Partition Constraints for Conjunctive Queries: Bounds and Worst-Case Optimal Joins

Kyle Deeds
University of Washington, Seattle, WA, USA

Timo Camillo Merkl
TU Wien, Austria

Abstract

In the last decade, various works have used statistics on relations to improve both the theory and practice of conjunctive query execution. Starting with the AGM bound which took advantage of relation sizes, later works incorporated statistics like functional dependencies and degree constraints. Each new statistic prompted work along two lines; bounding the size of conjunctive query outputs and worst-case optimal join algorithms. In this work, we continue in this vein by introducing a new statistic called a *partition constraint*. This statistic captures latent structure within relations by partitioning them into sub-relations which each have much tighter degree constraints. We show that this approach can both refine existing cardinality bounds and improve existing worst-case optimal join algorithms.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Worst-Case Optimal Joins, Cardinality Bounds, Degeneracy, Degree Constraints, Partition Constraints

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.17

Related Version *Full Version*: `https://arxiv.org/abs/2501.04190` [8]

Funding *Kyle Deeds*: Was supported by the NSF SHF 2312195 and NSF IIS 2314527 grants.
Timo Camillo Merkl: Was supported by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201, 10.47379/VRG18013].

Acknowledgements Part of this work was done when the authors were visiting the Simons Institute for the Theory of Computing.

1 Introduction

Efficient query execution is the cornerstone of modern database systems, where the speed at which information is retrieved often determines the effectiveness and user satisfaction of applications. In theoretical work, this problem is often restricted to the enumeration of conjunctive queries (CQs). One of the primary goals of database theory has been to classify the hardness of different queries and provide algorithms that enumerate them in optimal time. Most of these classical guarantees were described relative to the size, N, of the largest table in the database. In practical work, there has been a parallel effort to hone query optimizers which automatically generate an algorithm for query evaluation based on the particular properties of the data, finely tailoring the execution to the dataset at hand.

Recently, these lines of work have begun to meaningfully converge in the form of worst-case optimal join (WCOJ) algorithms. As a new paradigm in database theory, these methods provide data-dependent guarantees on query execution that take into account statistics $s(D)$ about the dataset being queried D. Concretely, WCOJ algorithms aim to only require time proportionate to (a bound on) the maximum join size over all databases D' with the same statistics as D, i.e., an optimal runtime on the *worst-case instance*. Crucially, the statistics that are used directly impact the kinds of guarantees that can be provided; more granular statistics allow for tighter worst-case analyses.

In a sequence of papers, increasingly more detailed statistics were incorporated into theoretical analyses. The foundational work in this direction by Grohe et al. used the size of each table in the database, often called *cardinality constraints* (CC)s, to produce bounds on the output size [2, 15]. This eventually led to a multitude of exciting WCOJ algorithms for CCs, e.g., Generic Join [22], Leapfrog Triejoin [24], and NPRR [21]. Instead of processing one join at a time, these algorithms processed one attribute at a time, asymptotically improving on traditional join plans. They have even seen significant uptake in the systems community with a variety of efficient implementations [1, 12, 25].

In later work, researchers incorporated functional dependencies (FDs) into cardinality bounds and then generalized CCs and FDs to *degree constraints* (DCs) [14, 18]. This work culminated in the development of the PANDA algorithm which leveraged information theory and proof-theoretic techniques to provide worst-case optimality relative to the polymatroid bound, modulo an additional poly-logarithmic factor [19].

A DC for a relation R on the attributes $\mathbf{Y}$ and a subset $\mathbf{X} \subseteq \mathbf{Y}$ asserts that for each fixed instantiation $\mathbf{x}$ of $\mathbf{X}$ there are only a limited number of completions to the whole set of attributes $\mathbf{Y}$. Thus, $\mathbf{X}$ functions like a weak version of a key. However, DCs are a brittle statistic; a single high frequency value per attribute can dramatically loosen a relation's DCs even if all other values only occur once. In the graph setting, where this corresponds to bounded (in- or out-)degree graphs, this has long been viewed as an overly restrictive condition because graphs often have highly skewed degree distributions. To overcome this, the graph theory community identified degeneracy as a natural graph invariant that allows for graphs of unbounded degree while permitting fast algorithms. For example, pattern counting on low degeneracy graphs has recently received considerable attention [3, 4, 5, 6, 7, 13].

In this work, we propose partition constraints (PCs), a declarative version of graph degeneracy for higher-arity data. PCs naturally extend DCs in the sense that every DC can be expressed as PC, but not the other way around. Informally, for a PC to hold over a relation R, we require it to be possible to split R such that each partition satisfies at least one DC. Again, for a binary edge relation $R = E$ of a directed graph $G = (V, E)$, this means that we can partition the edges into two sets E_1, E_2, such that the subgraph $G_1 := (V, E_1)$ has bounded out-degree while $G_2 := (V, E_2)$ has bounded in-degree.

We aim to provide a thorough analysis of the effect of PCs on conjunctive query answering. To that end, we show that PCs can provide asymptotic improvements to query bounds and evaluation, and we demonstrate how they can gracefully incorporate work on cardinality bounds and WCOJ algorithms for DCs. Further, we provide both exact and approximate algorithms for computing PCs and inspect to what extent PCs are present in well-known benchmark datasets.

Summary of results.

- We introduce PCs as a generalization of DCs and provide two algorithms to determine PCs in polynomial time as well as to partition the data to witness these constraints. One is exact and runs in quadratic time, while the second runs in linear time and provides a constant factor approximation.
- We develop new bounds on the output size of a join query. These bounds use DC-based bounds as a black box and naturally extend them to incorporate PCs. We show that these bounds are asymptotically tighter than bounds that rely on DCs alone. Further, we show that if the DC-based bounds are tight, then the PC-based bounds built on top of them will be tight as well.

- Using WCOJ algorithms for DCs as a black box, we provide improved join algorithms that are worst-case-optimal relative to the tighter PC-based bound. Notably, if an algorithm were proposed that is worst-case optimal relative to a tight DC-based bound, then this would immediately result in a WCOJ algorithm relative to a tight PC-based bound.

Structure of the paper. In Section 2, we provide some basic definitions and background. We formally introduce and discuss PCs in Section 3 and compute them on common benchmark data. In Section 4, we illustrate the benefits of the PC framework by thoroughly analyzing a concrete example. The developed techniques are then extended in Section 5 and applied to arbitrary CQs. We conclude and give some outlook to future work in Section 6. Full proofs of some results are deferred to the technical report [8], and we instead focus on providing intuition in the main body.

2 Preliminaries

Conjunctive queries. We assume the reader is familiar with relational algebra, in particular with joins, projections, and selection, which will respectively be denoted by $\bowtie, \pi$, and σ. In the context of the present paper, a *(conjunctive) query* (CQ), denoted using relational algebra, is an expression of the form

$$Q(\mathbf{Z}) \leftarrow R_1(\mathbf{Z}_1) \bowtie \cdots \bowtie R_k(\mathbf{Z}_k).$$

In this expression, each $R_i(\mathbf{Z}_i)$ is a relation over the set of variables $\mathbf{Z}_i$, and $Q(\mathbf{Z})$ is the output of the query where $\mathbf{Z} = \bigcup_i \mathbf{Z}_i$. Thus, we only consider *full conjunctive query*. When clear from the context, we omit the reference to the set of variables and simply write R_i and Q. We also use Q to refer to the whole CQ. A database instance I for Q is comprised of a concrete instance for each R_i, i.e., a set of tuples (also denoted by R_i) where each tuple $\mathbf{z}_i$ contains a value for each variable in $\mathbf{Z}_i$. The set of values appearing in I is referred to as dom, the domain of I. Furthermore, let $dom(Z)$ be the values assigned to $Z \in \mathbf{Z}$ by some relation R_i and, for $\mathbf{Y} \subseteq \mathbf{Z}$, $dom(\mathbf{Y}) := \times_{Y \in \mathbf{Y}} dom(Y)$. For a particular database instance I, we denote the answers to a CQ, i.e., the join of the R_i, as a relation $Q^I(\mathbf{Z})$. Lastly, a join algorithm $\mathcal{A}$ is any algorithm that receives a query Q and a database instance I as input and outputs the relation $Q^I(\mathbf{Z})$.

Degree constraints and bounds. In recent years, there has been a series of novel approaches to bound the size of $Q^I(\mathbf{Z})$ based on the structure of the query Q and a set of statistics about the database instance I. For full conjunctive queries, this recent round of work began with the AGM bound [2]. This bound takes the size of each input relation as the statistics and connects the size of the result to a weighted edge covering of the hypergraph induced by Q. Later bounds extended this approach by considering more complex statistics about the input relations. Functional dependencies were investigated first in [14]. These were then generalized to degree constraints (DCs) in [18] and degree sequences (DSs) in [9, 10, 17]. Our work in this paper continues in this vein by generalizing degree constraints further to account for the benefits of partitioning relations. So, we begin by defining degree constraints below.

▶ **Definition 1.** *Fix a particular relation $R(\mathbf{Z})$. Given two sets of variables $\mathbf{X}, \mathbf{Y}$ where $\mathbf{X} \subseteq \mathbf{Y} \subseteq \mathbf{Z}$, a degree constraint of the form $DC_R(\mathbf{X}, \mathbf{Y}, d)$ implies the following,*

$$\max_{\mathbf{x} \in dom(\mathbf{X})} |\pi_{\mathbf{Y}} \sigma_{\mathbf{X}=\mathbf{x}} R| \leq d.$$

A database instance I satisfying a (set of) degree constraints $\mathbf{DC}$ *is denoted by* $I \vDash \mathbf{DC}$. *For convenience, we denote the minimal d such that $DC_R(\mathbf{X}, \mathbf{Y}, d)$ holds by $DC_R(\mathbf{X}, \mathbf{Y})$. If $\mathbf{Y} = \mathbf{Z}$ and $\mathbf{X} = \emptyset$ the constraint simply bounds the cardinality of R and we write $CC_R(d)$. If clear from the context, we may omit R.*

In addition to generalizing DCs, our work is able to refine any of the previous bounding methods for DCs by incorporating them into our framework. Therefore, we introduce a general notation for DC-based bounds.

▶ **Definition 2.** *Given a conjunctive query Q and a set of degree constraints* $\mathbf{DC}$*, a cardinality bound $CB(Q, \mathbf{DC})$ is any function where the following is true,*

$$|Q^I(\mathbf{Z})| \leq CB(Q, \mathbf{DC}) \quad \forall\, I \vDash \mathbf{DC}.$$

Throughout this work, we will make specific reference to the combinatorics bound which we describe below. While it is impractical, this bound is computable and tight which makes it useful for theoretical analyses.

▶ **Definition 3.** *Given a conjunctive query Q and a set of degree constraints* $\mathbf{DC}$*, we define the combinatorics bound (for degree constraints) as*

$$CB_{Comb}(Q, \mathbf{DC}) = \sup_{I \vDash \mathbf{DC}} |Q^I(\mathbf{Z})|.$$

In this work, we make two minimal assumptions about bounds and statistics. First, we assume that cardinality bounds are finite by assuming that every variable in the query is covered by at least one cardinality constraint. Second, we assume that there is a bound on the growth of the bounds as our statistics increase. Formally, for a fixed query Q, we assume that every cardinality bound CB has a function f_Q such that for any $\alpha \in \mathbb{R}^+$ and degree constraints $\mathbf{DC}$ we have $CB(Q, \alpha \cdot \mathbf{DC}) \leq f_Q(\alpha) \cdot CB(Q, \mathbf{DC})$. Usually, cardinality bounds are expressed in terms of power products of the degree constraints [2, 14, 17]. In that case, $f_Q = O(\alpha^c)$ for some constant c.

Lastly, we will also incorporate existing work on worst-case optimal join (WCOJ) algorithms. Each WCOJ algorithm is optimal relative to a particular cardinality bound, and we describe that relationship formally as follows:

▶ **Definition 4.** *Denote the runtime of a join algorithm $\mathcal{A}$ on a conjunctive query Q and database instance I as $\mathcal{T}(\mathcal{A}, Q, I)$. $\mathcal{A}$ is worst-case optimal (WCO) relative to a cardinality bound $CB_{\mathcal{A}}$ if the following is true for all queries Q,*

$$\mathcal{T}(\mathcal{A}, Q, I) = O(|I| + CB_{\mathcal{A}}(Q, \mathbf{DC})), \quad \forall\, \mathbf{DC},\ I \vDash \mathbf{DC}.$$

Note that the hidden constant may only depend on the query Q and, importantly, neither on the set of degree constraints $\mathbf{DC}$ *nor on the database instance I. When $\mathcal{A}$ is optimal relative to CB_{Comb}, we simply call it worst-case optimal (relative to degree constraints).*

Note that some algorithms fulfill a slightly looser definition of worst-case optimal by allowing additional poly-logarithmic factors [19]. These algorithms can still be incorporated into this framework, but we choose the stricter definition to emphasize that we do not induce additional factors of this sort.

Much of the recent work on WCOJ algorithms can be categorized as *variable-at-a-time* (VAAT) algorithms. Intuitively, a VAAT computes the answers to a join query by answering the query for an increasingly large number of variables. This idea is at the heart of Generic Join [22], Leapfrog Triejoin [24], and NPRR [21]. We define this category formally as follows. For an arbitrary query $Q(\mathbf{Z}) \leftarrow R_1 \bowtie \cdots \bowtie R_k$, let Q_i denote the sub-query

$$Q_i(Z_1, \dots Z_i) \leftarrow \pi_{Z_1,\dots,Z_i} R_1 \bowtie \cdots \bowtie \pi_{Z_1,\dots,Z_i} R_k,$$

for an ordering $Z_1, \dots, Z_r = \mathbf{Z}$.

▶ **Definition 5.** *A join algorithm $\mathcal{A}$ that solves conjunctive queries $Q(\mathbf{Z}) \leftarrow R_1(\mathbf{Z}_1) \bowtie \cdots \bowtie R_k(\mathbf{Z}_k)$ is a VAAT algorithm if there is some ordering $Z_1, \dots, Z_r = \mathbf{Z}$ such that $\mathcal{A}$ takes time $\Omega(\max_i |Q_i^I|)$ for databases I. The choice of the ordering is allowed to depend on I.*

3 Partition Constraints

Partition constraints (PCs) extend the concept of DCs by allowing for the partitioning of relations. For clarity, we start with the binary setting, revisiting the example from the introduction. To that end, let $E(X, Y)$ be the edge relation of a directed graph $G = (V, E)$. For a degree constraint to hold on E, either the in- or the out-degree of E must be bounded. However, this is a strong requirement and often may not be satisfied, e.g. on a highly skewed social network graph. In these cases, it makes sense to look for further latent structure in the data. We suggest partitioning E such that each part satisfies a (different) degree constraint. We are interested in the following quantity where the minimum is taken over all bi-partitions of E, i.e. $E = E^X \cup E^Y$ (implicitly $E^X \cap E^Y = \emptyset$),

$$\min_{E^X \cup E^Y = E} \max\{DC_{E^X}(X, XY), DC_{E^Y}(Y, XY)\}. \tag{1}$$

This corresponds to a splitting of the graph into two graphs. In one of them, $G^X = (V, E^X)$, we attempt to minimize the maximum out-degree of the graph while in the other, $G^Y = (V, E^Y)$, we attempt to minimize the maximum in-degree. It is important to note that both of these quantities can be arbitrarily lower than the maximum in-degree and out-degree of the original graph. This is where the benefit of considering partitions comes from.

An example of such a partitioning is depicted in Figure 1. There, the dashed blue edges represent E^X while the solid red edges represent E^Y. The maximum out- and in-degree of the full graph is 5 while $G^X = (V, E^X)$ has a maximum out-degree of 1 and $G^Y = (V, E^Y)$ has a maximum in-degree of 1. In general, a class of graphs can have an unbounded out- and in-degree but always admit a bi-partitioning with an out- and in-degree of 1, respectively.

This approach is originally motivated by the graph property *degeneracy*. Intuitively, this property tries to allocate each edge of an undirected graph to one of its incident vertices such that no vertex is "responsible" for too many edges. Each partition $E = E^X \cup E^Y$ can be seen as a possible allocation where the edges in E^X are allocated to the X part of the tuples while the edges in E^Y are allocated to the Y part of the tuples. Thus, in general, if the undirected version of a graph class $\mathcal{G}$ has bounded degeneracy, the Quantity 1 must also be bounded for $\mathcal{G}$. In fact, the converse is true as well if the domain of the X and Y attributes are disjoint.

Formally, we define a PC on a relation R as below. Note, we say a collection of subrelations $(R^1, \dots, R^k)$ partition R when they are pairwise disjoint and $\bigcup_j R^j = R$.

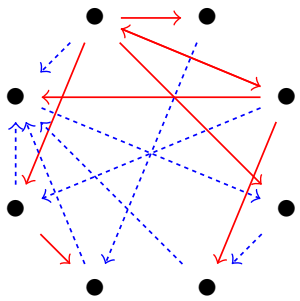

Figure 1 Graph Example.

Access

PersonID	RoomID
Ava	Beacon Hall
Ben	Beacon Hall
Cole	Delta Hall
Dan	Delta Hall
Emma	Gala Hall
Finn	Jade Hall
Porter	Beacon Hall
Porter	Delta Hall
Porter	Gala Hall
Porter	Jade Hall

$\texttt{Access}^{\texttt{PersonID}}$

PersonID	RoomID
Ava	Beacon Hall
Ben	Beacon Hall
Cole	Delta Hall
Dan	Delta Hall
Emma	Gala Hall
Finn	Jade Hall

$\texttt{Access}^{\texttt{RoomID}}$

PersonID	RoomID
Porter	Beacon Hall
Porter	Delta Hall
Porter	Gala Hall
Porter	Jade Hall

Figure 2 Access Example.

▶ **Definition 6.** *Fix a particular relation $R(\mathbf{Z})$, a subset $\mathbf{Y} \subseteq \mathbf{Z}$, and let $\mathcal{X} = \{\mathbf{X}^1, ..., \mathbf{X}^{|\mathcal{X}|}\} \subseteq 2^{\mathbf{Y}}$ be a set of sets of variables. Then, a partition constraint of the form $PC_R(\mathcal{X}, \mathbf{Y}, d)$ implies,*

$$\min_{(R^{\mathbf{X}})_{\mathbf{X}\in\mathcal{X}} \text{ partition } R} \max\{DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y}) \mid \mathbf{X} \in \mathcal{X}\} \leq d.$$

Again, a database instance I satisfying a (set of) partition constraints $\mathbf{PC}$ is denoted by $I \vDash \mathbf{PC}$. We denote the minimal d such that $PC_R(\mathcal{X}, \mathbf{Y}, d)$ holds by $PC_R(\mathcal{X}, \mathbf{Y})$ and we omit R if clear from the context.

This definition says that one can split the relation R into disjoint subsets $R^j \subseteq R, \bigcup_j R^j = R$ and associate each R^j with a degree constraint over a set of variables $\mathbf{X}^j \in \mathcal{X}$ such that the maximum of these constraints is then bounded by d. From an algorithmic point of view, each part R^j should be handled differently to make use of its unique, tighter DC. The core of this work is in describing how these partitions can be computed and how to make use of the new constraints on each part. Broadly, we show how these new constraints produce tighter bounds on the join size and how algorithms can meet these bounds. Note PCs are a strict generalization of DCs since we can define an arbitrary degree constraint $DC(\mathbf{X}, \mathbf{Y}, d)$ as $PC(\{\mathbf{X}\}, \mathbf{Y}, d)$. For simplicity, when we discuss a collection of PCs, we assume that there is at most one PCs per $(\mathcal{X}, \mathbf{Y})$ pair, i.e., there are never two $PC_R(\mathcal{X}, \mathbf{Y}, d), PC_R(\mathcal{X}, \mathbf{Y}, d'), d \neq d'$, as it suffices to keep the stronger constraint.

Next, we present an example of how relations with small PCs might arise in applications.

▶ **Example 7.** Consider a relation `Access(PersonID, RoomID)` that records who has access to which room at a university. This could be used to control the key card access of all faculty members, students, security, and cleaning personnel. Most people (faculty members and students) only need access to a limited number of rooms (lecture halls and offices). On the other hand, porters and other caretaker personnel need access to many different rooms, possibly all of them. However, each room only needs a small number of people taking care of it. Thus, it makes sense to partition `Access` into $\texttt{Access}^{\texttt{PersonID}} \cup \texttt{Access}^{\texttt{RoomID}}$ with the former tracking the access restrictions of the faculty members and students, and the latter tracking the access restrictions of the caretaker personnel. With this partition, both $DC_{\texttt{Access}^{\texttt{PersonID}}}(\texttt{PersonID}, \texttt{RoomID})$ and $DC_{\texttt{Access}^{\texttt{RoomID}}}(\texttt{RoomID}, \texttt{PersonID})$ should be small. Thus, $PC_{\texttt{Access}}(\{\texttt{PersonID}, \texttt{RoomID}\}, \texttt{PersonID RoomID})$ should be small as well.

Dataset	Max DC	Min DC	PC	$\lvert\mathcal{X}\rvert$
aids	11	11	3	2
yeast	154	119	9	2
dblp	321	113	34	2
wordnet	526	284	3	2
Stats/badges	899	456	8	2
Stats/comments	134887	45	15	3
Stats/post_links	10186	13	2	4
Stats/post_history	91976	32	3	4
Stats/votes	326320	427	33	4
IMDB/keywords	72496	540	71	2
IMDB/companies	1334883	94	13	4
IMDB/info	13398741	2937	123	4
IMDB/cast	25459763	1741	52	6

Figure 3 Example PCs.

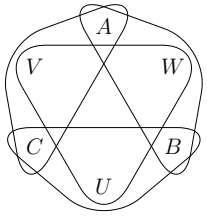

Figure 4 The Query $Q_{\bigcirc}$.

Figure 2 depicts a small example instance of the situation. There, the students each only need access to a single lecture hall to attend their courses, and all the rooms are taken care of by a single porter. Thus, following the suggested splitting, the respective DC for both subrelations $\texttt{Access}^{\texttt{PersonID}}$ and $\texttt{Access}^{\texttt{RoomID}}$ is 1 and, therefore, also the PC for the whole relation `Access` is 1.

To see how these statistics manifest in real world data, we calculated the PC of relations from some standard benchmarks which are displayed in Figure 3 [23, 16, 20]. We computed the PC using Algorithm 3 from Section 5. To model the interesting case of many-to-many joins, we first removed any attributes which are primary keys from each relation. Specifically, we computed the partition constraint $PC(\{X \mid X \in \mathbf{Y}\}, \mathbf{Y})$ where $\mathbf{Y}$ is the set of non-key attributes. We compare this with the minimum and maximum degree of these attributes before partitioning. Naively, by partitioning the data randomly, one would expect a PC roughly equal to the maximum DC divided by $|\mathcal{X}|$. Alternatively, by placing all tuples in the partition corresponding to the minimum DC, one can achieve a PC equal to the minimum DC. However, the computed PC is often much lower than both of these quantities. This implies that the partitioning is uncovering useful structure in the data rather than simply distributing high-degree values over multiple partitions.

3.1 Further Partitioning

At this point, one might wonder if further partitioning the data can meaningfully reduce a PC. That is, whether for a given relation $R(\mathbf{Z})$ and a particular set of degree constraints $\{DC_R(\mathbf{X}, \mathbf{Y}) \mid \mathbf{X} \in \mathcal{X}\}$, is it possible to decrease the maximum DC significantly by partitioning R into more than $|\mathcal{X}|$ parts? We show that this is not the case; neither pre-partitioning nor post-partitioning the data into k parts can reduce the PC by more than a factor of k. We prove the former first.

▶ **Proposition 8.** *Given a relation $R(\mathbf{Z})$, a subset $\mathbf{Y} \subseteq \mathbf{Z}$, variable sets $\mathcal{X} \subseteq 2^{\mathbf{Y}}$, and subrelations $(R^1, \ldots, R^k)$ that partition R. Then,*

$$\max_{i=1,\ldots,k} PC_{R^i}(\mathcal{X}, \mathbf{Y}) \geq PC_R(\mathcal{X}, \mathbf{Y})/k.$$

Proof. For the sake of contradiction, we assume that there exists a partitioning $(R^1, \ldots, R^k)$ of R such that,

$$\max_{i=1,\ldots,k} PC_{R^i}(\mathcal{X}, \mathbf{Y}) < PC_R(\mathcal{X}, \mathbf{Y})/k.$$

Then, we can partition each R^i to witness $PC_{R^i}(\mathcal{X}, \mathbf{Y})$. Let $\bigcup_{\mathbf{X}\in\mathcal{X}} R^{i,\mathbf{X}} = R^i$ be such that $DC_{R^{i,\mathbf{X}}}(\mathbf{X}, \mathbf{Y}) \leq PC_{R^i}(\mathcal{X}, \mathbf{Y})$ holds for each part $R^{i,\mathbf{X}}$. For each fixed $\mathbf{X} \in \mathcal{X}$, we can combine the sub-relations $R^{1,\mathbf{X}}, \ldots, R^{k,\mathbf{X}}$ into a single relation $R^{\mathbf{X}}$. These relations, $(R^{\mathbf{X}})_{\mathbf{X}\in\mathcal{X}}$, form a partition of R. The DC for each $R^{\mathbf{X}}$ is at most the sum of the DCs of $R^{1,\mathbf{X}}, \ldots, R^{k,\mathbf{X}}$. Further, this sum must be less than our initial PC by our assumption,

$$DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y}) \leq k \cdot PC_{R^i}(\mathcal{X}, \mathbf{Y}) < PC_R(\mathcal{X}, \mathbf{Y}).$$

This directly implies that

$$\max_{\mathbf{X}\in\mathcal{X}} DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y}) < PC_R(\mathcal{X}, \mathbf{Y}).$$

Because the PC is defined as the minimum value of this maximum DC over all possible partitions of R into $|\mathcal{X}|$ parts, this is a contradiction. ◀

Next, we show that post-partitioning cannot super-linearly reduce the degree either.

▶ **Proposition 9.** *Let $(R^1, \ldots, R^k)$ partition a relation $R(\mathbf{Z})$, and let $\mathbf{X} \subseteq \mathbf{Y} \subseteq \mathbf{Z}$ be two subsets of variables. Then,*

$$\max_{i=1,\ldots,k} DC_{R^i}(\mathbf{X}, \mathbf{Y}) \geq DC_R(\mathbf{X}, \mathbf{Y})/k.$$

Proof. Let $\mathbf{x}$ be the value of $\mathbf{X}$ within R that occurs in tuples with $d = DC_R(\mathbf{X}, \mathbf{Y})$ unique values $\mathbf{y}$ of $\mathbf{Y}$. For each of these values $\mathbf{y}$, a tuple containing $\mathbf{y}$ must be associated with one partition R^i. By the pigeonhole principle and the fact that there are $DC_R(\mathbf{X}, \mathbf{Y})$ of these tuples, it follows that some partition must contain at least $DC_R(\mathbf{X}, \mathbf{Y})/k$ of these tuples. Therefore, for some $i \in \{1, \ldots, k\}$, we have $DC_{P_i}(\mathbf{X}, \mathbf{Y}) \geq DC_R(\mathbf{X}, \mathbf{Y})/k$. ◀

Theorems 8 and 9 together show that for a given relation $R(\mathbf{Y})$ and a particular set of degree constraints $\{DC_R(\mathbf{X}, \mathbf{Z}) \mid \mathbf{X} \in \mathcal{X}\}$ deemed relevant, we only have to consider partitionings of R into at most $|\mathcal{X}|$ pieces. Thus, if the query size is viewed as a constant, then the number of useful partitions is also a constant.

4 The Hexagon Query

In this section, we show the benefits of the PC framework by demonstrating how it can lead to asymptotic improvements for both cardinality bounds and conjunctive query evaluation. Concretely, there is an example query and class of database instances where bounds based on **PC** are asymptotically tighter than those based on **DC**. Further, all VAAT algorithms (See Definition 5) are asymptotically slower than a **PC**-aware algorithm on this query and instance class. Formally, our aim is to show the following:

▶ **Theorem 10.** *There exists a query Q and a set of partition constraints $\mathbf{PC}$ with the degree constraint subset $\mathbf{DC} \subset \mathbf{PC}$ such that the following are true:*

1. *Bounds based on degree constraints are asymptotically sub-optimal, i.e.*

$$\sup_{I\vDash\mathbf{PC}} |Q^I(\mathbf{Z})| = o(CB_{Comb}(Q, \mathbf{DC})) = o(\sup_{I\vDash\mathbf{DC}} |Q^I(\mathbf{Z})|).$$

2. *There is an algorithm that enumerates* Q^I *for instances* $I \models \mathbf{PC}$ *in time* $O(\sup_{I \models \mathbf{PC}} |Q^I(\mathbf{Z})|)$.
3. *No VAAT algorithms can enumerate* Q^I *for instances* $I \models \mathbf{PC}$ *in time* $O(\sup_{I \models \mathbf{PC}} |Q^I(\mathbf{Z})|)$.

To prove these claims, we consider the hexagon query (also depicted in Figure 4)

$$Q_{\hexagon}(A, B, C, U, V, W) \leftarrow R_1(A, W, B) \bowtie R_2(B, U, C) \bowtie R_3(C, V, A) \bowtie R_4(U, V, W)$$

and impose the following set of PCs on the relations:

$$\begin{aligned}
&DC_{R_1}(AW, AWB, 1), \quad DC_{R_1}(WB, AWB, 1), \quad CC_{R_1}(n), \quad CC_{R_2}(n)\\
&DC_{R_2}(BU, BUC, 1), \quad DC_{R_2}(UC, BUC, 1), \quad CC_{R_3}(n), \quad CC_{R_4}(n),\\
&DC_{R_3}(CV, CVA, 1), \quad DC_{R_3}(VA, CVA, 1), \quad PC_{R_4}(\{U, V, W\}, UVW, 1).
\end{aligned}$$

We denote the whole set as **PC** and the subset of DCs as **DC**. We now prove the theorem's first claim. That is, the combinatorics bound on $Q_{\hexagon}$ is super linear when only considering **DC**, while there are only a linear number of answers to $Q_{\hexagon}$ over any database satisfying **PC**. We begin by providing a lower bound on the combinatorics bound of $Q_{\hexagon}$.

▶ **Lemma 11.** *The combinatorics bound of* $Q_{\hexagon}$ *based on* **DC** *is in* $\Omega(n^{\frac{4}{3}})$.

Proof Sketch. It suffices to provide a collection of databases $\mathcal{I}$ such that $I \models \mathbf{DC}$ and $|Q^I_{\hexagon}| = \Omega(n^{\frac{4}{3}})$ for $I \in \mathcal{I}$. To accomplish this, we introduce a new relation $R_{X,Y,Z}(X, Y, Z)$ with $|dom(X)| = |dom(Z)| = n^{\frac{2}{3}}$ and $|dom(Y)| = n^{\frac{1}{3}}$. Intuitively, think of $R_{X,Y,Z}$ as a bipartite graph from the domain of X to the domain of Z where Y identifies the edge for a given $x \in dom(X)$ or $z \in dom(Z)$. Thus, every $x \in dom(X)$ is connected to $n^{\frac{1}{3}}$ neighbors in $dom(Z)$ and, due to symmetry, also the other way around.

Therefore, the constraints $DC(XY, XYZ, 1)$, $DC(YZ, XYZ, 1)$, and $CC(n)$ are satisfied over $R_{X,Y,Z}$. Thus, we can use a relation of this type for R_1, R_2, R_3. Concretely, we use this relation in the following way: $R_1 = R_{A,W,B}, R_2 = R_{B,U,C}, R_3 = R_{C,V,A}$. Thus, for each ($n^{\frac{2}{3}}$ many) $a \in dom(A)$ there are $n^{\frac{1}{3}}$ many matching $b \in dom(B)$ and possibly up to $n^{\frac{1}{3}}$ many matching $c \in dom(C)$. Thus, in total, there may be up to $n^{\frac{4}{3}}$ answers to $R_1 \bowtie R_2 \bowtie R_3$. To accomplish this also for $Q_{\hexagon}$, we only have to make sure that the variables U, V, W also join in R_4. For that, we simply set $R_4 = dom(U) \times dom(V) \times dom(W)$. This relation then satisfies its cardinality constraint. (Note that it does not satisfy its PC.) ◀

We now provide an algorithm (Algorithm 1) that enumerates $Q_{\hexagon}$ in linear time for databases with the PC on R_4. This proves that the output size is linear, simultaneously completing our proof of claim 1 and claim 2. Algorithm 1 first decomposes R_4 into three parts with one DC on each part. For this, we apply a linear time greedy partitioning algorithm (for more details see Algorithm 2) and show that it results in partitions whose constraints are within a factor of 3 from the optimal **PC**. Then, for each part, a variable order is selected to take advantage of all DCs. As an example, for the part $R_4^W(U, V, W)$, there are at most 3 matching $(u, v) \in dom(U, V)$ pair for each value of $w \in dom(W)$. Thus, starting with a tuple (a, w, b) of R_1, we can use this fact to determine these values for u, v and then combine this with $DC_{R_2}(BU, BUC, 1)$ to determine the unique value for c. By this reasoning, all of the inner for-loops iterate over a single element or 3 pairs of elements. Thus, the nested loops are linear in their total runtime. We defer the formal proof to the technical report [8].

▶ **Lemma 12.** *Algorithm 1 enumerates* $Q^I_{\hexagon}$ *in time* $O(n)$ *for databases* $I \models \mathbf{PC}$.

Algorithm 1 Linear Hexagon Algorithm.

$R_4^U, R_4^V, R_4^W \leftarrow$ **decompose**$(R_4, \{U, V, W\}, UVW)$
▷ $DC_{R_4^U}(U, UVW, 3), DC_{R_4^V}(V, UVW, 3), DC_{R_4^W}(W, UVW, 3)$
for $(a, w, b) \in R_1$ **do**
  for $(u, v) \in \pi_{U,V}\sigma_{W=w}R_4^W$ **do**
    for $c \in (\pi_C\sigma_{B=b\wedge U=u}R_2 \cap \pi_C\sigma_{A=a\wedge V=v}R_3)$ **do**
      output (a, b, c, u, v, w)
for $(b, u, c) \in R_2$ **do**
  for $(v, w) \in \pi_{V,W}\sigma_{U=u}R_4^U$ **do**
    for $a \in (\pi_A\sigma_{B=b\wedge W=w}R_1 \cap \pi_A\sigma_{C=c\wedge V=v}R_3)$ **do**
      output (a, b, c, u, v, w)
for $(c, v, a) \in R_3$ **do**
  for $(u, w) \in \pi_{U,W}\sigma_{V=v}R_4^V$ **do**
    for $b \in (\pi_B\sigma_{A=a\wedge W=w}R_1 \cap \pi_B\sigma_{C=c\wedge U=u}R_2)$ **do**
      output (a, b, c, u, v, w)

Lemma 11 and Lemma 12 together prove the first two claims of Theorem 10. This shows that PCs have an asymptotic effect on query bounds and that we can take advantage of PCs to design new WCOJ algorithms to meet these bounds. Nevertheless, one might wonder whether the variable elimination idea of established WCOJ algorithms can already meet the improved bound and, in fact, achieve optimal runtimes. Proving the third claim in Theorem 10, we show that they cannot and, thus, the new techniques have to be employed. Specifically, we show that VAAT algorithms (Definition 5) require time $\Omega(n^{1.5})$ to compute $Q_{\hexagon}$ on database instances satisfying the PCs.

▶ **Lemma 13.** *VAAT algorithms require time $\Omega(n^{1.5})$ to enumerate $Q_{\hexagon}^I$ for databases $I \vDash \mathbf{PC}$.*

Proof Sketch. It suffices to provide a collection of databases $\mathcal{I}$ such that $I \vDash \mathbf{PC}$ and $\max_i |Q_{\hexagon_i}^I| = \Omega(n^{1.5})$ for databases $I \in \mathcal{I}$ and arbitrary ordering of the variables. To that end, we introduce two relations, a relation $C_{X,Y,Z}(X, Y, Z)$ and a set of disjoint paths $P_{X,Y,Z}(X, Y, Z)$. For $P_{X,Y,Z}(X, Y, Z)$, the domains of X, Y, Z are of size $\Theta(n)$ and $P_{X,Y,Z}$ simply matches X to Y and Z such that each $d \in dom(X) \cup dom(Y) \cup dom(Z)$ appears in exactly one tuple of $P_{X,Y,Z}$. On the other hand, think of $C_{X,Y,Z}$ as a complete bipartite graph from the domain of X to the domain of Y and Z uniquely identifies the edges. Thus, $|dom(X)| = |dom(Y)| = \Theta(\sqrt{n})$ while $|dom(Z)| = \Theta(n)$. Notice that for both relations, $DC(XY, XYZ, 1), DC(Z, XYZ, 1)$ hold. I.e., any pair of variables determine the last variable and there is a variable that determines the whole tuple on its own.

Consequently, the disjoint union of relations C and P multiple times (with permutated versions of C), e.g.,

$$R(X, Y, Z) = C_{X,Y,Z}(X, Y, Z) \cup C_{Y,Z,X}(Y, Z, X) \cup P_{X,Y,Z}(X, Y, Z),$$

satisfies $PC_R(\{X, Y, Z\}, XYZ, 1)$ and $DC(S_1S_2, XYZ, 1)$ for any $S_1S_2 \subseteq XYZ$.

Thus, we can set R_1, R_2, R_3, R_4 to be the disjoint union of P and all permutations of the relation C. Now, let $X_1, \ldots, X_6$ be an arbitrary variable order for $Q_{\hexagon}$. Then, a VAAT algorithm based on this variable order at least needs to compute the sets:

$$\ltimes_i \pi_{X_1}R_i, \quad \ltimes_i \pi_{X_1X_2}R_i, \quad \ltimes_i \pi_{X_1\cdots X_3}R_i, \quad \ltimes_i \pi_{X_1\cdots X_4}R_i, \quad \ltimes_i \pi_{X_1\cdots X_5}R_i, \quad \ltimes_i \pi_{X_1\cdots X_6}R_i.$$

Let us consider the set $\ltimes_i \pi_{X_1\cdots X_4}R_i$. There are two cases:

Case 1: X_5 and X_6 appear conjointly in a relation. Due to the symmetry of the query and the database, we can assume w.l.o.g, $X_5X_6 = UV$ and $\bowtie_i \pi_{X_1 \cdots X_4} R_i = \bowtie_i \pi_{ABCW} R_i$. Furthermore, $\pi_{ABW} C_{A,B,W} \subseteq \pi_{ABW} R_1$, $\pi_{BC} C_{B,C,U} \subseteq \pi_{BC} R_2$, $\pi_{AC} C_{A,C,V} \subseteq \pi_{AC} R_3, \pi_W P_{U,V,W} \subseteq \pi_W R_4$. Thus, intuitively, for at least $\Omega(\sqrt{n})$ elements $a \in dom(A)$ there are $\Omega(\sqrt{n})$ elements $b \in dom(B)$ and $\Omega(\sqrt{n})$ elements $c \in dom(C)$ that all join, and for each pair a, b there is an element $w \in dom(W)$ that fits. In total, $|\bowtie_i \pi_{ABCW} R_i| = \Omega(n^{1.5})$.

Case 2: X_5 and X_6 do not appear conjointly in a relation. Due to the symmetry of the query and the database, we can assume w.l.o.g, $X_5X_6 = AU$ and $\bowtie_i \pi_{X_1 \cdots X_4} R_i = \bowtie_i \pi_{BCVW} R_i$. Furthermore, $\pi_{BW} C_{B,W,A} \subseteq \pi_{BW} R_1$, $\pi_{BC} C_{B,C,U} \subseteq \pi_{BC} R_2$, $\pi_{CV} C_{C,V,A} \subseteq \pi_{CV} R_3, \pi_{VW} C_{V,W,U} \subseteq \pi_{VW} R_4$. Thus, intuitively, for at least $\Omega(\sqrt{n})$ elements $b \in dom(B)$ there are $\Omega(\sqrt{n})$ elements $w \in dom(W)$, $\Omega(\sqrt{n})$ elements $v \in dom(V)$ and, $\Omega(\sqrt{n})$ elements $c \in dom(C)$ that all join. In total, $|\bowtie_i \pi_{BCVW} R_i| = \Omega(n^2)$. ◀

5 Partition Constraints for General Conjunctive Queries

In the following, we extend the ideas of Section 4 to an arbitrary full conjunctive queries $Q(\mathbf{Z}) \leftarrow R_1(\mathbf{Z}_1) \bowtie \ldots \bowtie R_k(\mathbf{Z}_k)$ and an arbitrary set of partition constraints $\mathbf{PC} = \{PC_{P_1}(\mathcal{X}_1, \mathbf{Y}_1, d_1), \ldots, PC_{P_l}(\mathcal{X}_l, \mathbf{Y}_l, d_l)\}$. Recall, Algorithm 1 proceeds by decomposing R_4 in accordance with the PC and then, executes a VAAT style WCOJ algorithm over the decomposed instances. We proceed in the same way and start by concentrating on decomposing relations. After partitioning the relations, we show how to lift DC-based techniques for bounding and enumerating conjunctive queries to PCs.

5.1 Computing Constraints and Partitions

To take advantage of PCs, we need to be able to decompose an arbitrary relation $R(\mathbf{Z})$ according to a given partition constraint $PC(\mathcal{X}, \mathbf{Y}, d)$. For this task, we propose two poly-time algorithms; a linear approximate algorithm and a quadratic exact algorithm. Crucially, these algorithms do not need d to be computed beforehand, so these algorithms can also be used to compute PC constraints themselves, i.e. to compute the value of $PC(\mathcal{X}, \mathbf{Y})$. We start with the faster approximation algorithm before describing the exact algorithm.

Concretely, Algorithm 2 partitions a relation $R(\mathbf{Z})$ by distributing the tuples from the relation to partitions in a greedy fashion. At each point, it selects the set of variables $\mathbf{X} \in \mathcal{X}$ and particular value $\mathbf{x} \in \pi_{\mathbf{X}} R$ which occurs in the fewest tuples in the relation R. It then adds those tuples to the partition $R^{\mathbf{X}}$ and deletes them from the relation R. Intuitively, high degree pair $(\mathbf{X}, \mathbf{x})$ will be distributed to partitions late in this process. At this point, most of the matching tuples will already have been placed in different partitions. Formally, we claim the following runtime and approximation guarantee for Algorithm 2.

▶ **Theorem 14.** *For a relation $R(\mathbf{Z})$ and subsets $\mathbf{Y} \subseteq \mathbf{Z}, \mathcal{X} \subseteq 2^{\mathbf{Y}}$, Algorithm 2 computes a partitioning $\bigcup_{\mathbf{X} \in \mathcal{X}} R^{\mathbf{X}} = R$ in time $O(|R|)$ (data complexity) such that*

$$DC_{R^{\mathbf{x}}}(\mathbf{X}, \mathbf{Y}, |\mathcal{X}|d)$$

holds for every $\mathbf{X} \in \mathcal{X}$ where $d = PC(\mathcal{X}, \mathbf{Y})$.

Proof Sketch. We start by providing intuition for the linear runtime. Each iteration of the while loop (line 4) places a set of tuples in a partition (line 6) and removes them from the original relation (line 7). Both of these operations can be done in constant time per

Algorithm 2 Approximate Decomposition Algorithm.

decompose$(R, \mathcal{X}, \mathbf{Y})$
for $\mathbf{X} \in \mathcal{X}$ **do**
  $R^{\mathbf{X}} \leftarrow \emptyset$
while R is not empty **do**
  $\mathbf{X}, \mathbf{x} \leftarrow \text{argmin}_{\mathbf{X} \in \mathcal{X}, \mathbf{x} \in \pi_{\mathbf{X}} R} |\pi_{\mathbf{Y}} \sigma_{\mathbf{X}=\mathbf{x}} R|$
  $R^{\mathbf{X}} \leftarrow R^{\mathbf{X}} \cup \sigma_{\mathbf{X}=\mathbf{x}} R$
  $R \leftarrow R \setminus \sigma_{\mathbf{X}=\mathbf{x}} R$
return $(R^{\mathbf{X}})_{\mathbf{X} \in \mathcal{X}}, \max_{\mathbf{X} \in \mathcal{X}} DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y})$

tuple, so we simply need to show that we can identify the lowest degree attribute/value pair in constant time each iteration. This is done by creating a priority queue structure for each $\mathbf{X} \in \mathcal{X}$ where priority is equal to degree, and we begin by adding each tuple in R to each priority queue. Because the maximum degree is less than $|R|$, we can construct these queues in linear time using bucket sort. We will then decrement these queues by 1 each time a tuple is removed from R. While arbitrarily changing an element's priority typically requires $O(log(|R|))$ in a priority queue, we are merely decrementing by 1 which is a local operation that can be done in constant time. So, construction and maintenance of these structures is linear w.r.t. data size. We can then use these queues to look up the lowest degree attribute/value pair in constant time.

Next, we prove the approximation guarantee by contradiction. If the algorithm produces a partition $R^{\mathbf{X}}$ with $DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y}) > |\mathcal{X}|d$, then there must be some value $\mathbf{x} \in \pi_{\mathbf{X}} R^{\mathbf{X}}$ where $|\pi_{\mathbf{Y}} \sigma_{\mathbf{X}=\mathbf{x}} R^{\mathbf{X}}| > |\mathcal{X}|d$. At the moment before this value was inserted into $R^{\mathbf{X}}$ and deleted from R, all attribute/value pairs must have had degree at least $|\mathcal{X}|d$ in the current state of R which we denote $R_{\mathcal{A}}$. Through some algebraic manipulation, we show that this implies $|R_{\mathcal{A}}| > \sum_{\mathbf{X} \in \mathcal{X}} |\pi_{\mathbf{X}} R_{\mathcal{A}}|d$. On the other hand, we know that $R_{\mathcal{A}}$ respects the original partition constraint because $R_{\mathcal{A}} \subseteq R$, and we show that this implies the converse $|R_{\mathcal{A}}| \leq \sum_{\mathbf{X} \in \mathcal{X}} |\pi_{\mathbf{X}} R_{\mathcal{A}}|d$. This is a contradiction, so our algorithm must not produce a poor approximation in the first place. ◀

Next, we describe an exact algorithm that requires quadratic time. Intuitively, Algorithm 3 also computes a decomposition of R in a greedy fashion by iteratively allocating (groups of) tuples $\mathbf{y}_0$ of $\pi_Y R$ to partitions $R^{\mathbf{X}}$, preferring allocations to partitions where the maximum over the relevant degree constraints, i.e., $\max_{\mathbf{X}} DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y})$, does not increase. However, decisions greedily made at the start may be sub-optimal and may not lead to a decomposition that minimizes $\max_{\mathbf{X}} DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y})$. To overcome this, instead of simply allocating $\mathbf{y}_0$ to a partition, the algorithm also checks whether it is possible to achieve a better overall decomposition by reallocating some other tuples in a cascading manner. To that end, we look for elements $\mathbf{y}_1 \in \pi_Y R^{\mathbf{X}_1}, \ldots, \mathbf{y}_m \in \pi_Y R^{\mathbf{X}_m}$ and a further $\mathbf{X}_{m+1}$ such that for all $\mathbf{y}_i, i = 1, \ldots, m$, the tuples matching $\mathbf{y}_i$ can be moved from $R^{\mathbf{X}_i}$ to $R^{\mathbf{X}_{i+1}}$ and the tuples matching $\mathbf{y}_0$ can be added to $R^{\mathbf{X}_1}$, all without increasing $\max_{\mathbf{X}} DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y})$. To achieve this, $\mathbf{y}_i$ is selected such that it matches $\mathbf{y}_{i-1}$ on the variables $\mathbf{X}_i$. Thus, for each $R^{\mathbf{X}_i}$, the value of $DC_{R^{\mathbf{X}_i}}(\mathbf{X}, \mathbf{Y})$ is the same before and after the update. There only has to be space for $\mathbf{y}_m$ in the final relation $R^{\mathbf{X}_{m+1}}$.

The sequence $(\mathbf{y}_1, \ldots, \mathbf{y}_m, \mathbf{X}_1, \ldots, \mathbf{X}_{m+1})$ constitutes an augmenting path which was first introduced by [11] and used for matroids. Adapted to the present setting, we define an augmenting path as below. By slight abuse of notation, we write $\sigma_{\mathbf{X}=\mathbf{y}} R$ even though $\mathbf{y}$ fixes more variables than specified in $\mathbf{X}$. Naturally, this is meant to select those tuples of R that agree with $\mathbf{y}$ on the variables $\mathbf{X}$.

Algorithm 3 Exact Decomposition Algorithm.

```
1: decompose(R, 𝒳, Y)
2: d ← 0
3: for X ∈ 𝒳 do
4:     R^X ← ∅
5: for y_0 ∈ π_Y R do
6:     if there exists a shortest augmenting path (y_1, ..., y_m, X_1, ..., X_{m+1}) then
7:         for i = m, ..., 1 do
8:             R^{X_{i+1}} ← R^{X_{i+1}} ∪ σ_{Y=y_i} R^{X_i}
9:             R^{X_i} ← R^{X_i} \ σ_{Y=y_i} R^{X_i}
10:        R^{X_1} ← R^{X_1} ∪ σ_{Y=y_0} R
11:    else
12:        d ← d + 1
13:        R^X ← R^X ∪ σ_{Y=y_0} R          ▷ X ∈ 𝒳 can be selected arbitrarily here.
14:    R ← R \ σ_{Y=y_0} R
15: return (R^X)_{X∈𝒳}, d
```

▶ **Definition 15.** *Let $(R^{\mathbf{X}})_{\mathbf{X}\in\mathcal{X}}$ be pairwise disjoint subsets of some relation $R(\mathbf{Z})$ with $\mathbf{Y} \subseteq \mathbf{Z}, \mathcal{X} \subseteq 2^{\mathbf{Y}}$, and let $\mathbf{y}_0 \in \pi_{\mathbf{Y}} R \setminus \pi_{\mathbf{Y}} \bigcup_{\mathbf{X}\in\mathcal{X}} R^{\mathbf{X}}$ be a new tuple. An augmenting path $(\mathbf{y}_1, \ldots, \mathbf{y}_m, \mathbf{X}_1, \ldots, \mathbf{X}_{m+1})$ satisfies the following properties:*

1. *For all $i \in \{1, \ldots, m+1\} : \mathbf{X}_i \in \mathcal{X}$.*
2. *For all $i \in \{1, \ldots, m\} : \mathbf{y}_i \in \pi_{\mathbf{Y}} R^{\mathbf{X}_i}$.*
3. *For all $i \in \{1, \ldots, m\} : \mathbf{y}_i$ agrees with $\mathbf{y}_{i-1}$ on $\mathbf{X}_i$.*
4. *$|\pi_{\mathbf{Y}} \sigma_{\mathbf{X}_{m+1}=\mathbf{y}_m} R^{\mathbf{X}_{m+1}}| < \max_{\mathbf{X}} DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y})$.*

We omit the references to $(R^{\mathbf{X}})_{\mathbf{X}\in\mathcal{X}}$, $\mathbf{Y}$, and $\mathbf{y}_0$ when they are clear from the context.

The next theorem shows that Algorithm 3 correctly computes an optimal decomposition.

▶ **Theorem 16.** *For a relation $R(\mathbf{Z})$ and subsets $\mathbf{Y} \subseteq \mathbf{Z}, \mathcal{X} \subseteq 2^{\mathbf{Y}}$, Algorithm 3 computes a partitioning $\bigcup_{\mathbf{X}\in\mathcal{X}} R^{\mathbf{X}} = R$ in $O(|R|^2)$ time (data complexity) such that*

$$DC_{R^{\mathbf{X}}}(\mathbf{X}, \mathbf{Y}, d)$$

holds for every $\mathbf{X} \in \mathcal{X}$ where $d = PC(\mathcal{X}, \mathbf{Y})$.

Proof Sketch. The intuition for the algorithm's quadratic runtime boils down to ensuring that the search for an augmenting path is linear in $|R|$. To show that this is the case, we model the search for an augmenting path as a breadth-first search over a bipartite graph whose nodes correspond to full tuples $\mathbf{y} \in \pi_{\mathbf{Y}} R$ and the partial tuples $\mathbf{x} \in \pi_{\mathbf{X}} R$ for all $\mathbf{X} \in \mathcal{X}$. An edge exists between a tuple $\mathbf{y}$ and a partial tuple $\mathbf{x}$ if they agree on their shared attributes. This search starts from the new tuple $\mathbf{y}_0$ and completes when it finds a tuple $\mathbf{y}_m$ that can be placed in one of the relations $R^{\mathbf{X}}$ without increasing its DC. The number of edges and vertices in this graph is linear in the input data, so the breadth-first search is linear as well.

We now provide intuition for the algorithm's correctness. Suppose that we are partway through the algorithm and are now at the iteration where we add $\mathbf{y}_0$ to a partition. Further, let $R_{\mathcal{A}}$ be the set of tuples which have been added so far, including $\mathbf{y}_0$. Because there exists an optimal partitioning of $R_{\mathcal{A}}$, we should be able to place $\mathbf{y}_0$ in the partition where it exists in the optimal partitioning. If we cannot, then a tuple in that partition with the same $\mathbf{X}$-value must not be in its optimal partition. We identify one of these tuples and

move it to its optimal partition. We continue this process inductively of shifting one tuple at a time to its optimal partition until one of these shifts no longer violates the PC. The sequence of shifts that we have performed constitutes an augmenting path by definition, and its existence implies that we would not violate the PC in this iteration by incrementing d past it. This construction process must end in a finite number of moves because each step increases the number of tuples placed in their optimal partitioning. If all tuples are in their optimal partition, the final shift must not have violated the PC due to the optimal partition's definition. ◀

5.2 Lifting DC-Based Bounds and Algorithms to PCs

We now apply the developed decomposition algorithms to the general case and show how to use WCOJ algorithms as a black box to carry over guarantees for DCs to the general case of PCs. First, we extend the definition of an arbitrary cardinality bound CB defined over sets of DCs to sets of PCs.

▶ **Definition 17.** *Let CB be a cardinality bound. Then, we extend CB to PCs by setting*

$$CB(Q, \mathbf{PC}) := \sum_{\mathbf{X}^1 \in \mathcal{X}_1, \cdots, \mathbf{X}^l \in \mathcal{X}_l} CB(Q, \{DC_{P_1}(\mathbf{X}^1, \mathbf{Y}_1, d_1), \ldots, DC_{P_l}(\mathbf{X}^l, \mathbf{Y}_l, d_l)\}),$$

where $\mathbf{PC} = \{PC_{P_i}(\mathcal{X}_i, \mathbf{Y}_i, d_i) \mid i\}$ is an arbitrary set of partition constraints.

Note that the bound in Definition 17 is well-defined: If $\mathbf{PC}$ is simply a set of DCs, the extended version of the cardinality bound CB coincides with its original definition. If $\mathbf{PC}$ is an arbitrary set of PCs, then it remains a valid bound on the size of the join.

▶ **Proposition 18.** *Let $CB(Q, \mathbf{PC})$ be an extended cardinality bound. Then,*

$$|Q^I| \leq CB(Q, \mathbf{PC}) \quad \forall I \vDash \mathbf{PC}.$$

Proof. Let I be a database satisfying $\mathbf{PC} = \{PC_{P_i}(\mathcal{X}_i, \mathbf{Y}_i, d_i) \mid i = 1, \ldots, l\}$ and $Q \leftarrow R_1(\mathbf{Z}_1) \bowtie \cdots \bowtie R_k(\mathbf{Z}_k)$. Thus, for each P_i there is a partitioning $P_i = \bigcup_{j=1}^{|\mathcal{X}_i|} P_i^j$ with $DC_{P_i^j}(\mathbf{X}_i^j, \mathbf{Y}_i, d_i)$ and $\mathcal{X}_i = \{X_i^j | j = 1, \ldots, |\mathcal{X}_i|\}$. Now, let us now fix some $j_1 \in \{1, \ldots, |\mathcal{X}_1|\}, \ldots, j_l \in \{1, \ldots, |\mathcal{X}_l|\}$. Furthermore, for each $i \in \{1, \ldots, l\}$ let $s(i) \in \{1, \ldots, k\}$ be the relation $R_{s(i)} = P_i$. Set $R_s^{j_1 \cdots j_l} = \bigcap_{i:s=s(i)} P_i^{j_i}$ for all $s \in \{1, \ldots, k\}$. Then consider $Q^{j_1 \cdots j_l} = R_1^{j_1 \cdots j_l} \bowtie \cdots \bowtie R_k^{j_1 \cdots j_l}$. We claim two things:

1. $Q^{j_1 \cdots j_l}$ partitions Q^I, i.e., $Q^I = \bigcup_{j_1=1}^{|\mathcal{X}_1|} \cdots \bigcup_{j_l=1}^{|\mathcal{X}_l|} Q^{j_1 \cdots j_l}$, and
2. $|Q^{j_1 \cdots j_l}| \leq CB(Q, \{DC_{P_1}(\mathbf{X}_1^{j_1}, \mathbf{Y}_1, d_1), \ldots, DC_{P_l}(\mathbf{X}_l^{j_l}, \mathbf{Y}_l, d_l)\})$.

For the first bullet point, let $t \in Q^I$. Clearly, for each $i \in \{1, \ldots, l\}$ there exists exactly one j_i such that t agrees with an element in $P_i^{j_i}(\mathbf{Z}_{s(i)})$ on the variables $\mathbf{Z}_{s(i)}$. Thus, $t \in Q^{j_1 \cdots j_l}$.

For the second bullet point, consider the relations $P_i^{j_i}$. These are supersets of the relations $R_{s(i)}^{j_1 \cdots j_l}$ and, therefore, we can assert $DC_{R_{s(i)}^{j_1 \cdots j_l}}(\mathbf{X}_i^{j_i}, \mathbf{Y}_i, d_i)$. Viewed as a query, $Q^{j_1 \cdots j_l}$ has the same form as Q. Hence, $|Q^{j_1 \cdots j_l}| \leq CB(Q, \{DC_{P_1}(\mathbf{X}_1^{j_1}, \mathbf{Y}_1, d_1), \ldots, DC_{P_l}(\mathbf{X}_l^{j_l}, \mathbf{Y}_l, d_l)\})$. ◀

Crucially, this upper bound is not loose; it preserves the tightness of any underlying DC-based bound. Specifically, the extended version of the combinatorics bound CB_{Comb} is asymptotically close to the actual worst-case size of the join, with the constant depending on the query. For example, the extended version of combinatorics bound is $O(n)$ for the query $Q_{\hexagon}$ (see Section 4) when using all PCs while the combinatorics bound based on the DCs alone was $\Omega(n^{\frac{4}{3}})$.

▶ **Proposition 19.** *Let $CB_{Comb}(Q, \mathbf{PC})$ be the extended version of the combinatorics bound CB_{Comb}. Then,*

$$CB_{Comb}(Q, \mathbf{PC}) = O(\max_{I \vDash \mathbf{PC}} |Q^I|).$$

Proof. Let $\mathbf{X}^1 \in \mathcal{X}_1, \ldots, \mathbf{X}^l \in \mathcal{X}_l$ be such that $CB_{Comb}(Q, \mathbf{DC})$ is maximized where $\mathbf{DC} := \{DC_{P_1}(\mathbf{X}^1, \mathbf{Y}_1, d_1), \ldots, DC_{P_l}(\mathbf{X}^l, \mathbf{Y}_l, d_l)\}$. Thus, there exists a database I such that $I \vDash \mathbf{DC}$ and $|Q^I| = CB_{Comb}(Q, \mathbf{DC})$. Furthermore, I must then also trivially satisfy $\mathbf{PC}$ by definition. For each PC on P_i the witnessing partitioning is simply $P_i = P_i \cup \emptyset \cup \cdots \cup \emptyset$. Therefore,

$$CB_{Comb}(Q, \mathbf{PC}) \leq |\mathcal{X}_1| \ldots |\mathcal{X}_l| |Q^I| = O(\max_{D \vDash \mathbf{PC}} |Q^I|).$$

For the last equality, note that $|\mathcal{X}_1| \ldots |\mathcal{X}_l|$ is a query-dependent constant as we assume all PCs to be on different variables $\mathcal{X}_i$. ◀

Next, we show how algorithms that are worst-case optimal relative to a cardinality bound can likewise be adapted and become worst-case optimal relative to the extended bound. In this way, progress on WCOJ algorithms based on DCs immediately leads to improved algorithms that take advantage of PCs.

▶ **Theorem 20.** *Given a cardinality bound $CB_{\mathcal{A}}$. If there exists a join enumeration algorithm $\mathcal{A}$ which is worst-case optimal relative to $CB_{\mathcal{A}}$, then there exists an algorithm $\mathcal{A}^*$ which is worst-case optimal relative to the extended version of the cardinality bound. I.e., for fixed query Q, $\mathcal{A}^*$ runs in time $O(|I| + CB_{\mathcal{A}}(Q, \mathbf{PC}))$ for arbitrary $\mathbf{PC}$ and database $I \vDash \mathbf{PC}$.*

Proof. Suppose we have a query $Q \leftarrow R_1(\mathbf{Z}_1) \bowtie \ldots R_k(\mathbf{Z}_k)$, a set of partition constraints $\mathbf{PC} = \{PC_{P_1}(\mathcal{X}_1, \mathbf{Y}_1, d_1), \ldots, PC_{P_l}(\mathcal{X}_l, \mathbf{Y}_l, d_l)\}$, and a database $I \vDash \mathbf{PC}$. We can follow the same idea already used in the proof of Proposition 18. Concretely, we partition each P_i into $(P_i^j)_{j=1,\ldots,|\mathcal{X}_i|}$. For this, we use Algorithm 2 and Theorem 14. Thus, we get $DC_{P_i^j}(\mathbf{X}_i^j, \mathbf{Y}_i, |\mathcal{X}_i| d_i)$. Let $\alpha := \max_i |\mathcal{X}_i|$.

We can now continue following the idea of proof of Proposition 18 up to the two claims where we only need the first. Now, instead of bounding the size of $Q^{j_1,\ldots,j_l}$, we now want to compute each subquery using $\mathcal{A}$. Thus, note that $DC_{R^{j_1\cdots j_l}_{s(i)}}(\mathbf{X}_i^{j_i}, \mathbf{Y}_i, \alpha d_i)$. This implies that $\mathcal{A}$ runs in time $O(|I| + CB_{\mathcal{A}}(Q, \{DC_{P_i}(\mathbf{X}_i^{j_i}, \mathbf{Y}_i, \alpha d_i) \mid i\}))$. The constant α can be hidden in the O-notation while summing up the time required for each $Q^{j_1,\ldots,j_l}$ results in an overall runtime of $O(|I| + \sum_{\mathbf{X}^1 \in \mathcal{X}_1 \cdots \mathbf{X}^l \in \mathcal{X}_l} CB_{\mathcal{A}}(Q, \{DC_{P_i}(\mathbf{X}_i^{j_i}, \mathbf{Y}_i, d_i) \mid i\}) = O(|I| + CB_{\mathcal{A}}(Q, \mathbf{PC}))$. ◀

For example, PANDA is a WCOJ algorithm relative to the polymatroid bound, and we can use this approach to translate it to an optimal algorithm for the extended polymatroid bound. While it is an open problem to produce a WCOJ algorithm relative to $CB_{Comb}(Q, \mathbf{DC})$, any such algorithm now immediately results in a WCOJ algorithm relative to $CB_{Comb}(Q, \mathbf{PC})$. Combined with Proposition 19 this implies that such an algorithm then only takes time relative to the worst-case join size of instances that satisfy the same set of PCs.

▶ **Corollary 21.** *If there exists a WCOJ algorithm relative to $CB_{Comb}(Q, \mathbf{DC})$, then there exists an WCOJ algorithm relative to $CB_{Comb}(Q, \mathbf{PC})$.*

6 Conclusions and Future Work

In this work, we introduced PCs as a generalization of DCs, uncovering a latent structure within relations and that is present in standard benchmarks. PCs enable a more refined approach to query processing, offering asymptotic improvements to both cardinality bounds and join algorithms. We presented algorithms to compute PCs and identify the corresponding partitioning that witness these constraints. To harness this structure, we then developed techniques to lift both cardinality bounds and WCOJ algorithms from the DC framework to the PC framework. Crucially, our use of DC-based bounds and algorithms as black boxes allows future advances in the DC setting to be seamlessly integrated into the PC framework.

On the practical side, future research should explore when and where it is beneficial to leverage the additional structure provided by PCs. In particular, finding ways to minimize the constant factor overhead by only considering a useful subset of PCs or sharing work across evaluations on different partitions could yield significant practical improvements in query performance. On the other hand, further theoretical work should try to incorporate additional statistics into this partitioning framework, e.g., l_p-norms of degree sequences. Ultimately, the goal of this line of work is to capture the inherent complexity of join instances through both the structure of the query and the data.

References

1 Christopher R. Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. Emptyheaded: A relational engine for graph processing. *ACM Trans. Database Syst.*, 42(4):20:1–20:44, 2017. doi:10.1145/3129246.

2 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013. doi:10.1137/110859440.

3 Suman K. Bera, Lior Gishboliner, Yevgeny Levanzov, C. Seshadhri, and Asaf Shapira. Counting subgraphs in degenerate graphs. *J. ACM*, 69(3):23:1–23:21, 2022. doi:10.1145/3520240.

4 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ITCS.2020.38.

5 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2315–2332. SIAM, 2021. doi:10.1137/1.9781611976465.138.

6 Suman K. Bera and C. Seshadhri. How the degeneracy helps for triangle counting in graph streams. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 457–467. ACM, 2020. doi:10.1145/3375395.3387665.

7 Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 276–285. IEEE, 2021. doi:10.1109/FOCS52979.2021.00036.

8 Kyle Deeds and Timo Camillo Merkl. Partition constraints for conjunctive queries: Bounds and worst-case optimal joins [technical report], 2025. arXiv:2501.04190.

9 Kyle Deeds, Dan Suciu, Magda Balazinska, and Walter Cai. Degree sequence bound for join cardinality estimation. In Floris Geerts and Brecht Vandevoort, editors, *26th International Conference on Database Theory, ICDT 2023, March 28-31, 2023, Ioannina, Greece*, volume 255 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICDT.2023.8.

10 Kyle B. Deeds, Dan Suciu, and Magdalena Balazinska. Safebound: A practical system for generating cardinality bounds. *Proc. ACM Manag. Data*, 1(1):53:1–53:26, 2023. doi:10.1145/3588907.

11 Jack Edmonds. Minimum partition of a matroid into independent subsets. *J. Res. Nat. Bur. Standards Sect. B*, 69:67–72, 1965.

12 Michael J. Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, and Thomas Neumann. Adopting worst-case optimal joins in relational database systems. *Proc. VLDB Endow.*, 13(11):1891–1904, 2020. URL: http://www.vldb.org/pvldb/vol13/p1891-freitag.pdf.

13 Lior Gishboliner, Yevgeny Levanzov, Asaf Shapira, and Raphael Yuster. Counting homomorphic cycles in degenerate graphs. *ACM Trans. Algorithms*, 19(1):2:1–2:22, 2023. doi:10.1145/3560820.

14 Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16:1–16:35, 2012. doi:10.1145/2220357.2220363.

15 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 289–298. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109590.

16 Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. Cardinality estimation in DBMS: A comprehensive benchmark evaluation. *Proc. VLDB Endow.*, 15(4):752–765, 2021. doi:10.14778/3503585.3503586.

17 Mahmoud Abo Khamis, Vasileios Nakos, Dan Olteanu, and Dan Suciu. Join size bounds using lp-norms on degree sequences. *Proc. ACM Manag. Data*, 2(2):96, 2024. doi:10.1145/3651597.

18 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 327–342. ACM, 2016. doi:10.1145/2902251.2902289.

19 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 Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 429–444. ACM, 2017. doi:10.1145/3034786.3056105.

20 Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, 2015. doi:10.14778/2850583.2850594.

21 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 37–48. ACM, 2012. doi:10.1145/2213556.2213565.

22 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013. doi:10.1145/2590989.2590991.

23 Shixuan Sun and Qiong Luo. In-memory subgraph matching: An in-depth study. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1083–1098. ACM, 2020. doi:10.1145/3318464.3380581.

24 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.

25 Yisu Remy Wang, Max Willsey, and Dan Suciu. Free join: Unifying worst-case optimal and traditional joins. *Proc. ACM Manag. Data*, 1(2):150:1–150:23, 2023. doi:10.1145/3589295.