan circuit above to compute a *segmented scan:* We are given a binary associative operator $\oplus$ and a sequence X that consists of k segments $X = (X_1, \ldots, X_k)$. The i-th segment has length N_i, and $X_i = ((a_i, b_{ij}))_{j=1}^{N_i}$ where $a_i \neq a_j$ for any $i \neq j$, namely, the a_i's delineate the segments and we would like to compute the prefix sums segment by segment. More precisely, a $\oplus$-*segmented scan* takes X as input, and produces an output sequence $Y = (Y_1, \ldots, Y_k)$, where Y_i is the $\oplus$-scan of $\{b_{ij}\}_{j=1}^{N_i}$. Since X are represented by two sequences A and B where $X[i] = (A[i], B[i])$, we also call this operation as a $\oplus$-*scan on B segmented by A*. Note that neither k nor any N_i is known to the circuit, so one cannot simply compute each Y_i using a $\oplus$-scan.

We modify the original $\oplus$-scan circuit to perform a $\oplus$-segmented scan. Define the binary operator $\bar{\oplus}$ as follows:

$$(a_1, b_1)\bar{\oplus}(a_2, b_2) = \begin{cases} (a_2, b_1 \oplus b_2), & \text{if } a_1 = a_2, \\ (a_2, b_2), & \text{otherwise.} \end{cases}$$

It can be verified that $\bar{\oplus}$ is associative, and applying a $\bar{\oplus}$-scan on X yields Y exactly.

5.2 Aggregation

Without loss of generality, consider the aggregation $\Pi_{A,\mathbf{agg}(B)}(R)$, where R is a relation with attributes A and B, and $|R| \leq N$. We describe the aggregation circuit in Algorithm 5. The correctness of our circuit is obvious, and the depth and size are $\tilde{O}(1)$ and $\tilde{O}(N)$ respectively.

Algorithm 5: Aggregation Circuit

```
1 Sort $R$ by $A$;
2 Run agg-scan circuit on $\{t.B\}_{t \in R}$ segmented by $\{t.A\}_{t \in R}$;
3 Let $t_i$ be the $i$-th tuple of $R$;
4 for $i \leftarrow 2$ to $N$ do in parallel
5     if $t_i$ is not dummy and $t_i.A = t_{i-1}.A$ then
6         Set $t_{i-1}$ to dummy;
7 return $R$;
```

5.3 Primary Key Join

Join circuits require more work. Recall that the join gate computes $R \bowtie S$ under the constraints $|R| \leq M$, $\deg_F(S) \leq N$, $|S| \leq N'$ where F is the set of common attributes between R and S, and we need to design a circuit of depth $\tilde{O}(1)$ and size $\tilde{O}(MN + N')$ for this purpose. The naive circuit that checks all pairs would result in a size of $O(MN')$.

Without loss of generality, assume R has attributes A, B and S has attributes B, C. We first consider the special case $N = 1$, i.e., B is the primary key of S, so we call this join a *primary key join.*

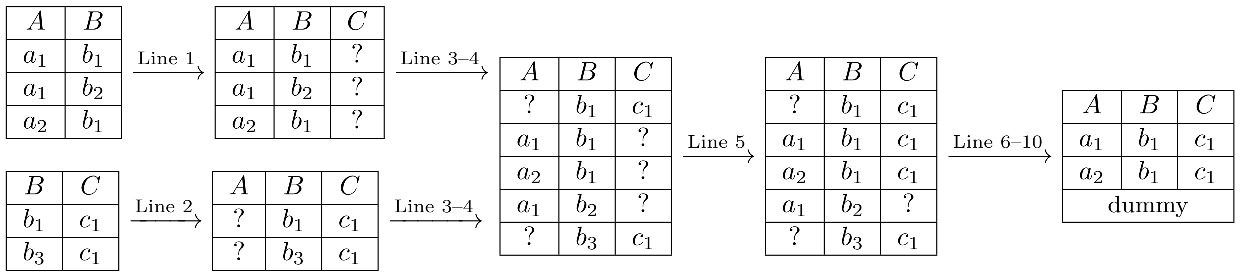


Figure 3: Primary key join example

Circuit Description We assume the existence of a dummy value '?' that will never appear in the database instance. Define the binary (repetition) operator $\oplus$ as $c_1 \oplus c_2 = c_1$. We also introduce the *truncation* operation: When we are certain that a relation R with size N has at most M non-dummy tuples for some $M < N$, we first sort R so that non-dummy tuples are in front of dummy ones, and discard the last $N - M$ tuples of R which must be dummy. Note that when $N \le M$, the truncation does nothing.

The primary key join circuit is described in Algorithm 6. See Figure 3 for a running example.

Algorithm 6: Primary Key Join Circuit

```
1  Add attribute $C$ to $R$ with values set to '?';
2  Add attribute $A$ to $S$ with values set to '?';
3  $J(A,B,C) \leftarrow R(A,B,C) \cup S(A,B,C)$;
4  Sort $J$ by $(B,C)$ such that tuples with $C \neq$ '?' is placed in the front of other tuples with the same
     values on $B$;
5  Run $\oplus$-scan circuit on $\{t.C\}_{t \in J}$ segmented by $\{t.B\}_{t \in J}$;
6  for $t \in J$ do in parallel
7      if $t.A =$ '?' or $t.C =$ '?' then
8          Set $t$ to dummy;
9  Truncate $J$ to size $M$;
10 return $J(A,B,C)$;
```

Correctness Proof to Algorithm 6. Note that for each b, there is at most 1 tuple with $C \neq$ '?' in $\sigma_{B=b}(J)$ after line 3, because B is the primary key of S. If it exists, it is placed at the beginning of $\sigma_{B=b}(J)$, and its value c is copied to other tuples. Otherwise the tuple does not exists, meaning there is no tuple in S that can join with $\sigma_{B=b}(R)$, thus these tuples are set to dummy. $\square$

Complexity Analysis Since $|R| = M$ and $|S| = N'$ (including dummy tuples), $|J|$ has $M + N'$ tuples. All subsequent circuits on J have $\tilde{O}(1)$ depth and $\tilde{O}(M + N')$ size, so does the primary key join circuit.

5.4 Degree-bounded Join

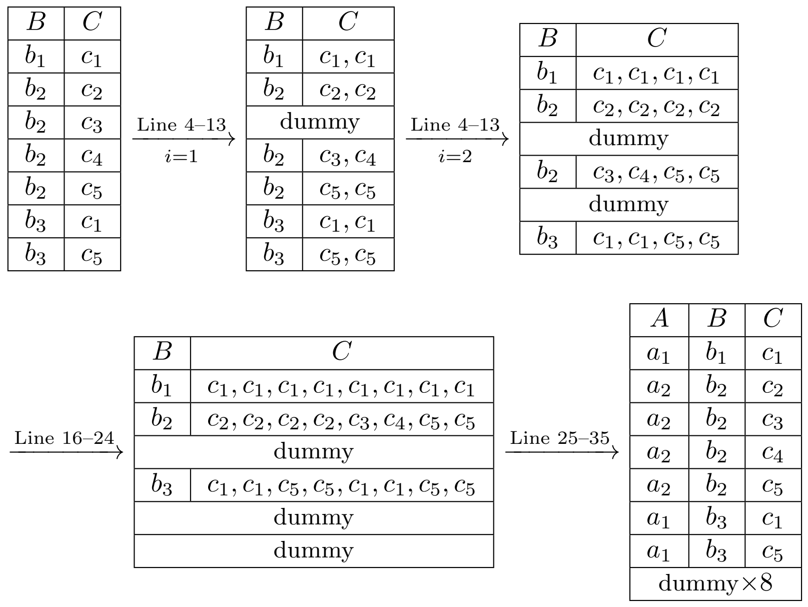
Next, we design the join circuit for a general $N \ge 2$. Assume $N = 2^n + 1$ for some $n \in \mathbb{N}$. Note that if this is not the case, we can relax N to $\bar{N} = 2^{\lceil \log N \rceil} + 1$ without affecting the asymptotic circuit size.

Circuit Description Our circuit works by concatenating values of C in $\sigma_{B=b}(S)$ for each $b \in B$. For this purpose, for each tuple $(b, c) \in S$, we rewrite it as $(b, (c))$, where (c) represents a length-1 sequence. We denote by (c_1, c_2) the concatenation of two sequences c_1 and c_2. The join circuit is described in Algorithm 7, and a running example is given in Figure 4. Below we elaborate on the idea to help with the understanding.

The key observation of the algorithm is that, for any b, tuples with $B = b$ are consecutive. If we combine each pair of neighboring tuples (line 6–11), for each $B = b$, only the first tuple and the last tuple may not succeed, since their neighboring tuples may not have the same value on B or are dummy tuples. It can

Algorithm 7: Degree-bounded Join Circuit

```
Update $S \leftarrow S \bowtie \Pi_B(R)$;
Sort $S$ by $B$; Truncate $S$ to size $MN$;
for $i \leftarrow 1$ to $n$ do
    $N_i \leftarrow |S|$;
    Define $t_j = (b_j, c_j)$ as the $j$-th tuple of $S$ for any $j$;
    for $j \leftarrow 1$ to $\lfloor N_i/2 \rfloor$ do in parallel
        if $b_{2j-1} = b_{2j}$ and $t_{2j}$ is not dummy then
            Update $t_{2j}.C \leftarrow (c_{2j-1}, c_{2j})$;
            Set $t_{2j-1}$ to dummy;
        else
            Update $t_{2j}.C \leftarrow (c_{2j}, c_{2j})$;
    for $j \leftarrow 1$ to $\lceil N_i/2 \rceil$ do in parallel
        Update $t_{2j-1}.C \leftarrow (c_{2j-1}, c_{2j-1})$;
    $N_{i+1} \leftarrow \min\{N_i, (2^{n-i}+1)M\}$;
    Sort $S$ by $B$; Truncate $S$ to size $N_{i+1}$;
$N' \leftarrow |S|$;
Define $t_j = (b_j, c_j)$ as the $j$-th tuple of $S$ for any $j$;
for $j \leftarrow 1$ to $N' - 1$ do in parallel
    if $t_i.B = t_{i+1}.B$ and $t_{i+1}.B$ is not dummy then
        Update $t_i.C \leftarrow (t_i.C, t_{i+1}.C)$;
        Set $t_{i+1}$ to dummy;
    else
        Update $t_i.C \leftarrow (t_i.C, t_i.C)$;
Update $t_{N'}.C \leftarrow (t_{N'}.C, t_{N'}.C)$;
Truncate $S$ to size $M$;
Compute $J(A, B, C) \leftarrow R \bowtie S$ using a primary key join circuit;
Let $J'(A, B, C)$ be an empty relation;
for $t \in J$ do in parallel
    $(a, b, (c_1, \ldots, c_{2^{n+1}})) \leftarrow t$;
    for $i = 1$ to $2^{n+1}$ do in parallel
        Insert $(a, b, c_i)$ to $J'$;
Remove duplicates in $J'$ by updating $J' \leftarrow \Pi_{A,B,C}(J')$;
Truncate $J'$ to size $MN$;
return $J'$;
```



B	C
b_1	c_1
b_2	c_2
b_2	c_3
b_2	c_4
b_2	c_5
b_3	c_1
b_3	c_5

$\xrightarrow[i=1]{\text{Line 4–13}}$

B	C
b_1	c_1, c_1
b_2	c_2, c_2
dummy	
b_2	c_3, c_4
b_2	c_5, c_5
b_3	c_1, c_1
b_3	c_5, c_5

$\xrightarrow[i=2]{\text{Line 4–13}}$

B	C
b_1	c_1, c_1, c_1, c_1
b_2	c_2, c_2, c_2, c_2
dummy	
b_2	c_3, c_4, c_5, c_5
dummy	
b_3	c_1, c_1, c_5, c_5

$\xrightarrow{\text{Line 16–24}}$

B	C
b_1	$c_1, c_1, c_1, c_1, c_1, c_1, c_1, c_1$
b_2	$c_2, c_2, c_2, c_2, c_3, c_4, c_5, c_5$
dummy	
b_3	$c_1, c_1, c_5, c_5, c_1, c_1, c_5, c_5$
dummy	
dummy	

$\xrightarrow{\text{Line 25–35}}$

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_2	b_2	c_3
a_2	b_2	c_4
a_2	b_2	c_5
a_1	b_3	c_1
a_1	b_3	c_5
dummy×8		

Figure 4: Degree-bounded join example, where $M = 3, N = 5, R = \{(a_1, b_1), (a_2, b_2), (a_1, b_3)\}$.

be verified that when the degree of b in S is $d > 1$, then after going through line 6–11, it becomes at most $\lfloor d/2 \rfloor + 1$. Specifically, the degree constraint $\deg(BC|B) \le 2^n + 1$ becomes $\deg(BC|B) \le 2^{n-1} + 1$. Therefore, after repeating line 6–11 for n times, the degree bound becomes $2^{n-1} + 1, 2^{n-2} + 1, \ldots, 2^0 + 1 = 2$. Note that the degree bound cannot be reduced from 2 to 1 by more repetitions. For example, after sorting the third table of Figure 4, the two tuples with $B = b_2$ have no chance to be combined. Therefore, we use a different method for the last combination in line 16–24, which reduces the degree bound to 1, making B the primary key of S. Then we can invoke the primary key join circuit.

The reader may wonder why we do not use the scan circuit to combine the C values, by defining $c_1 \oplus c_2 = (c_1, c_2)$. The problem with this method is that the cost of each $\oplus$ operation is no longer constant, as the size of C doubles after each level. In fact, doing so would result in a circuit of quadratic size. The key to avoiding the size blowup is the observation that, as more C values are combined, the number of non-dummy tuples in S must reduce. Then we throw away some dummy tuples by truncation to reduce the size of S after each level, so that the size of the circuit is only logarithmically larger.

Correctness Proof to Algorithm 7. We prove that we never throw non-dummy tuples in line 14–15. As explained in Section 5.4, we have $\deg(BC|B) \le 2^{n-i} + 1$. Therefore, the number of tuples in S that can join with R is at most $(2^{n-i} + 1)M$, and tuples that cannot join have already been marked as dummy in line 1, hence line 14–15 do not throw non-dummy tuples. $\square$

Complexity Analysis We analyze the complexity of line 1–15 in Algorithm 7, as other parts obviously have $\tilde{O}(1)$ depth and $\tilde{O}(MN)$ size. Line 1–2 computes a projection, primary key join, sorting, and a truncation, so the total depth is $\tilde{O}(1)$ and size is $\tilde{O}(M + N')$. After truncation, $|S| \le MN$ always holds in line 3–15, so the depth of the circuit in each loop is dominated by that of sorting, which is $\tilde{O}(1)$. In the i-th loop, we have $N_i \le (2^{n-i+1} + 1)M$, and the circuit size of line 7–11 and line 13 is $O(2^i)$ (the length of C); the circuit size of the sorting is $\tilde{O}(N_i \cdot 2^i)$. Therefore, the circuit size of line 3–15 is $\tilde{O}\left(\sum_{i=1}^{n} N_i \cdot 2^i\right) = \tilde{O}\left(\sum_{i=1}^{n} (2^{n-i+1} + 1)M \cdot 2^i\right) = \tilde{O}(MN)$. The total size is therefore $\tilde{O}(MN + N')$.

6 Output-sensitive Circuits

Running times of RAM algorithms come in two flavors: $\tilde{O}(f(\mathsf{DC}))$ and $\tilde{O}(g(\mathsf{DC}) + |Q(\mathbf{D})|)$, where the latter is referred to as an *output-sensitive* running time, which is more attractive, both theoretically and practically: Theoretically, because $|Q(\mathbf{D})|$ is the unavoidable cost of reporting the output, $g(\mathsf{DC})$ represents the "intrinsic" cost of the query. Indeed, for many queries we have $g(\mathsf{DC}) < f(\mathsf{DC})$, while $f(\mathsf{DC})$ usually matches the worst-case bound on $|Q(\mathbf{D})|$, so $\tilde{O}(f(\mathsf{DC}))$ is worst-case optimal simply because the output size can be this large on the worst-case $\mathbf{D}$. In practice, an output-sensitive algorithm can benefit from the fact that $|Q(\mathbf{D})|$ is much smaller than $f(\mathsf{DC})$ on most real-world instances.

Unfortunately, since a circuit, hence its size, must be independent of $\mathbf{D}$, the size has to be $\tilde{O}(f(\mathsf{DC}))$ just to prepare for the worst-case input. Thus there is no counterpart of "output-sensitive" algorithms in the circuit model. Nevertheless, we observe that in many applications of circuits, such as parallel computing, MPC, and querying outsourced data (not query evaluation by hardware though), one does not have to complete the entire query evaluation with a single circuit. With these applications in mind, we define an *output-sensitive circuit* to consist of two uniform families of circuits: The first family is parameterized by DC, and each circuit in this family computes $|Q(\mathbf{D})|$ for any $\mathbf{D}$ conforming to DC; the second family is parameterized by DC and OUT, and each circuit in this family computes $Q(\mathbf{D})$ for any $\mathbf{D}$ subject to both the degree constraint DC and the *output size constraint* $|Q(\mathbf{D})| = \text{OUT}$. In any of the applications mentioned above, one uses the first circuit to compute $|Q(\mathbf{D})|$ (for BCQs, this step can be skipped since $|Q(\mathbf{D})| = 1$), and then uses $\text{OUT} = |Q(\mathbf{D})|$ as a parameter to build and evaluate the second circuit. Note that in MPC model or querying outsourced data, this would reveal $|Q(\mathbf{D})|$, but this is allowed, since privacy in these models is defined to be the protection of anything *beyond* the query results, while the output size is certainly part of the query results.

Below, we first review output-sensitive algorithms in the RAM model. In Section 6.2 and 6.3, we show how to construct the second circuit given DC and OUT. Then we discuss how to build the first circuit in Section 6.4.

6.1 Output-sensitive RAM Algorithms

Output-sensitive algorithms for conjunctive query evaluation in the RAM model are obtained by combining the *generalized hypertree decomposition* (GHD) [19], the Yannakakis algorithm [34] for acyclic joins, and a worst-case optimal algorithm for FCQs such as PANDA [3].

We start with the GHD for FCQs:

Definition 1. *Given an FCQ Q with hypergraph $\mathcal{H} = ([n], \mathcal{E})$, a generalized hypertree decomposition is a pair $(\mathcal{T}, \chi)$ of a tree $\mathcal{T} = (\mathcal{V}_{\mathcal{T}}, \mathcal{E}_{\mathcal{T}})$ and function $\chi : \mathcal{V}_{\mathcal{T}} \to 2^{[n]}$ such that*

- *every hyperedge $F \in \mathcal{E}$ is a subset of some $\chi(t)$, $t \in \mathcal{V}_{\mathcal{T}}$; and*
- *for each attribute $i \in [n]$, the tree nodes containing i, i.e., $\{t \in \mathcal{V}_{\mathcal{T}} \mid i \in \chi(t)\}$, form a connected component of $\mathcal{T}$.*

The set $\chi(t)$ of each node t is referred to as a *bag* of the GHD. Given a GHD of Q, one first computes the FCQ associated with each bag $\chi(t)$, $t \in \mathcal{V}_{\mathcal{T}}$, and then runs the Yannakakis algorithm [34] on $\mathcal{T}$. The total running time will be the total time of computing all the bags, plus $O(|Q(\mathbf{D})|)$. As the query size is considered a constant, we find the GHD that minimizes the maximum running time for computing the bags. This leads to various definitions of *width*. Plugging in PANDA as the algorithm for computing each bag, we obtain a running time of $\tilde{O}(N + 2^{\mathsf{da\text{-}fhtw}(Q)} + |Q(\mathbf{D})|)$, where

$$\mathsf{da\text{-}fhtw}(Q) \overset{\text{def}}{=\!=} \min_{(\mathcal{T},\chi)} \max_{t \in \mathcal{V}_{\mathcal{T}}} \max_{h \in \Gamma \cap \mathsf{HDC}} h(\chi(t)) \tag{6}$$

is the *degree-aware fractional hypertree width* of Q.

For a non-full conjunctive query, a common approach is to restrict the GHD to be *free-connex* [8], which, intuitively speaking, puts all free variables above the bound variables. The only change we have to make is that in (6), $(\mathcal{T}, \chi)$ only ranges over free-connex GHDs. Note that for non-full conjunctive queries, $|Q(\mathbf{D})|$ is usually smaller than on the FCQ, but $\mathsf{da\text{-}fhtw}(Q)$ becomes larger as the GHD is restricted to be free-connex. For a BCQ Q, all GHDs are free-connex (since there are no free variables) and $|Q(\mathbf{D})| = 1$, so a BCQ can be evaluated in time $\tilde{O}(N + 2^{\mathsf{da\text{-}fhtw}(Q)})$ without any restriction on the GHD.

6.2 Yannakakis by Circuits

In order to turn GHD-based RAM algorithms to output-sensitive circuits (as defined above), we need two more components in addition to PANDA-C: The first is a circuit for computing $|Q(\mathbf{D})|$ given DC, and the second is a circuit version of the Yannakakis algorithm for given DC and $\text{OUT} = |Q(\mathbf{D})|$. The first circuit has $\tilde{O}(1)$ depth and $\tilde{O}(N + 2^{\mathsf{da\text{-}fhtw}(Q)})$ size, while the second has $\tilde{O}(1)$ depth and $\tilde{O}(N + 2^{\mathsf{da\text{-}fhtw}(Q)} + \text{OUT})$ size, Thus, the total size of the two circuits matches the RAM running time. We describe the second circuit, called Yannakakis-C, in this subsection; the first circuit is actually a simple variant of this circuit.

We adopt the 3-phase version of the Yannakakis algorithm as described in [32]. Let $(\mathcal{T}, \chi)$ be a free-connect GHD of a given query Q. In the first phase, which we call *reduce*, after computing the FCQ associated with each bag, it performs a bottom-up pass over $\mathcal{T}$ to remove all bounded variables by projections and semijoins. Now $\mathcal{T}$ becomes an acyclic FCQ. In the second phase, it makes two semijoin passes to remove all *dangling tuples*, i.e., tuples that will not appear in the query results. In the third phase, it performs another bottom-up pass of joins to compute the query results.

The circuit for the reduce phase, Reduce-C, is described in Algorithm 8. This is readily a circuit, observing that a semijoin $R_X \ltimes S_Y$ can be implemented as $R_X \bowtie \Pi_{X \cap Y}(S_Y)$. After the projection, the join becomes a primary key join, so a semijoin can be implemented by a circuit of $\tilde{O}(1)$ depth and $\tilde{O}(|R_X| + |R_Y|)$ size. Thus, Reduce-C has $\tilde{O}(1)$ depth and $\tilde{O}(N + 2^{\mathsf{da\text{-}fhtw}(Q)})$ size.

Algorithm 8: Reduce-C(Q)

```
1  Let $(\mathcal{T}, \chi)$ be a free-connex GHD of $Q$;
2  for $v \in \mathcal{V}_{\mathcal{T}}$ do in parallel
3      $B \leftarrow \chi(v)$;
4      Let $\langle \boldsymbol{\delta}_v, \boldsymbol{h}_v \rangle \geq h(B)$ be the Shannon-flow inequality such that
         $\langle \boldsymbol{\delta}_v, \boldsymbol{h}_v \rangle = \max\{h(B) \mid h \in \Gamma_n \cap \mathsf{HDC}\}$ (see Theorem 1);
5      Let $\mathsf{ProofSeq}_v$ be the proof sequence of $\langle \boldsymbol{\delta}_v, \boldsymbol{h}_v \rangle \geq h(B)$ (see Theorem 2);
6      $T_B \leftarrow \mathsf{PANDA\text{-}C}(\mathcal{R}, \mathsf{DC}, (\boldsymbol{\delta}_v, \boldsymbol{h}_v), \mathsf{ProofSeq}_v)$;
7  for $v \in \mathcal{V}_{\mathcal{T}}$ in bottom-up manner (root excluded) do
8      $p \leftarrow$ the parent of $v$;
9      $B \leftarrow \chi(v)$; $B_p \leftarrow \chi(p)$;
10     $F \leftarrow$ the set of free variables in $B$;
11     if $F \subseteq B_p$ then
12         $T_{B_p} \leftarrow T_{B_p} \ltimes T_B$;
13         Remove $v$ from $\mathcal{T}$;
14     else
15         Update $\chi(v) \leftarrow F$ and $T_F \leftarrow \Pi_F(T_B)$;
16         Stop going upwards from $v$;
17 return $(\mathcal{T}, \chi, \{T_{\chi(v)}\}_{v \in \mathcal{V}_{\mathcal{T}}})$;
```

The second phase, described in line 2–9 of Algorithm 9, also only needs semijoins, so can be implemented by a circuit of $\tilde{O}(1)$ depth and linear size.

The third phase (line 10–18 of Algorithm 9) requires some more work since it uses joins (line 14) over relations whose degrees may not be bounded. In general, such a join needs a circuit of size $|T_B| \cdot |T_{B_p}|$, which may be greater than OUT. However, after all dangling tuples have been removed, it is guaranteed that $|T_B \bowtie T_{B_p}| \leq \text{OUT}$. Thus, we need a join circuit between two relations whose join size is bounded instead of the degrees.

6.3 Output-bounded Join

Let $R(A, B)$ and $S(B, C)$ be two relations with $|R| = M$, $|S| = N$ and $|R \bowtie S| \leq \text{OUT} \leq MN$. The output-bounded join circuit will output a relation of size exactly OUT (padding dummy tuples if necessary), while the circuit should have size $\tilde{O}(M + N + \text{OUT})$. Note that an output-bounded join is more general than

Algorithm 9: Yannakakis-C(Q, OUT)

```
1  $(\mathcal{T}, \chi, \{T_{\chi(v)}\}_{v \in \mathcal{V}_\mathcal{T}}) \leftarrow$ Reduce-C$(Q)$;
2  for $v \in \mathcal{V}_\mathcal{T}$ in bottom-up manner (root excluded) do
3      $p \leftarrow$ the parent of $v$;
4      $B \leftarrow \chi(v)$; $B_p \leftarrow \chi(p)$;
5      Update $T_{B_p} \leftarrow T_{B_p} \ltimes T_B$;
6  for $v \in \mathcal{V}_\mathcal{T}$ in top-down manner (leaves excluded) do
7      for $v_c \in$ the children of $v$ do in parallel
8          $B \leftarrow \chi(v)$; $B_c \leftarrow \chi(v_c)$;
9          Update $T_{B_c} \leftarrow T_{B_c} \ltimes T_B$;
10 for $v \in \mathcal{V}_\mathcal{T}$ in bottom-up manner (root excluded) do
11     $p \leftarrow$ the parent of $v$;
12     $B \leftarrow \chi(v)$; $B_p \leftarrow \chi(p)$;
13     $\mathrm{OUT}_t \leftarrow \min(\mathrm{OUT}, |T_B| \cdot |T_{B_p}|)$;
14     Compute $T_{B \cup B_p} \leftarrow T_{B_p} \bowtie T_B$ by a output-bounded join circuit with output size bound $\mathrm{OUT}_t$;
15     Remove $v$ from $\mathcal{T}$;
16     Update $\chi(p) \leftarrow B \cup B_p$;
17 Let $r$ be the root node of $\mathcal{T}$;
18 return $T_{\chi(r)}$;
```

a degree-bounded join studied in Section 5.4: if $\deg_S(B) \le d$, then $\mathrm{OUT} \le Md$, so an output-bounded join circuit with $\mathrm{OUT} = Md$ also works for a degree-bounded join.

Circuit Description The circuit is described in Algorithm 10. It uses the decomposition operator in PANDA-C, as well as the degree-bounded join in Section 5.4. In addition, it needs a truncation operator, which was introduced in Section 5.3.

Algorithm 10: Output-bounded Join Circuit

```
1  Decompose $S \rightarrow \cup_{i=1}^{k} S_i$ subject to (4);
2  for $i \leftarrow 1$ to $k$ do in parallel
3      Compute $R_i \leftarrow R \ltimes S_i$;
4      $N_i \leftarrow \lfloor \mathrm{OUT}/2^{i-1} \rfloor$;
5      Truncate $R_i$ to size $N_i$;
6      Compute $J_i(A, B, C) \leftarrow R_i \bowtie S_i$ subject to the degree constraint $\deg_{S_i}(B) \le 2^{i-1}$;
7  $J \leftarrow \cup_{i=1}^{k} J_i$;
8  Truncate $J$ to size OUT;
9  return $J$;
```

Correctness Proof to Algorithm 10. If we do not do the truncation in line 5, then it is obvious that J contains all join results because $\{S_i\}_{i=1}^k$ is a partitioning of S. Truncating J to size OUT in line 8 does not remove any join results as we are guaranteed that $|R \bowtie S| \le \mathrm{OUT}$.

Next we show that the truncation in line 5 does not remove non-dummy tuples from R_i, i.e., $|R_i| \le N_i \overset{\mathrm{def}}{=\!=} \lfloor \mathrm{OUT}/2^{i-1} \rfloor$. Note that $R_i = R \ltimes S_i$ is the set of tuples in R that can join with S_i, and $\deg_{S_i}(B) \ge 2^{i-1}$. Therefore, each tuple in R_i produces at least 2^{i-1} tuples in $R_i \bowtie S_i$, so we have $2^{i-1}|R_i| \le |R_i \bowtie S_i| \le |R \bowtie S| \le \mathrm{OUT}$, which implies $|R_i| \le \mathrm{OUT}/2^{i-1}$, or equivalently $|R_i| \le N_i$, as $|R_i|$ is an integer. □

Complexity Analysis The depth of the circuit is obviously $\tilde{O}(1)$, so we only analyze the size. Consider any $i \in [k]$. Since $|R| = M$, $|R_i| \le \mathrm{OUT}/2^{i-1}$, $|S_i| \le |S| = N$, and $\deg_{S_i}(B) \le 2^{i-1}$, the join circuit in line

9 has size $\tilde{O}(\mathrm{OUT}/2^{i-1} \cdot 2^{i-1} + N) = \tilde{O}(\mathrm{OUT} + N)$. Therefore, the total size of the output-bounded join circuit is $\tilde{O}(M + N + \mathrm{OUT})$, recalling that the decomposition produces $k = \tilde{O}(1)$ sub-relations.

6.4 Computing OUT

We can use a variant of Yannakakis-C to compute $\mathrm{OUT} = |Q(\mathbf{D})|$. We still first perform the reduce phase to remove all bound variables. Then we perform the bottom-up semijoin phase, except that in each semijoin $R_X \ltimes S_Y = R_X \bowtie \Pi_{X\cap Y}(S_Y)$, we replace the projection with a **sum** aggregation, and after the join, we need to multiply the sub-join sizes together. The circuit is described in Algorithm 11, in which we use a *map* operator $\rho_{\psi,F'}(R_F) = \{\psi(t) \mid t \in R_F\}$ for a function $\psi : \mathrm{dom}(F) \to \mathrm{dom}(F')$ that returns a relation with attribute set F'. This operator can be trivially implemented by a Boolean circuit of linear size and $\tilde{O}(1)$ depth.

Algorithm 11: Computing OUT

```
1  $(\mathcal{T}, \chi, \{T_{\chi(v)}\}_{v\in\mathcal{V}_\mathcal{T}}) \leftarrow$ Reduce-C$(Q)$;
2  for $v \in \mathcal{V}_\mathcal{T}$ do in parallel
3      Add an attribute $A_v$ to $T_{\chi(v)}$ with values set to 1;
4  for $v \in \mathcal{V}_\mathcal{T}$ in bottom-up manner (root excluded) do
5      $p \leftarrow$ the parent of $v$;
6      $B \leftarrow \chi(v)$; $B_p \leftarrow \chi(p)$;
7      $F \leftarrow B \cap B_p$;
8      $T_{B_pA_p} \leftarrow \rho_{\psi,B_pA_p}(T_{B_pA_p} \bowtie \Pi_{F,\mathbf{sum}(A_v)}(T_{BA_v}))$, where $\psi(t) = (t.B_p, t.A_p \cdot t.\mathbf{sum}(A_v))$;
9      Remove $v$ from $\mathcal{T}$;
10 Let $r$ be the root node of $\mathcal{T}$;
11 return $\Pi_{\emptyset,\mathbf{sum}(A_r)}(T_{\chi(r)})$;
```

Finally we conclude the following theorem.

Theorem 5. *For any CQ Q, there is a uniform family of Boolean circuits which evaluates $Q(\mathbf{D})$ for any $\mathbf{D}$ conforming to given degree constraints* DC *and output size* $\mathrm{OUT} = Q(\mathbf{D})$*. The circuits have $\tilde{O}(1)$ depth and $\tilde{O}(N + 2^{\mathsf{da\text{-}fhtw}(Q)} + \mathrm{OUT})$ size. Besides, the output size* OUT *can be computed by another uniform family of Boolean circuits given only degree constraints* DC*, which have $\tilde{O}(1)$ depth and $\tilde{O}(N + 2^{\mathsf{da\text{-}fhtw}(Q)})$ size.*

7 Extensions

Achieving Submodular Width All the development in this section use one GHD. Marx [27] observe that using multiple GHDs can further reduce the running time. PANDA can be used to compute disjunctive datalog rules. Together with multiple GHDs, this leads to a running time where the width is the *degree-aware submodular width* [3]:

$$\mathsf{da\text{-}subw}(Q) \overset{\text{def}}{=\!=} \max_{h\in\Gamma\cap\mathsf{HDC}} \min_{(\mathcal{T},\chi)} \max_{t\in\mathcal{V}_\mathcal{T}} h(\chi(t)),$$

which is always no larger than $\mathsf{da\text{-}fhtw}(Q)$. As PANDA-C is a relational circuit for PANDA, the results in Theorem 5 also hold even if $\mathsf{da\text{-}fhtw}(Q)$ is replaced by $\mathsf{da\text{-}subw}(Q)$.

Aggregation Queries Join-aggregate queries over semirings, i.e., AJAR/FAQ queries as defined in [23, 2], can also be handled by circuits: We first compute OUT using Algorithm 11. Then we invoke Yannakakis-C. To compute the aggregations correctly, we attach an annotation to each tuple. In Yannakakis-C, we replace each projection (including the one used in semijoins) by an $\oplus$-aggregation circuit; after each join, we use a map circuit to compute the $\otimes$-aggregation of the tuples comprising each join result. Here $\oplus$ and $\otimes$ are the semiring addition and multiplications. These additional circuits do not increase the overall depth/size by more than a constant factor. This way, the results in Theorem 5 also hold for join-aggregate queries. However, $\mathsf{da\text{-}fhtw}(Q)$ cannot be replaced by $\mathsf{da\text{-}subw}(Q)$ for join-aggregate queries, since using multiple GHDs cannot ensure that each annotation is aggregated only once. In fact, how to achieve the submodular width remains an open problem even in the RAM model [3].

Acknowledgments

This work has been supported by HKRGC under grants 16201318, 16201819, and 16205420. We thank the anonymous reviewers of PODS'22 for valuable suggestions on improving the presentation of the paper.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995. ISBN 0-201-53771-0. URL `http://webdam.inria.fr/Alice/`.

[2] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Faq: Questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, page 13–28, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341912. doi: 10.1145/2902251.2902280. URL `https://doi.org/10.1145/2902251.2902280`.

[3] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, page 429–444, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450341981. doi: 10.1145/3034786.3056105. URL `https://doi.org/10.1145/3034786.3056105`.

[4] M. Ajtai, J. Komlós, and E. Szemerédi. An o(n log n) sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 1–9, New York, NY, USA, 1983. Association for Computing Machinery. ISBN 0897910990. doi: 10.1145/800061.808726. URL `https://doi-org.lib.ezproxy.ust.hk/10.1145/800061.808726`.

[5] Arvind Arasu and Raghav Kaushik. Oblivious query processing. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 26–37. OpenProceedings.org, 2014. doi: 10.5441/002/icdt.2014.07. URL `https://doi.org/10.5441/002/icdt.2014.07`.

[6] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. Optorama: Optimal oblivious ram. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 403–432, Cham, 2020. Springer International Publishing.

[7] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '08, page 739–748, USA, 2008. IEEE Computer Society. ISBN 9780769534367. doi: 10.1109/FOCS.2008.43. URL `https://doi-org.lib.ezproxy.ust.hk/10.1109/FOCS.2008.43`.

[8] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic*, pages 208–222, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74915-8.

[9] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), page 307–314, New York, NY, USA, 1968. Association for Computing Machinery. ISBN 9781450378970. doi: 10.1145/1468075.1468121. URL `https://doi-org.lib.ezproxy.ust.hk/10.1145/1468075.1468121`.

[10] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. Smcql: Secure querying for federated databases. *Proc. VLDB Endow.*, 10(6):673–684, February 2017. ISSN 2150-8097. doi: 10.14778/3055330.3055334. URL `https://doi.org/10.14778/3055330.3055334`.

[11] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, page 1–10. Association for Computing Machinery, 1988.

[12] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, April 1974. ISSN 0004-5411. doi: 10.1145/321812.321815. URL `https://doi.org/10.1145/321812.321815`.

[13] Saba Eskandarian and Matei Zaharia. Oblidb: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, oct 2019. ISSN 2150-8097. doi: 10.14778/3364324.3364331. URL `https://doi.org/10.14778/3364324.3364331`.

[14] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

[15] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062. doi: 10.1145/1536414.1536440. URL `https://doi.org/10.1145/1536414.1536440`.

[16] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT'11, page 129–148, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 9783642204647.

[17] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996. ISSN 0004-5411. doi: 10.1145/233551.233553. URL `https://doi.org/10.1145/233551.233553`.

[18] Oded Goldreich, S. Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229. Association for Computing Machinery, 01 1987. ISBN 0-89791-221-7. doi: 10.1145/28395.28420.

[19] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[20] Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3), June 2012. ISSN 0004-5411. doi: 10.1145/2220357.2220363. URL `https://doi.org/10.1145/2220357.2220363`.

[21] W. Daniel Hillis and Guy L. Steele. Data parallel algorithms. *Commun. ACM*, 29(12):1170–1183, December 1986. ISSN 0001-0782. doi: 10.1145/7902.7903. URL `https://doi.org/10.1145/7902.7903`.

[22] Xiao Hu. Cover or pack: New upper and lower bounds for massively parallel joins. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '21, New York, NY, USA, 2021. Association for Computing Machinery.

[23] Manas R. Joglekar, Rohan Puttagunta, and Christopher Ré. Ajar: Aggregations and joins over annotated relations. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, page 91–106, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341912. doi: 10.1145/2902251.2902293. URL `https://doi.org/10.1145/2902251.2902293`.

[24] Bas Ketsman and Dan Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, page 417–428, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450341981. doi: 10.1145/3034786.3034788. URL `https://doi.org/10.1145/3034786.3034788`.

[25] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? *CoRR*, abs/1612.02503, 2016. URL `http://arxiv.org/abs/1612.02503`.

[26] Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, page 223–234, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306607. doi: 10.1145/1989284.1989310. URL `https://doi.org/10.1145/1989284.1989310`.

[27] Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6), November 2013. ISSN 0004-5411. doi: 10.1145/2535926. URL `https://doi.org/10.1145/2535926`.

[28] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3), March 2018. ISSN 0004-5411. doi: 10.1145/3180143. URL `https://doi.org/10.1145/3180143`.

[29] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[30] Yufei Tao. A Simple Parallel Algorithm for Natural Joins on Binary Relations. In Carsten Lutz and Jean Christoph Jung, editors, *23rd International Conference on Database Theory (ICDT 2020)*, volume 155 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-139-9. doi: 10.4230/LIPIcs.ICDT.2020.25. URL `https://drops.dagstuhl.de/opus/volltexte/2020/11949`.

[31] Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 96–106. OpenProceedings.org, 2014. doi: 10.5441/002/icdt.2014.13. URL `https://doi.org/10.5441/002/icdt.2014.13`.

[32] Yilei Wang and Ke Yi. Secure yannakakis: Join-aggregate queries over private data. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2021. Association for Computing Machinery.

[33] Rishabh Poddar Sukrit Kalra Avishay Yanai, Ryan Deng, and Raluca Ada Popa Joseph M Hellerstein. Senate: A maliciously-secure MPC platform for collaborative analytics. In *30th USENIX Security Symposium (USENIX Security 21)*, Vancouver, B.C., August 2021. USENIX Association. URL `https://www.usenix.org/conference/usenixsecurity21/presentation/poddar`.

[34] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, VLDB '81, page 82–94. VLDB Endowment, 1981.

[35] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, page 162–167, USA, 1986. IEEE Computer Society. ISBN 0818607408. doi: 10.1109/SFCS.1986.25. URL `https://doi.org/10.1109/SFCS.1986.25`.