Efficient Algorithms for Cardinality Estimation and Conjunctive Query Evaluation With Simple Degree Constraints

SUNGJIN IM, University of California at Santa Cruz, USA
BENJAMIN MOSELEY, Carnegie Mellon University, United States
HUNG Q. NGO, RelationalAI, USA
KIRK PRUHS, University of Pittsburgh, USA

Cardinality estimation and conjunctive query evaluation are two of the most fundamental problems in database query processing. Recent work proposed, studied, and implemented a robust and practical information-theoretic cardinality estimation framework. In this framework, the estimator is the cardinality upper bound of a conjunctive query subject to "degree-constraints", which model a rich set of input data statistics. For general degree constraints, computing this bound is computationally hard. Researchers have naturally sought efficiently computable relaxed upper bounds that are as tight as possible. The polymatroid bound is the tightest among those relaxed upper bounds. While it is an open question whether the polymatroid bound can be computed in polynomial-time in general, it is known to be computable in polynomial-time for some classes of degree constraints.

Our focus is on a common class of degree constraints called simple degree constraints. Researchers had not previously determined how to compute the polymatroid bound in polynomial time for this class of constraints. Our first main result is a polynomial time algorithm to compute the polymatroid bound given simple degree constraints. Our second main result is a polynomial-time algorithm to compute a "proof sequence" establishing this bound. This proof sequence can then be incorporated in the PANDA-framework to give a faster algorithm to evaluate a conjunctive query. In addition, we show computational limitations to extending our results to broader classes of degree constraints. Finally, our technique leads naturally to a new relaxed upper bound called the *flow bound*, which is computationally tractable.

CCS Concepts: • **Theory of computation → Database query languages (principles)**.

Additional Key Words and Phrases: Cardinality estimation, conjunctive query evaluation, polymatroid bound, proof sequence, polynomial time algorithms

ACM Reference Format:
Sungjin Im, Benjamin Moseley, Hung Q. Ngo, and Kirk Pruhs. 2025. Efficient Algorithms for Cardinality Estimation and Conjunctive Query Evaluation With Simple Degree Constraints. *Proc. ACM Manag. Data* 3, 2 (PODS), Article 96 (May 2025), 26 pages. https://doi.org/10.1145/3725233

Authors' Contact Information: Sungjin Im, sungjin@ucsc.edu, University of California at Santa Cruz, Computer Science and Engineering, Santa Cruz, California, USA; Benjamin Moseley, Carnegie Mellon University, Tepper School of Business, Pittsburgh, PA, United States, moseleyb@andrew.cmu.edu; Hung Q. Ngo, RelationalAI, Berkeley, CA, USA, hung.q.ngo@gmail.com; Kirk Pruhs, University of Pittsburgh, Computer Science Department, Pittsburgh, PA, USA, kirk@cs.pitt.edu.

ACM 2836-6573/2025/5-ART96
https://doi.org/10.1145/3725233

1 Introduction

1.1 Motivations

Estimating a tight upper bound on the output size of a (full) conjunctive query is an important problem in query optimization and evaluation, for many reasons. First, the bound is used to estimate whether the computation can fit within a given memory budget. Second, cardinalities of intermediate relations are the main parameters used to estimate the cost of query plans [23], and the intermediate size bound is used as a *cardinality estimator* [9, 10, 17, 18, 26]. These "pessimistic" estimators are designed to be robust [31], removing the assumptions that lead to the well-documented impediment of selectivity-based estimators [21], where the estimates often under approximate the real intermediate sizes by orders of magnitudes. Third, the bound is used as a yardstick to measure the quality of a join algorithm [25].

The history of bounding the output size of a conjunctive query (CQ) is rich [2–6, 8, 11, 12, 14, 15, 19, 24, 25, 29]. In the simplest setting where the query is a join of base tables with known cardinalities, the *AGM-bound* [5] is tight w.r.t. data complexity. up to a $O(2^n)$ factor, where n is the number of variables in the query. Join algorithms running in asymptotically the same amount of time are "worst-case optimal" [27, 28, 30]. In practice, however, we know a lot more information about the inputs beyond cardinalities. Query optimizers typically make use of distinct value counts, heavy hitters, primary and and foreign keys information [22, 26], in addition to function predicates. Every additional piece of information is a constraint that may drastically reduce the size of the query output. For example, in a triangle query $R(a,b) \wedge S(b,c) \wedge T(a,c)$, if we knew a was a primary key in relation R, then the output size is bounded by $\min\{|T|, |R| \cdot |S|\}$, which can be asymptotically much less than the AGM-bound of $\sqrt{|R| \cdot |S| \cdot |T|}$.

To model these common classes of input statistics and key constraints, Abo Khamis et al. [2, 3] introduced an abstract class of constraints, called degree constraints.[1] A degree constraint is a triple (X, Y, c), where $X \subseteq Y$ are sets of variables, and 2^c is positive integer. The constraint holds for a predicate or relation R if $\max_{\boldsymbol{x}} |\pi_Y \sigma_{X=\boldsymbol{x}} R| \leq 2^c$, where π and σ are the projection and selection operators, respectively. The intuitive meaning of the triple (X, Y, c) is that the number of possible bindings for variables in Y can attain, given a particular binding of the variables in X, is at most 2^c. If $X = \emptyset$ then this is a *cardinality constraint*, which says $|\pi_Y R| \leq 2^c$; if $c = 0$ (i.e. $2^c = 1$) then this is a *functional dependency* from X to Y.

Given a set DC of degree constraints, and a full CQ Q; we write $\boldsymbol{I} \models \text{DC}$ to denote that a database instance $\boldsymbol{I}$ satisfies the constraints DC. A robust cardinality estimator is $\sup_{\boldsymbol{I} \models \text{DC}} |Q(\boldsymbol{I})|$, the upper bound on the number of output tuples of Q over all databases satisfying the constraints [7, 17, 26]. We will refer to this bound as the *combinatorial bound*. For arbitrary degree constraints, it is not clear how to even compute it. Even in the simplest case when all degree constraints are cardinality constraints, approximating the combinatorial bound better than the 2^n-ratio is already hard [5], where n is the number of variables in Q.

Consequently, researchers have naturally sought relaxed upper bounds to the combinatorial bound that are (efficiently) computable, and that are as tight as possible [3, 5, 13, 14, 25, 26, 31]. Most of these bounds are characterized by two collections: the collection $\mathcal{F}$ of set-functions $h : 2^{[n]} \to \mathbb{R}_+$ we are optimizing over, and the collection DC of (degree-) constraints the functions have to satisfy. Here, n is the number of variables in the query. Abusing notations, we also use DC to denote the constraints over set-functions: $h(Y) - h(X) \leq c$, for every $(X, Y, c) \in \text{DC}$. Given DC and $\mathcal{F}$, define

$$\text{DC}[\mathcal{F}] := \sup\{h([n]) \mid h \in \mathcal{F} \cap \text{DC}\}. \tag{1}$$

[1] Degree constraints are also generalized to frequency moments collected on input tables [19, 26].

Different classes of $\mathcal{F}$ and DC give rise to classes of bounds with varying degrees of tightness (how close to the combinatorial bound) and computational complexities [3]. This paper focuses on the *polymatroid bound* [3, 19], obtained by setting $\mathcal{F} = \Gamma_n$, the collection of all n-dimensional polymatroid functions. The upper bound $\log_2 \sup_{\boldsymbol{I} \models \mathrm{DC}} |Q(\boldsymbol{I})| \leq \mathrm{DC}[\Gamma_n]$ is the *tightest* known upper bound that is plausibly computable in polynomial-time [26]. (See Section 2 for more.)

More precisely, the *polymatroid bound* $\mathrm{DC}[\Gamma_n]$ for a collection $\mathrm{DC} = \{(X_i, Y_i, c_i) \mid i \in [k]\}$ of k degree constraints over variables $X_i \subseteq Y_i \subseteq [n]$, can be computed by solving:

$$\begin{array}{lrcll} \max & h([n]) & & & \\ \text{s.t.} & h(Y_i) - h(X_i) & \leq & c_i & \forall i \in [k] \\ & h & \in & \Gamma_n & \end{array} \tag{2}$$

The optimization problem (2) is a linear program (LP, see Sec. 2). with an *exponential* number of variables and constraints. In the query processing pipeline, for every query the optimizer has to issue *many* cardinality estimates when searching for an optimal query plan. Hence, solving the LP (2) in polynomial time is crucial.

While it is an *open* question whether the polymatroid bound can be computed in poly-time for general degree constraints, it is known to be computable in poly-time if the input degree constraints fall into one of the following cases: (a) all degree constraints are cardinality constraints [5, 19]; in this case the polymatroid bound is the same as the AGM bound; (b) the *constraint dependency graph* is *acyclic* [25]; this is a generalization of the all cardinality constraints case;[2] or (c) the constraints include cardinality constraints and *simple* functional dependencies (FD) [14]; Simple FDs are degree constraints of the form (X, Y, c) with $c = 0$ and $|X| = 1$

Our focus in this paper will be on another class of commonly occurring types of degree constraints called *simple* degree constraints. A degree constraint (X, Y, c) is *simple* if $|X| \leq 1$. This class strictly generalizes cardinality constraints and simple functional dependencies. Simple degree constraints also occur in the context of query containment under bag semantics, where they are the key ingredient for the rare special case when the problem is known to be decidable [1]. In particular, Lemma 3.13 from [1] showed that, for simple degree constraints, there is always an optimal polymatroid function of a special type called a *normal* function. The proof was a rather involved constructive argument that showed how to iteratively modify any feasible polymatroid function to obtain a feasible normal function without decreasing the objective value. While this observation offered some structural insight, it certainly did not resolve the issue of whether the polymatroid bound could be efficiently computed when the degree constraints are simple.

The polymatroid bound (2) is deeply connected to conjunctive query evaluation thanks to the PANDA framework [3, 19]. From an optimal solution to the dual D of the LP (2), [3] showed how to construct a specific linear inequality that is valid for all polymatroids called a *Shannon-flow inequality*. The validity of the Shannon-flow inequality can be proved by constructing a specific sequence of elemental Shannon inequalities. This sequence is called a *proof-sequence* for the Shannon-flow inequality. From a proof sequence of length ℓ, the PANDA algorithm can answer a conjunctive query in time $O(|\boldsymbol{I}|) + (\log |\boldsymbol{I}|)^{O(\ell)} U$, where $U = 2^{\mathrm{DC}[\Gamma_n]}$ is the upper bound on the output size given by the polymatroid bound, and $\boldsymbol{I}$ is the input database instance.

While PANDA is a *very general* framework for algorithmically constructing completely non-trivial query plans that optimize for the total number of tuples scanned, the major drawback in the runtime expression is that as the length of the proof sequence appears in the exponent. Previous results [3] can only establish ℓ to be *exponential* in n. A shorter proof sequence would drastically improve the running time of PANDA.

[2]The constraint dependency graph is the graph whose vertices are the variables $[n]$ and there is a directed edge (u, v) if and only if there exists a degree constraint (X_i, Y_i, c_i) where $u \in X_i$ and $v \in Y_i - X_i$.

The problems outlined above set the context for our questions and contributions. First, under simple degree constraints is it possible to efficiently compute the polymatroid bound? Second, can a proof sequence of polynomially bounded length be constructed? And third, what is a good relaxation of the polymatroid bound that is tractable?

1.2 Our Contributions

Our first main contribution, stated in Theorem 1.1, is to show that the polymatroid bound can be modeled by a *polynomial-sized* LP for simple degree constraints, and thus is computable in poly-time when the degree constraints are simple.

Theorem 1.1. *Let* DC *be a collection of k simple degree constraints over n variables. The polymatroid bound* $\mathrm{DC}[\Gamma_n]$ *can be modeled by a LP where the number of variables is $O(kn^2)$ and the number of constraints is $O(kn)$. Thus the polymatroid bound is computable in time polynomial in n and k.*

We prove Theorem 1.1 in Section 3, but let us give a brief overview here. The proof outline is shown schematically in Figure 1. The LP formulation P of the polymatroid bound $\mathrm{DC}[\Gamma_n]$ in (2) has exponentially many variables and exponentially many constraints, and so does P's dual D. The starting point is the observation that there are only polynomially many constraints in P where the constant right-hand side is non-zero (namely the constraints that correspond to the degree constraints), and thus the objective of D only has polynomial size. By projecting the feasible region for the LP D down onto the region of the variables in the objective, we show how to obtain an LP formulation $D_{\boldsymbol{\delta}}^{\text{simple}}$ of the polymatroid bound that has polynomially many variables, but still exponentially many constraints. The most natural interpretation of $D_{\boldsymbol{\delta}}^{\text{simple}}$ is as a sort of min-cost *hypergraph* cut problem. We show that this hypergraph cut problem can be reformulated as n min-cost network flow problems in a standard graph. This results in a natural polynomially-sized LP formulation $D_{\boldsymbol{\delta}}^{\text{flow}}$.

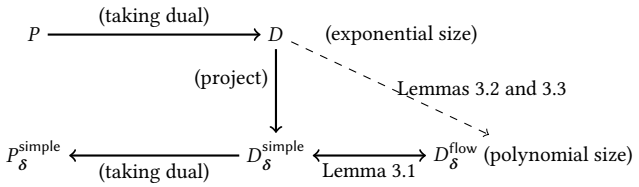

Fig. 1. Outline of the proof of Theorem 1.1.

As an ancillary result, we observe in Section 3 that the dual of $D_{\boldsymbol{\delta}}^{\text{simple}}$ is a natural LP formulation of the polymatroid bound when the polymatroidal functions are restricted to normal functions. This gives an alternate proof, that uses LP duality, of the fact that for simple degree constraints there is an optimal polymatroid function that is normal. We believe that this dual-project-dual proof is simpler and more informative than the inductive proof in [1].

Our second main contribution, stated in Theorem 1.2, is to show that there is a polynomial length proof sequence for simple instances. Beyond improving PANDA's runtime, this theorem is a fundamental result about the geometry of submodular functions.

Theorem 1.2. *Let* DC *be a collection of k simple degree constraints over n variables. There is a polynomial-time algorithm that computes a proof sequence for the polymatroid bound of length $O(k^2n^2 + kn^3)$.*

The proof of Theorem 1.2 in Section 4 constructively shows how to create a proof sequence from a feasible solution to our LP $D_{\boldsymbol{\delta}}^{\text{flow}}$, where the length of the proof sequence is linear in the cardinality of the support of the feasible solution. Intuitively, this proof sequence is exponentially shorter than the one that would be produced using the method in [3] (even under simple degree constraints) because it utilizes the special properties of the types of feasible solutions for the LP D that are derivable from feasible solutions to our LP $D_{\boldsymbol{\delta}}^{\text{flow}}$. And because the proof sequence is exponentially shorter, this exponentially decreases the run time when using the PANDA algorithm.

We next prove that allowing other types of degree constraints that are "just beyond simple" in some sense results in problems that are as hard as general constraints. The proof of the following theorem is in Section 5.

Theorem 1.3. *Even if the set of input degree constraints* DC *is restricted to either any one of the following cases, computing* $\mathsf{DC}[\Gamma_n]$ *is still as hard as computing the polymatroid bound for general degree constraints:*

(a) If DC *is a union of a set of simple degree constraints and a set of acyclic degree constraints.*
(b) If DC *is a union of a set of simple degree constraints and a set of FD constraints.*
(c) If DC *contains only degree constraints* (X, Y, c) *with* $|X| \leq 2$ *and* $|Y| \leq 3$.

As for simple degree constraints there is always an optimal polymatroid function that is normal, one might plausibly conjecture that an explanation for the polymatroid bound being efficiently computable for simple degree constraints is that the optimal normal function is efficiently computable for general degree constraints. We show that this conjecture is highly unlikely to be true by showing that it is NP-hard to compute the optimal normal function on general instances. This is stated in Theorem 1.4, which is proved in Section 6.

Theorem 1.4. *Given a collection* DC *of degree constraints, it is NP-hard to compute the polymatroid bound* $\mathsf{DC}[\Gamma_n]$ *when the functions are additionally restricted to be normal.*

Theorem 1.3 implies that, perhaps, we have reached (or are at least nearing) the limits of natural classes of degree constraints for which computing the polymatroid bound is easier than computing the polymatroid bound for general instances. We next seek a tractable relaxation of the polymatroid bound. Section 7 introduces a new upper bound called the *flow bound*, denoted by $\text{flow-bound}(\mathsf{DC}, \pi)$, which is parameterized by a permutation π of the variables. The flow bound is strictly better than the previously known tractable relaxation called the *chain bound* $\mathsf{DC}_\pi[\Gamma_n]$ (See [25] and Sec 2), thus it can be used as a tighter yet poly-time computable cardinality estimator.

Theorem 1.5. *The flow-bound satisfies the following properties:*

(a) For all collections DC *of degree constraints, and any given permutation* π *of the variables, the flow bound can be computed in polynomial time, and is tighter than the chain bound, that is:*

$$\mathsf{DC}[\Gamma_n] \leq \text{flow-bound}(\mathsf{DC}, \pi) \leq \mathsf{DC}_\pi[\Gamma_n] \tag{3}$$

(b) If DC *is either simple or acyclic, then in polynomial time in* n *and* $|\mathsf{DC}|$*, we can compute a permutation* π *such that* $\text{flow-bound}(\mathsf{DC}, \pi) = \mathsf{DC}[\Gamma_n]$.
(c) There are classes of instances DC *where there exists permutations* π *for which the ratio-gap* $\frac{\mathsf{DC}_\pi[\Gamma_n]}{\text{flow-bound}(\mathsf{DC},\pi)}$ *is unbounded above.*

2 Background

This section presents the minimal background required to understand the results in this paper.

2.1 Classes of set functions

Given a function $h : 2^{[n]} \to \mathbb{R}_+$, and $X \subseteq Y \subseteq [n]$, define $h(Y|X) = h(Y) - h(X)$. The function h is *monotone* if $h(Y|X) \geq 0$ for all $X \subseteq Y$; it is *submodular* if $h(I|I \cap J) \geq h(I \cup J|J)$, $\forall I, J \subseteq [n]$. It is a *polymatroid* if it is non-negative, monotone, submodular, with $h(\emptyset) = 0$. For every $W \subsetneq [n]$, a *step function* $s_W : 2^{[n]} \to \mathbb{R}_+$ is defined by $s_W(X) = 0$ if $X \subseteq W$ and $s_W(X) = 1$ otherwise. A function h is *normal* (also called a *weighted coverage function*) if it is a non-negative linear combination of step functions. A function h is *modular* if there is a non-negative value w_i for each variable $i \in [n]$ such that for all $X \subseteq [n]$ it is the case that $h(X) = \sum_{i \in X} w_i$.

Let Γ_n, M_n, N_n denote the set of all polymatroid, modular, normal functions over $[n]$, respectively. It is known [3] that $M_n \subseteq N_n \subseteq \Gamma_n$. Note that all three sets are polyhedral. In other words, when we view each function as a vector over $2^{[n]}$, the set of vectors forms a convex polyhedron in that vector space, defined by a finite number of hyperplanes. To optimize linear objectives over these sets is to solve linear programs.

2.2 Shannon-flow inequalities and proof sequences

Let $\mathcal{P} \subseteq 2^{[n]} \times 2^{[n]}$ denote the set of all pairs (X, Y) such that $\emptyset \subseteq X \subsetneq Y \subseteq [n]$. Let $\boldsymbol{\delta} = (\delta_{Y|X})_{(X,Y) \in \mathcal{P}}$ be a vector of non-negative reals. The inequality

$$h([n]|\emptyset) \leq \sum_{(X,Y) \in \mathcal{P}} \delta_{Y|X} \cdot h(Y|X) \tag{4}$$

is called a *Shannon-flow inequality* [3] if it is satisfied by all polymatroids $h \in \Gamma_n$.

One way to prove that (4) holds for all polymatroids is to turn the RHS into the LHS by repeatedly applying one of the following replacements:

- *Decomposition:* $h(Y|\emptyset) \to h(X|\emptyset) + h(Y|X)$, for $X \subsetneq Y$.
- *Composition:* $h(X|\emptyset) + h(Y|X) \to h(Y|\emptyset)$, for $X \subsetneq Y$.
- *Monotonicity:* $h(Y|X) \to 0$, for $X \subseteq Y$.
- *Submodularity:* $h(I|I \cap J) \to h(I \cup J|J)$, for $I \perp J$, which means $I \not\subseteq J$ and $J \not\subseteq I$.

Note that, we can replace $\to$ by $\geq$ in all four cases above to obtain valid Shannon inequalities that are satisfied by all polymatroids. Each replacement step is called a "*proof step*," which can be multiplied with a *non-negative* weight w. For example, $w \cdot h(I|I \cap J) \geq w \cdot h(I \cup J|J)$ is a valid inequality. We prepend w to the name of the operation to denote the weight being used, so we will refer to the prior inequality as an w-submodularity step. Note also that, the terms $h(Y|X)$ are manipulated as symbolic variables.

A *proof sequence* for the inequality (4) is a sequence of proof steps, where

- Every step is one of the above proof steps, accompanied by a non-negative weight w
- After every step is applied, the coefficient of every term $h(Y|X)$ remains non-negative.
- The sequence starts with the RHS of (4) and ends with the LHS.

Every proof step thus transforms a collection of non-negatively weighted ("conditional polymatroid") terms into another collection of non-negatively weighted terms. One of the main results in the paper [3] stats that, the inequality (4) is a Shannon-flow inequality *if and only if* there exists a proof sequence for it.

2.3 Comparison of polymatroid bound to other bounds

Recall the bound $\mathrm{DC}[\mathcal{F}]$ defined in (1). By parameterizing this bound with combinations of $\mathcal{F}$ and DC, we obtain a hierarchy of bounds, which we briefly summarize here. By setting $\mathcal{F} = \Gamma_n$, N_n, M_n, we obtain the *polymatroid bound* $\mathrm{DC}[\Gamma_n]$, the *normal bound* $\mathrm{DC}[\mathsf{N}_n]$, and the *modular bound* $\mathrm{DC}[M_n]$, respectively. For a permutation π of $[n]$, let DC_π denote the collection of degree

constraints obtained by modifying each $(X_i, Y_i, c_i) \in \mathrm{DC}$ by retaining the variables in Y_i that either are in X_i or are listed after all variables in X_i in the π-order. Note that DC_π is a relaxation of DC in the sense that if a database instance satisfies DC, then it also satisfies DC_π. The following inequalities are known [25, 29]:

$$\mathrm{DC}[M_n] \le \mathrm{DC}[\mathsf{N}_n] \le \mathrm{DC}[\Gamma_n] \le \min_\pi \mathrm{DC}_\pi[\Gamma_n] \qquad \mathrm{DC}[M_n] \le n \cdot \log \max_{\boldsymbol{I} \models \mathrm{DC}} |Q(\boldsymbol{I})| \tag{5}$$

$$\log \max_{\boldsymbol{I} \models \mathrm{DC}} |Q(\boldsymbol{I})| \le \mathrm{DC}[\Gamma_n] \qquad \mathrm{DC}[\mathsf{N}_n] \le 2^n \cdot \log \max_{\boldsymbol{I} \models \mathrm{DC}} |Q(\boldsymbol{I})| \tag{6}$$

Fix a permutation π (a "chain"), then the *chain-bound* $\mathrm{DC}_\pi[\Gamma_n]$ is computable in poly-time [25]. If DC is acyclic, then we can compute a permutation π (in poly-time) such that $\mathrm{DC}_\pi[\Gamma_n] = \mathrm{DC}[M_n]$.

3 Computing the polymatroid bound in polynomial time

3.1 Review of the linear programming formulation

This subsection reviews the relevant progress made in [3]. The LP formulation for computing the polymatroid bound $\mathrm{DC}[\Gamma_n]$ on a collection $\mathrm{DC} = \{(X_i, Y_i, c_i) \mid i \in [k]\}$ of degree constraints was shown in equation (2). We now write down this LP more explicitly, by listing all the constraints defining the polymatroids. To make the formulation more symmetrical, in the following, we do not restrict $h(\emptyset) = 0$; instead of maximizing $h([n])$, we maximize the shifted quantity $h([n]) - h(\emptyset)$. The function $h'(X) = h(X) - h(\emptyset)$ is a polymatroid.

There is a variable $h(X)$ for each subset X of $[n]$. The LP, denoted by P, is:

$$\begin{array}{lll} P: & \max & h([n]) - h(\emptyset) \\ \text{s.t.} & h(Y \cup X) - h(X) - h(Y) + h(Y \cap X) \le 0 & \forall X \forall Y, X \perp Y \\ & h(Y) - h(X) \ge 0 & \forall X \forall Y, X \subsetneq Y \\ & h(Y_i) - h(X_i) \le c_i & \forall i \in [k] \end{array} \tag{7}$$

where $X \perp Y$ means $X \not\subseteq Y$ and $Y \not\subseteq X$. The first collection of constraints enforce that the function h is submodular, the second enforces that the function h is monotone, and the third enforces the degree constraints. We will adopt the convention that all variables in our LPs are constrained to be *non-negative* unless explicitly mentioned otherwise. Note that the linear program P has both exponentially many variables and exponentially many constraints.

To formulate the dual LP D of P, we associate dual variables $\sigma_{X,Y}$, dual variables $\mu_{X,Y}$ and dual variables δ_i with the three types of constraints in P (in that order). The dual D is then:

$$\begin{array}{lll} D: & \min & \sum_{i \in [k]} c_i \cdot \delta_i \\ & \text{s.t.} & \mathrm{excess}([n]) \ge 1 \\ & & \mathrm{excess}(\emptyset) \ge -1 \\ & & \mathrm{excess}(Z) \ge 0, \qquad \forall Z \ne \emptyset, [n] \end{array} \tag{8}$$

where $\mathrm{excess}(Z)$ is defined by:

$$\mathrm{excess}(Z) := \sum_{i:Z=Y_i} \delta_i - \sum_{i:Z=X_i} \delta_i + \sum_{\substack{I \perp J \\ I \cap J = Z}} \sigma_{I,J} + \sum_{\substack{I' \perp J' \\ I' \cup J' = Z}} \sigma_{I',J'} - \sum_{J:J \perp Z} \sigma_{Z,J} - \sum_{X:X \subsetneq Z} \mu_{X,Z} + \sum_{Y:Z \subsetneq Y} \mu_{Z,Y}$$

We use $\boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\mu}$ to denote the vectors of δ_i, $\sigma_{X,Y}$, and $\mu_{X,Y}$ variables.

The dual D can be interpreted as encoding a min-cost flow problem on a hypergraph $\mathcal{L}$. The vertices of $\mathcal{L}$ are the subsets of $[n]$. Each variable $\mu_{X,Y}$ in D represents the flow on a directed edge in $\mathcal{L}$ from Y to X. Each variable δ_i in D represents both the flow and capacity on the directed edge from X_i to Y_i in $\mathcal{L}$. The cost to buy this capacity is $\delta_i \cdot c_i$. Each variable $\sigma_{X,Y}$ in D represents the flow leaving each of X and Y, and entering each of $X \cap Y$ and $X \cup Y$ through the hyperedge

containing four vertices. The σ variables involve flow between four nodes, which is why this is a hypergraph flow problem, not a graph flow problem. Flow can not be created at any vertex other than the empty set, and is conserved at all vertices other than the empty set and $[n]$. That is, there is flow conservation at each vertex like a standard flow problem. The objective is to minimize the cost of the bought capacity subject to the constraint that this capacity can support a unit of flow from the source $s = \emptyset$ to the sink $t = [n]$.

Example 1 (Running Example Instance)**.** It is challenging to construct a small example that illustrates all the interesting aspects of our algorithm design, but the following collection of degree constraints is sufficient to illustrate many interesting aspects:

$$(1)\ h(ab) \le 1; \quad (2)\ h(bc) \le 2; \quad (3)\ h(ac) \le 1; \quad (4)\ h(ad) - h(a) \le 1$$

These degree constraints are indexed (1) through (4) as indicated and $k = 4$. The optimal solution for D to buy the following capacities, and route flow as follows:

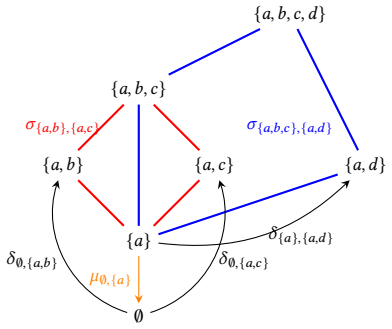

Fig. 2. The feasible flow in the optimal solution for D, where all depicted variables are set to 1.

Set $\delta_1, \delta_3, \delta_4$ equal to 1 and δ_2 to 0. The objective will be 3. This can support the flow as follows. A flow of 1 is sent from $\emptyset$ to $\{a, b\}$ and from $\emptyset$ to $\{a, c\}$ on the corresponding degree constraints. Then a flow of 1 is sent from $\{a, b\}$ and $\{a, c\}$ to both $\{a, b, c\}$ and $\{a\}$ using $\sigma_{\{a,b\},\{a,c\}}$. The flow of 1 at $\{a\}$ is sent to $\{a, d\}$ on the degree constraint δ_4. A flow of 1 is then sent to both $\{a, b, c, d\}$ and $\{a, \}$ from $\{a, b, c\}$ and $\{a, d\}$ using $\sigma_{\{a,b,c\},\{a,d\}}$. Thus, a flow of 1 reaches the set of all elements. Finally the leftover flow of 1 at $\{a\}$ is returned to $\emptyset$ on $\mu_{\emptyset,\{a\}}$.

3.2 Polynomial sized linear programming formulation (proof of Theorem 1.1)

This is where we leave the foundation laid by [3] and begin covering new ground. We shall show that the LP D can be replaced by projecting its feasible region down to the space of the δ_i variables, resulting in the following LP:

$$D_{\boldsymbol{\delta}}^{\text{simple}}: \qquad \min \quad \sum_{i\in[k]} c_i \cdot \delta_i \tag{9}$$

$$\text{s.t.} \quad \sum_{i\in[k]:\ X_i\subseteq V, Y_i\not\subseteq V} \delta_i \ge 1 \qquad \forall V \subsetneq [n]$$

$D_{\boldsymbol{\delta}}^{\text{simple}}$ can be interpreted as a min-cost cut problem in the hypergraph network $\mathcal{L}$, where the goal is to cheaply buy sufficient capacity on the δ edges so the δ edges crossing various cuts, determined by the degree constraints, have in aggregate at least unit capacity. Note that $D_{\boldsymbol{\delta}}^{\text{simple}}$ has only linearly many variables (one for each degree constraint), but exponentially many constraints.

We next present a linear program D_{δ}^{flow} that shall be shown to be equivalent to D and $D_{\delta}^{\text{simple}}$, but with only polynomially many variables and only polynomially many constraints. The intuition behind D_{δ}^{flow} is that it has a natural interpretation as a min-cost flow problem in a (directed graph) network $G = (V, E)$ with a single source $s = \emptyset$ and one sink for each variable. The vertices in V consist of the empty set $\emptyset$, the singleton sets $\{i\}$ for $i \in [n]$, and the sets Y_i, $i \in [k]$. The edges in E are all the degree constraint edges (X_i, Y_i) from $\mathcal{L}$, and all the $\mu_{X,Y}$ edges from $\mathcal{L}$ where X and Y are vertices in V. The cost of the degree constraint edges (X_i, Y_i) remains δ_i, and the costs of the $\mu_{X,Y}$ edges remain 0. So G is a subgraph of the hypergraph $\mathcal{L}$. See Figure 3 for an illustration of G for our running example. Now the problem is to spend as little as possible to buy enough capacity so that for all sinks/vertices $t \in [n]$ it is the case that there is sufficient capacity to route a unit of flow from the source $s = \emptyset$ to the sink t.

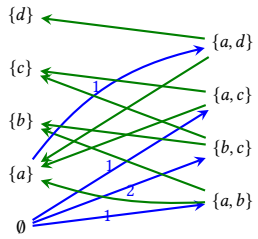

Fig. 3. The auxiliary graph G for the running example. The blue edges correspond to the degree constraints, with annotated costs. The green edges correspond to the μ variables, and cost 0.

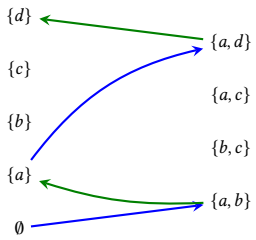

Fig. 4. A unit flow in the auxiliary graph G from $\emptyset$ to $\{d\}$. For this flow to be feasible, a unit capacity must be bought on these edges.

This problem is naturally modeled by the following linear program D_{δ}^{flow}:

$$
\begin{aligned}
D_{\delta}^{\text{flow}}: \quad & \min \sum_{i \in [k]} c_i \cdot \delta_i && && && (10)\\
\text{s.t.} \quad & f_{i,t} \le \delta_i && \forall i \in [k] && \forall t \in [n]\\
& \text{excess}_t(t) = 1 && && \forall t \in [n]\\
& \text{excess}_t(\emptyset) = -1 && && \forall t \in [n]\\
& \text{excess}_t(Z) \ge 0 && \forall Z \in G \setminus \{\emptyset\} \setminus \{t\} && \forall t \in [n]
\end{aligned}
$$

where,

$$\text{excess}_t(Z) := \sum_{i: Z = Y_i} f_{i,t} - \sum_{i: Z = X_i} f_{i,t} + \sum_{X: X \subsetneq Z} \mu_{X,Z,t} + \sum_{Y: Z \subsetneq Y} \mu_{Z,Y,t}$$

The interpretation of $f_{i,t}$ is the flow routed from X_i to Y_i in G for the flow problem where the sink is $\{t\}$; $\mu_{Z,Y,t}$ is the flow routed from Y to Z in G. So the first set of constraints say that the capacity bounds are respected, the second and third set of constraints ensure that unit flow leaves the source and arrives at the appropriate sink, and the last set of constraints ensure flow conservation. Note that as the graph G has $O(k + n)$ vertices and $O(kn)$ edges, the LP D_{δ}^{flow} has $O(kn^2)$ variables and $O(kn)$ constraints.

Example 2 (Running example)**.** The optimal solution for D_{δ}^{flow} on our running example is to buy unit capacity on the μ edges for a cost of 0, buy unit capacity on degree constraint edges $\emptyset \to ab$, $\emptyset \to ac$, and $a \to ad$ for a cost of 1 each, resulting in a total cost of 3. This supports a flow of 1 to a by routing a unit of flow on the path $\emptyset \to ab \to a$, supports a flow of 1 to b by routing a unit of flow

on the path $\emptyset \to ab \to b$, supports a flow of 1 to c by routing a unit of flow on the path $\emptyset \to ac \to c$, and supports a flow of 1 to d by routing a unit of flow on the path $\emptyset \to ab \to a \to ad \to d$. This flow to d is shown in Figure 4.

We now formally show that the linear programs D, $D^{\text{simple}}_{\boldsymbol{\delta}}$ and $D^{\text{flow}}_{\boldsymbol{\delta}}$ all have the same objective function. Refer to Fig. 1 for the high-level plan. To show that they are equivalent, it is sufficient to just consider the feasible regions for these LPs. We prove equivalence in the following manner. Lemma 3.1 shows feasible regions of the LPs $D^{\text{simple}}_{\boldsymbol{\delta}}$ and $D^{\text{flow}}_{\boldsymbol{\delta}}$ are identical. Then Lemma 3.2 shows that the polyhedron defined by projecting the feasible region of the LP D onto the $\boldsymbol{\delta}$-space is a subset of the feasible region defined by the LP $D^{\text{flow}}_{\boldsymbol{\delta}}$. Finally Lemma 3.3 shows that polyhedron defined by projecting the feasible region of the LP D onto the $\boldsymbol{\delta}$-space is a superset of the feasible region defined by the LP $D^{\text{flow}}_{\boldsymbol{\delta}}$.

Lemma 3.1. *The feasible regions of the linear programs $D^{\text{simple}}_{\boldsymbol{\delta}}$ and $D^{\text{flow}}_{\boldsymbol{\delta}}$ are identical.*

Proof. Assume for some setting of the δ_i variables, that $D^{\text{flow}}_{\boldsymbol{\delta}}$ is infeasible. Then there exists a $t \in [n]$ such that the max flow between the source $s = \emptyset$ and $\{t\}$ is less than 1. Since the value of the maximum s-t flow is equal the value of the minimum s-t cut, there must be a subset W of vertices in G such that $s \in W$ and $t \notin W$, where the aggregate capacities leaving W is less than one. By taking $V := \{i \in [n] \mid \{i\} \in W\}$ we obtain a violated constraint for $D^{\text{simple}}_{\boldsymbol{\delta}}$.

Conversely, assume that for some setting of the δ_i variables, that $D^{\text{simple}}_{\boldsymbol{\delta}}$ is infeasible. Then there is a set $V \subsetneq [n]$ such that $\sum_{i\in[k]:X_i\subseteq V, Y_i\not\subseteq V} \delta_i < 1$. Fix an arbitrary $t \in [n] \setminus V$, and let $W := \{s\} \cup \{\{i\} \mid i \in V\}$. The (s,t)-cut $(W, V(G) \setminus W)$ has capacity $\sum_{i\in[k]:X_i\subseteq V, Y_i\not\subseteq V} \delta_i$ which is strictly less than 1. This means $\boldsymbol{\delta}$ is not feasible for $D^{\text{flow}}_{\boldsymbol{\delta}}$, a contradiction. □

Lemma 3.2. *The polyhedron defined by projecting the feasible region of the linear program D onto the $\boldsymbol{\delta}$-space is a subset of the feasible region defined by the linear program $D^{\text{flow}}_{\boldsymbol{\delta}}$.*

Proof. Let $(\boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\mu})$ be a feasible solution to the linear program D, where $\boldsymbol{\delta} = (\delta_i)_{i\in[k]}$. We show that $\boldsymbol{\delta}$ is feasible for the linear program $D^{\text{flow}}_{\boldsymbol{\delta}}$. Assume to the contrary that $\boldsymbol{\delta}$ is *not* feasible for $D^{\text{flow}}_{\boldsymbol{\delta}}$, then there exists a $t \in [n]$ such that there is a cut in the flow network G with capacity < 1 that separates $s = \emptyset$ and $\{t\}$. In particular, let V be the union of all singleton sets that are on the same side of this cut as $s = \emptyset$; then, $\sum_{i\in[k]:\ X_i\subseteq V, Y_i\not\subseteq V} \delta_i < 1$.

Now consider the flow hypernetwork $\mathcal{L}$ associated with the linear program D. Let $W := \{s\} \cup \{\{i\} \mid i \in V\}$ be a set of vertices in $\mathcal{L}$. Then in the hypergraph $\mathcal{L}$ we claim that the aggregate flow coming out of W can be at most $\sum_{i\in[k]:\ X_i\subseteq V, Y_i\not\subseteq V} \delta_i < 1$. Since the net flow out of W is equal to the total flow received by vertices not in W, which is $\sum_{Z\not\in W} \text{excess}(Z) \geq 1$, we reach a contradiction.

To see that the claim holds, note that no $\mu_{X,Y}$ edge can cause flow to escape W, and no $\sigma_{X,Y}$ hyperedge can contribute a positive flow to leave W: if $X \in W$ and $Y \in W$ then $X \cup Y \in W$, and if either $X \notin W$ or $Y \notin W$ then $\sigma_{X,Y}$ does not route any positive net flow out of W. □

Lemma 3.3. *The polyhedron defined by projecting the feasible region of the linear program D onto the $\boldsymbol{\delta}$-space is a superset of the feasible region defined by the linear program $D^{\text{flow}}_{\boldsymbol{\delta}}$.*

Proof. To prove this lemma we constructively show how to extend a feasible solution $\boldsymbol{\delta}$ for $D^{\text{flow}}_{\boldsymbol{\delta}}$ to a feasible solution $(\boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\mu})$ for D by setting $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ variables. The extension is shown in Algorithm 1. After initialization, the outer loop iterates over i from 0 to $n-1$. We shall show that this loop maintains the following *outer loop invariant* on the setting of the variables in D, at the end of iteration i

(1) The excess at the vertex $[i+1]$ in $\mathcal{L}$ is 1.

Algorithm 1 Constructing a feasible solution $(\boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\mu})$ to the LP D

$\boldsymbol{\mu} \leftarrow \mathbf{0}$, $\boldsymbol{\sigma} \leftarrow \mathbf{0}$
for $j = 1$ to k **do** ▷ For each degree constraint (X_j, Y_j, c_j)
  $\mu_{X_j,Y_j} \leftarrow \delta_j$
for $i = 0$ to $n-1$ **do** ▷ Outer Loop
  Let $\mathcal{P}^{i+1}$ be the collection of simple flow paths routing 1 unit of flow from $s = \emptyset$ to $t = \{i+1\}$
  for each path $P \in \mathcal{P}^{i+1}$ with flow value ϵ **do** ▷ Forward Path Loop
    for each edge $(A, B) \in P$ from $\emptyset$ to $\{i+1\}$ **do**
      if $A \cup [i] \subsetneq B \cup [i]$ **then**
        Decrease $\mu_{A\cup[i],B\cup[i]}$ by ϵ
      else if $B \cup [i] \subsetneq A \cup [i]$ **then**
        Increase $\mu_{B\cup[i],A\cup[i]}$ by ϵ
  for each path $P \in \mathcal{P}^{i+1}$ with flow value ϵ **do** ▷ Backward Path Loop
    for each edge $(A, B) \in P$ from $\{i+1\}$ back to $\emptyset$ **do**
      if $A \cup [i+1] \subsetneq B \cup [i+1]$ **then**
        Increase $\mu_{A\cup[i+1],B\cup[i+1]}$ by ϵ
      else if $B \cup [i+1] \subsetneq A \cup [i+1]$ **then**
        Decrease $\mu_{B\cup[i],A\cup[i]}$ by ϵ
        if $i+1 \in A$ **then** ▷ This means $A \cup [i] = A \cup [i+1]$
          Increase $\mu_{B\cup[i],B\cup[i+1]}$ by ϵ
        else
          Increase $\sigma_{A\cup[i],B\cup[i+1]}$ by ϵ
  for each $j \in [k]$, where $X_j \cup [i+1] \subsetneq Y_j \cup [i+1]$ **do** ▷ Cleanup Loop
    $\epsilon \leftarrow \delta_j - f_{j,i+1}$
    Reduce $\mu_{X_j\cup[i],Y_j\cup[i]}$ by ϵ
    if $i+1 \in Y_j$ **then**
      Increase $\mu_{X_j\cup[i],X_j\cup[i+1]}$ and $\mu_{X_j\cup[i+1],Y_j\cup[i+1]}$ by ϵ each
    else
      Increase $\sigma_{X_j\cup[i+1],Y_j\cup[i]}$ by ϵ

(2) The excess at every vertex in $\mathcal{L}$, besides $\emptyset$ and $[i+1]$ is zero.
(3) For every $j \in [k]$, if $X_j \cup [i+1] \subsetneq Y_j \cup [i+1]$, then we have $\mu_{X_j\cup[i+1],Y_j\cup[i+1]} = \delta_j$.
(4) $\boldsymbol{\mu}, \boldsymbol{\sigma} \geq \mathbf{0}$.

Note that, if the invariant is satisfied at the end of iteration $i = n-1$, then the resulting $(\boldsymbol{\delta}, \boldsymbol{\sigma}, \boldsymbol{\mu})$ will represent a feasible solution for the LP D, which proves the lemma.

We now prove the invariant by induction on i. It is satisfied at $i = 0$, where $[0]$ is defined to be the empty set. For the inductive step, note that the body of the outer loop has three blocks of inner loops: the forward path loop, the backward path loop, and the cleanup loop. To extend the inductive hypothesis from i to $i+1$, let $P \in \mathcal{P}^{i+1}$ be a simple flow path in the graph G that routes ϵ-amount of flow from $s = \emptyset$ to $t = \{i+1\}$. The first claim examines the effect of the forward path loop on the variables in D.

Claim 1. The net effect on an iteration of the forward path loop processing a path P, with flow ϵ, on the setting of the variables in D is:

(a) The excess at the vertex $[i]$ in $\mathcal{L}$ is reduced by ϵ, and at the vertex $[i+1]$ is increased by ϵ.
(b) The excess at every vertex in $\mathcal{L}$, besides $\emptyset$, $[i]$ and $[i+1]$ is zero.

(c) For each degree constraint $j \in [k]$ where $(X_j, Y_j) = (A, B) \in P$, and $A \cup [i] \subsetneq B \cup [i]$, it is the case that $\mu_{[i]\cup X_j,[i]\cup Y_j}$ decreases by ϵ.
(d) $\boldsymbol{\mu}, \boldsymbol{\sigma} \geq \mathbf{0}$.

While the algorithm examines each edge (A, B) in P one by one, the real changes are on the "lifted" edge $(A \cup [i], B \cup [i])$. In particular, the algorithm examines the path $P[i]$ constructed from P by replacing each edge (A, B) by $(A \cup [i], B \cup [i])$. An illustration of the paths P, $P[i]$, $P[i+1]$, and flow value settings is shown in Figure 5. Note that $P[i]$ starts from $[i]$ and ends at $[i+1]$;

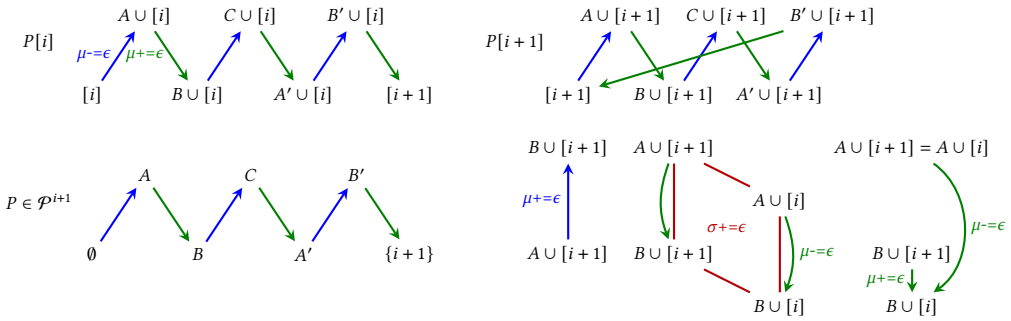

Fig. 5. Illustrations of Forward and backward passes

furthermore, if $A \cup [i] = B \cup [i]$ (i.e. a self-loop) then the edge is not processed. Statements (a) and (b) follow from the fact that, as we enter a vertex on the path $P[i]$, we either increase or decrease the excess by ϵ, only to decrease or increase it by ϵ when processing the very next edge. The only exceptions are the starting vertex $[i]$ which loses ϵ excess, and the ending vertex $[i+1]$ which gains ϵ excess. Statement (c) follows trivially from line 9 of the algorithm. Part (d) follows from the induction hypothesis that condition (3) of the outer loop invariant holds for iteration i.

Thus the aggregate effect of the forward path loop (after all paths are processed) is to increase the excess at $[i+1]$ from 0 to 1, and to decrease the excess at $[i]$ from from 1 to 0. This establishes the first two conditions of the outer loop invariant for iteration $i+1$. The purpose of the backward path loop and the cleanup loop is to establish the 3rd condition of the outer loop invariant for iteration $i+1$. We always maintain $\boldsymbol{\sigma}, \boldsymbol{\mu} \geq \mathbf{0}$ throughout. The backward path loop increases such a $\mu_{X_j\cup[i+1],Y_j\cup[i+1]}$ to the flow value $f_{j,i+1}$, where the value of $f_{j,i+1}$ comes from the optimal solution to the linear program $D^{\text{flow}}_{\boldsymbol{\delta}}$. This can still be smaller than δ_j. The cleanup loop then increases $\mu_{X_j\cup[i+1],Y_j\cup[i+1]}$ to δ_j, giving the desired property.

The next claim examines the effect of the backward path loop on the variables in D.

Claim 2. The net effect of an iteration of the backward path loop processing a path P is:

(a) The excess of all nodes in $\mathcal{L}$ does not change.
(b) If there is a degree constraint j where $(X_j, Y_j) = (A, B) \in P$, and $X_j \cup [i+1] \neq Y_j \cup [i+1]$ then $\mu_{A\cup[i+1],B\cup[i+1]}$ will increase by ϵ.
(c) $\boldsymbol{\mu}, \boldsymbol{\sigma} \geq \mathbf{0}$.

Let $P[i+1]$ be the lifted path constructed from P by replacing each edge (A, B) by an edge $(A \cup [i+1], B \cup [i+1])$, and removing all self loops. The backward path loop processes edges in $P[i+1]$. Note that $P[i+1]$ is a closed loop as both the start and the end are $[i+1]$. Thus it will be sufficient to argue that for each *processed* edge (A, B) in P it is the case that the excess of $A \cup [i+1]$ increases by ϵ, the excess of $B \cup [i+1]$ decreases by ϵ, the excess of all other nodes does not change,

and if there is a degree constraint j where $A = X_j$, $B = Y_j$, and $X_j \cup [i+1] \neq Y_j \cup [i+1]$ then $\mu_{A\cup[i+1],B\cup[i+1]}$ will increase by ϵ.

First consider the case that $A \cup [i+1] \subsetneq B \cup [i+1]$, and thus there is a degree constraint j where $(A, B) = (X_j, Y_j)$. Since we increase $\mu_{A\cup[i+1],B\cup[i+1]}$ by ϵ in line 15, it follows that the excess of $A \cup [i+1]$ increases by ϵ and the excess of $B \cup [i+1]$ decreases by ϵ. The excess of all other nodes does not change. Furthermore, (b) and (c) hold for this case.

Second, consider the case that $B \cup [i+1] \subsetneq A \cup [i+1]$. Decreasing $\mu_{B\cup[i],A\cup[i]}$ by ϵ in line 15 has the effect of increasing the excess of $A \cup [i]$ and decreasing the excess of $B \cup [i]$. Figure 5 illustrates the following cases. If $i+1 \in A$ then, since $A \cup [i] = A \cup [i+1]$ we have an excess increase of ϵ at $A \cup [i+1]$. Increasing $\mu_{B\cup[i],B\cup[i+1]}$ by ϵ in line 19 balances the excess at $B \cup [i]$ and reduces the excess at $B \cup [i+1]$ by ϵ. If $i+1 \notin A$, then $(B \cup [i+1]) \cap A \cup [i] = B \cup [i]$ and $(B \cup [i+1]) \cup (A \cup [i]) = A \cup [i+1]$. In this case, increasing $\sigma_{A\cup[i],B\cup[i+1]}$ by ϵ in line 21 balances the excess changes at $A \cup [i]$ and $B \cup [i]$ (due to the decrease in $\mu_{B\cup[i],A\cup[i]}$), increases the excess at $A \cup [i+1]$, and decreases the excess at $B \cup [i+1]$, as desired.

In summary, after the backward path loop, for each degree constraint $j \in [k]$, where $X_j \cup [i+1] \subsetneq Y_j \cup [i+1]$, it is the case that $\mu_{X_j\cup[i+1],Y_j\cup[i+1]} = f_{j,i+1}$, and the value of $\mu_{X_j\cup[i],Y_j\cup[i]}$ is what remains, which is $\delta_j - f_{j,i+1}$. To bring it up to δ_j, in the cleanup loop we iterate over each degree constraint $j \in [k]$ where $f_{j,i+1} < \delta_j$ and adjust the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ variables accordingly. The analysis is analogous to the analysis of the backward path loop above. □

Example 3 (Running example – constructing a feasible solution for D)**.** Order the nodes as (1) a, (2) b, (3) c, (4) d. Briefly, we discuss iterations $i = 0$ and $i = 3$. Recall that the optimal solution sets δ_1, δ_3 and δ_4 all to one and initially these variables are one in the lattice. Before the outer loop, $\mu_{\emptyset,\{a,b\}}, \mu_{\emptyset,\{a,c\}}$ and $\mu_{\{a\},\{a,d\}}$ are all also set to one.

First Iteration: Consider the outer loop where $i = 0$. In the auxiliary graph, the flow to $\{a\}$ is a single path $\emptyset, \{ab\}, \{ab\}, \{a\}$. During the forward path loop, the variable $\mu_{\emptyset,\{a,b\}}$ decreases by 1 and $\mu_{\{a,b\},\{a\}}$ increases by one. In the backward path loop, $\mu_{\{a\},\{a,b\}}$ decreases by one and then increases again by one. During the cleanup loop, the degree constraint $(\emptyset, \{a, c\})$ results in $\mu_{\emptyset,\{a,c\}}$ decreases by one. Then $\mu_{\emptyset,\{a\}}$ and $\mu_{\{a\},\{a,c\}}$ increase by one. Next consider the degree constraint, $(\{a\}, \{a, d\})$. The variable $\mu_{\{a\},\{a,d\}}$ decreases by one and then immediately increase by one again.

Last Iteration: In this case $[i]$ is $\{a, b, c\}$. Just before the outer loop, $\mu_{\{a,b,c\},\{a,b,c,d\}}$ is set to one, corresponding to δ_4 and the excess at vertex $\{a, b, c\}$ is one. In the auxiliary graph, the flow to $\{d\}$ a single path $\emptyset, \{ab\}, \{ab\}, \{a\}, \{a\}, \{ad\}, \{a, d\}, \{d\}$. In the forward pass, nothing changes when processing the edges $\emptyset, \{ab\}$ and $\{ab\}, \{a\}$.[3] When $\{a\}, \{ad\}$ is processed, $\mu_{\{a,b,c\},\{a,b,c,d\}}$ decreases by 1. Nothing changes when processing the last edge of the path and $[i+1]$ is the universe $\{a, b, c, d\}$ so nothing changes in the backward or cleanup loops. This effectively gets a flow of 1 to the universe.

3.3 Simple degree constraints and the normal bound

An interesting consequence of our approach is the following result, first proved in [1].

Proposition 3.1. *If the input degree constraints are simple, then* $\mathsf{DC}[\Gamma_n] = \mathsf{DC}[\mathsf{N}_n]$.

Proof. The dual LP of $D_{\boldsymbol{\delta}}^{\text{simple}}$ (9) is

$$P_{\boldsymbol{\delta}}^{\text{simple}}: \quad \max \sum_{V\subset[n]} \lambda_V \qquad (11)$$
$$\text{s.t.} \sum_{V\subset[n]:\, X_i\subseteq V, Y_i\not\subseteq V} \lambda_V \le c_i, \qquad \forall i \in [k]$$

[3]There is no change because $\emptyset \cup [i] = \{ab\} \cup [i] = \{a\} \cup [i]$.

From the results of Section 3.2, we know that the LP $P_{\boldsymbol{\delta}}^{\text{simple}}$ shown in (11) has optimal value equal $\mathrm{DC}[\Gamma_n]$. We prove that this LP is the same as $\mathrm{DC}[\mathsf{N}_n]$. Recall that $h \in \mathsf{N}_n$ iff $h = \sum_V \lambda_V s_V$, for some non-negative λ_V, and where s_V is the step function defined by $s_V(X) := \mathbf{1}_{X \not\subseteq V}$ (the indicator function for the event $X \subseteq V$). It follows that (11) is *exactly* $\mathrm{DC}[\mathsf{N}_n]$, because

$$h(X) = \sum_{V: X \not\subseteq V} \lambda_V \qquad h([n]) = \sum_{V \subset [n]} \lambda_V \qquad h(Y) - h(X) = \sum_{V: X \subseteq V, Y \not\subseteq V} \lambda_V. \qquad X \subseteq Y$$

In terms of weighted coverage function [20], λ_V is the weight of the vertex connected to vertices in V in the standard bipartite representation (WLOG we assume that there is only one such vertex). □

4 Computing a polynomial sized proof-sequence for simple instances

This section proves Theorem 1.2 by constructing, from a feasible solution to the linear program $D_{\boldsymbol{\delta}}^{\text{flow}}$, a proof sequence for the Shannon-flow inequality $h([n]|\emptyset) \le \sum_{j \in [k]} \delta_j \cdot h(Y|X)$. The construction is given in Algorithm 2, whose execution traces the construction of the σ and μ variables from the δ variables in Algorithm 1. We now show in Lemma 4.1 an invariant of Algorithm 2 that is sufficient to show that the proof sequence is correct. To see this note that this invariant implies that in the end the excess at $[n]$ is 1.

Lemma 4.1. *Starting from the sum $\sum_{j \in [k]} \delta_j \cdot h(Y_j|X_j)$, the proof sequence constructed in Algorithm 2 satisfies the following invariants. If we were to run Algorithm 1 in lock step with Algorithm 2, then after every edge (A, B) is processed the following holds:*

(a) The coefficient of $h(Y \mid \emptyset)$ is the excess at Y, for every $Y \subseteq [n]$.
(b) The coefficient of $h(Y \mid X)$ is exactly $\mu_{X,Y}$, for every $\emptyset \neq X \subseteq Y \subseteq [n]$.

Proof. We prove the lemma by induction. Initially, when we initialize $\boldsymbol{\mu}$ in line 3 of Algorithm 1, the invariants hold trivially. We verify that the invariant holds after each proof step. Please also refer to the proof of Lemma 3.3 as we need to run the two proofs in parallel.

In the forward path loop, we traverse edges $(A \cup [i], B \cup [i])$ of $P[i]$ from $\emptyset$ to $\{i+1\}$. When $A \cup [i] \subsetneq B \cup [i]$, in Algorithm 1 we decrease $\mu_{A\cup[i],B\cup[i]}$ by ϵ, increase the excess at $B \cup [i]$ by ϵ, reduce the excess at $A \cup [i]$ by ϵ, which correspond precisely to ϵ-composing $h(A \cup [i]) + h(B \cup [i] \mid A \cup [i]) \to h(B \cup [i])$ in Algorithm 2. The case when $B \cup [i] \subsetneq A \cup [i]$ is the converse.

In the backward path loop, we traverse edges $(A \cup [i+1], B \cup [i+1])$ of $P[i+1]$ from $i+1$ back to $\emptyset$. For $A \cup [i+1] \subsetneq B \cup [i+1]$, in Algorithm 1 we increase $\mu_{A\cup[i+1],B\cup[i+1]}$, decrease the excess at $B \cup [i+1]$, and increase the excess at $A \cup [i+1]$ by ϵ. These correspond precisely to ϵ-decomposing $h(B \cup [i+1]) \to h(A \cup [i+1]) + h(B \cup [i+1] \mid A \cup [i+1])$ in Algorithm 2. When $B \cup [i+1] \subsetneq A \cup [i+1]$, there are two cases. For $i+1 \in A$, Algorithm 1 decreases $\mu_{B\cup[i],A\cup[i]}$ increases $\mu_{B\cup[i],B\cup[i+1]}$, reduces the excess at $B \cup [i+1]$, and increases the excess at $A \cup [i+1]$ by ϵ. This corresponds to the ϵ-decomposing step $h(B \cup [i+1]) \to h(B \cup [i]) + h(B \cup [i+1] \mid B \cup [i])$ and the ϵ-composing step $h(B \cup [i]) + h(A \cup [i] \mid B \cup [i]) \to h(A \cup [i]) = h(A \cup [i+1])$ in Algorithm 2. For $i+1 \notin A$, Algorithm 1 increases $\sigma_{A\cup[i],B\cup[i+1]}$, reduces $\mu_{B\cup[i],A\cup[i]}$, reduces the excess at $B \cup [i+1]$, and increases the excess at $A \cup [i+1]$ by ϵ. This corresponds to the ϵ-submodularity and ϵ-compose steps as shown in lines 20 and 21 of Algorithm 2.

Lastly, the cleanup loop is self-explanatory, designed to maintain the invariants. □

Note that as the graph G has $O(k+n)$ vertices and $O(kn)$ edges, one can assume without loss of generality that the cardinality of $\mathcal{P}^{i+1}$ is $O(kn)$, and the length of each path $P \in \mathcal{P}^t$ is $O(k+n)$. As each edge $e \in P$ introduces $O(1)$ steps to the proof sequence when P is processed, and there are at most n choices for the sink t, we can conclude that the length of the resulting proof sequence is $O((k+n)kn^2)$, or equivalently $O(k^2n^2 + kn^3)$.

Algorithm 2 Constructing a proof sequence from $\boldsymbol{\delta}$ feasible to $D^{\text{flow}}_{\boldsymbol{\delta}}$

(We write $h(X)$ to mean $h(X \mid \emptyset)$ for short)
for $i = 0$ to $n-1$ **do** ▷ Outer Loop
  Let $\mathcal{P}^{i+1}$ be the collection of simple flow paths routing 1 unit of flow from $s = \emptyset$ to $t = \{i+1\}$
  for each path $P \in \mathcal{P}^{i+1}$ with flow value ϵ **do** ▷ Forward Path Loop
    for each edge $(A, B) \in P$ from $\emptyset$ to $\{i+1\}$ **do**
      if $A \cup [i] \subsetneq B \cup [i]$ **then**
        ϵ-Compose: $h(A \cup [i]) + h(B \cup [i] \mid A \cup [i]) \to h(B \cup [i])$
      else if $B \cup [i] \subsetneq A \cup [i]$ **then**
        ϵ-Decompose: $h(A \cup [i]) \to h(B \cup [i]) + h(A \cup [i] \mid B \cup [i])$
  for each path $P \in \mathcal{P}^{i+1}$ with flow value ϵ **do** ▷ Backward Path Loop
    for each edge $(A, B) \in P$ from $\{i+1\}$ back to $\emptyset$ **do**
      if $A \cup [i+1] \subsetneq B \cup [i+1]$ **then**
        ϵ-Decompose: $h(B \cup [i+1]) \to h(A \cup [i+1]) + h(B \cup [i+1] \mid A \cup [i+1])$
      else if $B \cup [i+1] \subsetneq A \cup [i+1]$ **then**
        if $i+1 \in A$ **then** ▷ This means $A \cup [i] = A \cup [i+1]$
          if $B \cup [i+1] \neq B \cup [i]$ **then**
            ϵ-Decompose: $h(B \cup [i+1]) \to h(B \cup [i]) + h(B \cup [i+1] \mid B \cup [i])$
          ϵ-Compose: $h(B \cup [i]) + h(A \cup [i] \mid B \cup [i]) \to h(A \cup [i]) = h(A \cup [i+1])$
        else
          ϵ-Submodularity: $h(A \cup [i] \mid B \cup [i]) \to h(A \cup [i+1] \mid B \cup [i+1])$
          ϵ-Compose: $h(B \cup [i+1]) + h(A \cup [i+1] \mid B \cup [i+1]) \to h(A \cup [i+1])$
  for each $j \in [k]$, where $X_j \cup [i+1] \subsetneq Y_j \cup [i+1]$ **do** and $[i+1] \not\subset X$ ▷ Cleanup Loop
    $\epsilon \leftarrow \delta_j - f_{j,i+1}$
    if $i+1 \in Y_j$ **then**
      ϵ-Decompose: $h(Y_j \cup [i] \mid X_j \cup [i]) \to h(Y_j \cup [i] \mid X_j \cup [i+1]) + h(X_j \cup [i+1] \mid X_j \cup [i])$
      ϵ-Monotonicity: $h(X_j \cup [i+1] \mid X_j \cup [i]) \to 0$
    else
      ϵ-Submodularity: $h(X_j \cup [i] \mid Y_j \cup [i]) \to h(X_j \cup [i+1] \mid Y_j \cup [i+1])$

Example 4 (Running example)**.** Algorithm 2 yields the following proof sequence:

$$
\begin{aligned}
& h(\emptyset) + h(ab|\emptyset) + h(ac|\emptyset) + h(ad|a) \\
= \;& h(ab) + h(ac|\emptyset) + h(ad|a) && \text{[Forward pass 1-Compose: } h(ab) = h(\emptyset) + h(ab|\emptyset)\text{]} \\
= \;& h(a) + h(ab|a) + h(ac|\emptyset) + h(ad|a) && \text{[Forward pass 1-Decompose: } h(ab) = h(a) + h(ab|a)\text{]} \\
= \;& h(ab) + h(ac|\emptyset) + h(ad|a) && \text{[Backward pass 1-Compose: } h(a) + h(ab|a) = h(ab)\text{]} \\
= \;& h(a) + h(ab|a) + h(ac|\emptyset) + h(ad|a) && \text{[Backward pass 1-Decompose: } h(ab) = h(a) + h(ab|a)\text{]} \\
\geq \;& h(a) + h(ab|a) + h(ac|a) + h(a|\emptyset) + h(ad|a) && \text{[Clean-up: 1-Decompose: } h(ac|\emptyset) \geq h(ac|a) + h(a|\emptyset)\text{]} \\
\geq \;& h(a) + h(ab|a) + h(ac|a) + h(ad|a) && \text{[Clean-up: 1-Monotonicity: } h(a|\emptyset) \geq 0\text{]} \\
= \;& h(ab) + h(ac|a) + h(ad|a) && \text{[Forward pass 1-Compose: } h(ab) = h(a) + h(ab|a)\text{]} \\
\geq \;& h(ab) + h(abc|ab) + h(ad|a) && \text{[Clean-up: 1-Submodularity: } h(ac|a) \geq h(abc|ab)\text{]} \\
\geq \;& h(ab) + h(abc|ab) + h(abd|ab) && \text{[Clean-up: 1-Submodularity: } h(ad|a) \geq h(abd|ab)\text{]} \\
= \;& h(abc) + h(abd|ab) && \text{[Forward pass 1-Compose: } h(abc) = h(ab) + h(abc|ab)\text{]} \\
\geq \;& h(abc) + h(abcd|abc) && \text{[Clean-up: 1-Submodularity: } h(abd|ab) \geq h(abcd|abc)\text{]}
\end{aligned}
$$

$$\geq \quad h(abcd) \qquad \text{[Forward pass 1-Compose: } h(abcd) = h(abc) + h(abcd|abc)\text{]}$$

5 Lower bounds

In this section we present three classes of seemingly easy instances which turn out to be as hard as general instances. Lemmas 5.1, 5.2, and 5.3 combined would imply Theorem 1.3. Due to page limitations, we only provide a proof sketch here, with the full proofs deferred to Appendix A.

Lemma 5.1. *For the problem of computing the polymatroid bound, an arbitrary instance can be converted into another instance in polynomial time without changing the bound, where the degree constraints is the union of a set of acyclic degree constraints and a set of simple degree constraints (in fact functional dependencies) Further, each FD contains exactly two variables.*

The reduction is as follows. Consider an arbitrary instance I consisting of the universe $U := [n]$ and a set of degree constraints, DC. The new instance I' has $U' := \cup_{i\in[n]}\{x_i, y_i\}$ as universe where x_i and y_i are distinct copies of i and the following set DC$'$ of degree constraints. For each $i \in [n]$, we first add simple degree constraints $(\{x_i\}, \{x_i, y_i\}, 0)$ and $(\{y_i\}, \{x_i, y_i\}, 0)$ to DC$'$. Then for each $(A, B, c) \in$ DC, we create a new degree constraint (A', B', c) by replacing each $i \in A$ with x_i and each $j \in B$ with y_j, and add it to DC$'$. By construction these degree constraints are from $\{x_1, x_2, \cdots, x_n\}$ to $\{y_1, y_2, \cdots, y_n\}$ and therefore are acyclic.

The key observation is that x_i and y_i are indistinguishable in computing the polymatroid bound for I'. We then show that given a polymatroid f achieving the optimum bound for I, we can create a polymatroid g for I' such that $f(U) = g(U')$, and vice versa.

Lemma 5.2. *There is a polynomial-time reduction from a general instance to an instance preserving the polymatroid bound, where for each degree constraint (X, Y, c) we have $|Y| \leq 3$ and $|X| \leq 2$. Further, the new instance satisfies the following:*

- *If $|Y| = 3$, then $|X| = 2$ and $c = 0$.*
- *If $|Y| = 2$, then $|X| = 1$.*

The high-level idea is to repeatedly replace two variables with a new variable in a degree constraint. We first discuss how to choose two variables to combine. Assume there is a degree constraint (X, Y, c) where $|X| > 2$. Then we combine an arbitrary pair of elements in X. If $|X| \leq 2$ for all degree constraints, and there is a degree constraint where $|Y| > 3$, we combine arbitrary two variables in $Y \setminus X$. If there is a degree constraint (X, Y, c) where $|X| = 2$, $|Y| = 3$ and $c > 0$, we combine the two variables in X. Here, we make this replacement in only one degree constraint in each iteration.

Assume we are to combine variables $x, y \in [n]$ into a new variable $z \notin [n]$ (e.g. $z = n + 1$) in a degree constraint $(X, Y, c) \in$ DC. We do so by creating a new degree constraint (X', Y', c) and add it to DC$'$ where

$$(X', Y', c) := \begin{cases} (X \setminus \{x, y\} \cup \{z\}, Y \setminus \{x, y\} \cup \{z\}, c) & \text{if } \{x, y\} \subseteq X \subseteq Y \\ (X, Y \setminus \{x, y\} \cup \{z\}, c) & \text{if } \{x, y\} \subseteq Y \setminus X \end{cases}$$

Further, we add functional dependencies $(\{x, y\}, \{x, y, z\}, 0)$, $(\{z\}, \{x, z\}, 0)$ and $(\{z\}, \{y, z\}, 0)$ to DC$'$, which we call *consistency constraints*. Intuitively, consistency constraints enforce that the variable z and the tuple of variables (x, y) are equivalent. We show this iterative reduction process terminates and preserves the polymatroid bound.

Lemma 5.3. *There is a polynomial-time reduction from a general instance to an instance consisting of a set of simple degree constraints and a set of functional dependency constraints, while preserving the polymatroid bound.*

To show the lemma, we transform a general instance as follows. We repeat the following: If there exist $x_1 \neq x_2 \in X$ for some degree constraint (X, Y, c), replace the constraint with $(X \setminus \{x_1, x_2\} \cup \{x_{12}\}, Y \setminus \{x_1, x_2\} \cup \{x_{12}\}, c)$ and analogously update other constraints; and ii) For consistency, we add functional dependencies $(\{x_1, x_2\}, \{x_1, x_2, x_{12}\}, 0)$ and $(\{x_{12}\}, \{x_1, x_2, x_{12}\}, 0)$.

6 Hardness of computing normal bounds

This section sketches the proof that the normal bound $\mathrm{DC}[\mathsf{N}_n]$ cannot be solved in polynomial time unless P = NP, i.e., Theorem 1.4. The full proof can be found in Appendix B. Recall that a function is normal if it is a non-negative linear combination of step functions, where a step function is defined as $s_V(X) = 0$ if $X \subseteq V$ and 1 otherwise for a subset $V \subsetneq [n]$. To show the hardness result, we consider the dual linear programming formulation, which is exactly $D^{\mathsf{simple}}_{\boldsymbol{\delta}}$ in equation (9). Let $\Delta(\mathrm{DC})$ denote the convex region over δ defined by the constraints in DC. We first show the separation problem is hard.

Lemma 6.1. *Given a set* DC *of degree constraints and a vector* $\hat{\delta} \in \mathbb{R}^{|\mathrm{DC}|}_{\geq 0}$*, checking if* $\hat{\delta} \notin \Delta(\mathrm{DC})$ *is NP-complete. Further, this remains the case under the extra condition that* $\lambda\hat{\delta} \in \Delta(\mathrm{DC})$ *for some* $\lambda > 1$.

We prove this theorem using a reduction from the Hitting Set problem, which is well-known to be NP-complete. In the Hitting Set problem, the input is a set of n elements $E = \{e_1, \ldots, e_n\}$, a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of m subsets of E, and an integer $k > 0$. The answer is true iff there exists a subset L of k elements such that for every set $S_i \in \mathcal{S}$ is 'hit' by the set L chosen, i.e., $L \cap S_i \neq \emptyset$ for all $i \in [m]$.

Lemma 6.2. *There exists a hitting set of size* k *in the original instance* H *if and only if* $\hat{\delta} \notin \Delta(\mathrm{DC})$.

The above lemma shows checking $\hat{\delta} \notin \Delta(\mathrm{DC})$ is NP-hard. While there exist certain relationships among optimization problems, membership problems and their variants [16], in general hardness of the membership problem doesn't necessarily imply hardness of the optimization problem. However, using the special structure of the convex body in consideration, we can show such an implication in our setting, proving the desired hardness result.

7 The flow bound

Deferring the proof of Theorem 1.5 to Appendix C, we give a high-level overview of the bound here. The flow bound $\text{flow-bound}(\mathrm{DC}, \pi)$ is based on a relaxation $\mathrm{DC}^{\mathsf{flow}}_\pi$ of the input degree constraints DC that is less relaxed than the relaxation DC_π used in the chain bound. In particular, every *simple* degree constraint in DC_π is retained in $\mathrm{DC}^{\mathsf{flow}}_\pi$ as is. Let k_s denote the number of simple degree constraints. And, for every non-simple degree constraint $(X, Y, c) \in \mathrm{DC}$, $\mathrm{DC}^{\mathsf{flow}}_\pi$ contains the constraint (X, Y', c) where $Y' \subseteq Y$ contains the variables in Y that come *after* all variables in X in the permutation π (note that this is the same as DC_π). For convenience we reindex the degree constraints in $\mathrm{DC}^{\mathsf{flow}}_\pi$ such that all k_s simple degree constraints appear before any non-simple degree constraints.

The flow bound $\text{flow-bound}(\mathrm{DC}, \pi)$ is then defined by the objective value of the following polynomial-sized linear program:

$$D_{\text{flow}}: \quad \min \sum_{i \in [k]} c_i \cdot \delta_i \tag{12}$$

$$\text{s.t.} \quad f_{i,t} \leq \delta_i \qquad \forall i \in [k_s], \forall t \in [n] \tag{13}$$

$$\text{excess}_t(t) \geq 1 - \sum_{\substack{i \in [k] \setminus [k_s], \\ t \in Y_i - X_i}} \delta_i \qquad \forall t \in [n] \tag{14}$$

$$\text{excess}_t(\emptyset) \geq -1 \qquad \forall t \in [n] \tag{15}$$

$$\text{excess}_t(Z) \geq 0 \qquad \forall Z \in G \setminus \{\emptyset\} \setminus \{t\}, \qquad \forall t \in [n] \tag{16}$$

$$f_{i,t} = 0 \qquad \forall i \in [k_s], t \in [n] \cap (Y_i \setminus X_i) \tag{17}$$

where $\text{excess}_t(Z)$ is defined as follows:

$$\text{excess}_t(Z) := \sum_{i:Z=Y_i} f_{i,t} - \sum_{i:Z=X_i} f_{i,t} + \sum_{X:X\subset Z} \mu_{X,Z,t} + \sum_{Y:Z\subset Y} \mu_{Z,Y,t}$$

As in the dual linear program $D^{\text{flow}}_{\boldsymbol{\delta}}$ for simple degree constraints, intuitively D_{flow} encodes n min-cost flow problems, however the difference is in the constraint (14) that in D_{flow} the demand of node t is reduced (from 1) by an amount equal to the capacity of the non-simple degree constraints that can route flow directly to t. We additionally note that the objective only considers cardinality constraints, simple degree constraints and non-simple constraints that agree with π. Finally, flow is only sent on simple degree constraints and cardinality constraints.

Note that one could modify the linear program for flow-bound$_\pi$(DC) by allowing the "source" for the flow to sink t to not only be the empty set, but also any singleton vertex in $[t-1]$. All of theoretical results would still hold for this modified linear program, but this modified linear program would be better in practice as it would never result in a worse bound, and for some instances it would result in a significantly better bound. Further note that, by our reduction in Appendix A.1, computing $\text{DC}^{\text{flow}}_\pi[\Gamma_n]$, the polymatroid bound on our relaxed degree constraints $\text{DC}^{\text{flow}}_\pi$, is as hard as computing the polymatroid bound on arbitrary instances.

8 Concluding Remarks

Our main contributions are polynomial-time algorithms to compute the polymatroid bound and polynomial length proof sequences for simple degree constraints. These results nudge the information theoretic framework from [2, 3] towards greater practicality. In fact, our technique and the flow-bound from Section 7 were adopted in the recent work of Zhang et al. [31] to make part of their cardinality estimation framework practical.

The main major open problem remains determining the computational complexity of the polymatroid bound. While we proved some negative results regarding the hardness of computing the polymatroid bound beyond simple degree constraints, we should still be looking for other ways to parameterize the input so that the polymatroid bound can be computed in polynomial time.

Acknowledgments

Sungjin Im was supported in part by NSF grants CCF-2423106, CCF-2121745, and CCF-1844939, and Office of Naval Research Award N00014-22-1-2701. Benjamin Moseley was supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair, NSF grants CCF-2121744 and CCF-1845146, and Office of Naval Research Award N00014-22-1-2702. Kirk Pruhs was supported in part by the NSF grants CCF-2209654 and CCF-1907673, and an IBM Faculty Award. Part of this work was conducted while the authors participated in the Fall 2023 Simons Program on Logic and Algorithms in Databases and AI.

References

[1] Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. 2020. Bag Query Containment and Information Theory. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, Dan Suciu, Yufei Tao, and Zhewei Wei (Eds.). ACM, 95–112. https://doi.org/10.1145/3375395.3387645

[2] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2016. Computing Join Queries with Functional Dependencies. In *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*, Tova Milo and Wang-Chiew Tan (Eds.). ACM, 327–342. https://doi.org/10.1145/2902251.2902289

[3] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2017. 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 2017, Chicago, IL, USA, May 14-19, 2017*, Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts (Eds.). ACM, 429–444. https://doi.org/10.1145/3034786.3056105

[4] Noga Alon. 1981. On the number of subgraphs of prescribed type of graphs with a given number of edges. *Israel J. Math.* 38, 1-2 (1981), 116–130. https://doi.org/10.1007/BF02761855

[5] Albert Atserias, Martin Grohe, and Dániel Marx. 2008. Size Bounds and Query Plans for Relational Joins. In *FOCS*. IEEE Computer Society, 739–748.

[6] Béla Bollobás and Andrew Thomason. 1995. Projections of bodies and hereditary properties of hypergraphs. *Bull. London Math. Soc.* 27, 5 (1995), 417–424. https://doi.org/10.1112/blms/27.5.417

[7] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 18–35. https://doi.org/10.1145/3299869.3319894

[8] F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer. 1986. Some intersection theorems for ordered sets and graphs. *J. Combin. Theory Ser. A* 43, 1 (1986), 23–37. https://doi.org/10.1016/0097-3165(86)90019-1

[9] Kyle Deeds, Dan Suciu, Magda Balazinska, and Walter Cai. 2023. Degree Sequence Bound for Join Cardinality Estimation. In *26th International Conference on Database Theory, ICDT 2023, March 28-31, 2023, Ioannina, Greece (LIPIcs, Vol. 255)*, Floris Geerts and Brecht Vandevoort (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:18. https://doi.org/10.4230/LIPICS.ICDT.2023.8

[10] Kyle B. Deeds, Dan Suciu, and Magdalena Balazinska. 2023. SafeBound: A Practical System for Generating Cardinality Bounds. *Proc. ACM Manag. Data* 1, 1 (2023), 53:1–53:26. https://doi.org/10.1145/3588907

[11] Ehud Friedgut. 2004. Hypergraphs, entropy, and inequalities. *Amer. Math. Monthly* 111, 9 (2004), 749–760. https://doi.org/10.2307/4145187

[12] Ehud Friedgut and Jeff Kahn. 1998. On the number of copies of one hypergraph in another. *Israel J. Math.* 105 (1998), 251–256. https://doi.org/10.1007/BF02780332

[13] Tomasz Gogacz and Szymon Torunczyk. 2017. Entropy Bounds for Conjunctive Queries with Functional Dependencies. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy (LIPIcs, Vol. 68)*, Michael Benedikt and Giorgio Orsi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 15:1–15:17. https://doi.org/10.4230/LIPIcs.ICDT.2017.15

[14] Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. 2012. Size and Treewidth Bounds for Conjunctive Queries. *J. ACM* 59, 3 (2012), 16. https://doi.org/10.1145/2220357.2220363

[15] Martin Grohe and Dániel Marx. 2014. Constraint Solving via Fractional Edge Covers. *ACM Transactions on Algorithms* 11, 1 (2014), 4. https://doi.org/10.1145/2636918

[16] Martin Grötschel, László Lovász, and Alexander Schrijver. 2012. *Geometric algorithms and combinatorial optimization*. Vol. 2. Springer Science & Business Media.

[17] Mahmoud Abo Khamis, Kyle Deeds, Dan Olteanu, and Dan Suciu. 2024. Pessimistic Cardinality Estimation. *SIGMOD Rec.* 53, 4 (2024), 1–17. https://doi.org/10.1145/3712311.3712313

[18] Mahmoud Abo Khamis, Vasileios Nakos, Dan Olteanu, and Dan Suciu. 2024. Join Size Bounds using l_p-Norms on Degree Sequences. *Proc. ACM Manag. Data* 2, 2 (2024), 96. https://doi.org/10.1145/3651597

[19] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2024. PANDA: Query Evaluation in Submodular Width. *CoRR* abs/2402.02001 (2024). https://doi.org/10.48550/ARXIV.2402.02001 arXiv:2402.02001

[20] Andreas Krause and Daniel Golovin. 2014. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*, Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli (Eds.). Cambridge University Press, 71–104. https://doi.org/10.1017/CBO9781139177801.004

[21] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215. https://doi.org/10.14778/2850583.2850594

[22] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2018. Query optimization through the looking glass, and what we found running the Join Order Benchmark. *VLDB J.* 27, 5 (2018), 643–668. https://doi.org/10.1007/s00778-017-0480-7

[23] G. Lohman. 2014. Is Query Optimization a Solved Problem? http://wp.sigmod.org/?p=1075.

[24] L. H. Loomis and H. Whitney. 1949. An inequality related to the isoperimetric inequality. *Bull. Amer. Math. Soc* 55 (1949), 961–962.

[25] Hung Q. Ngo. 2018. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, Jan Van den Bussche and Marcelo Arenas (Eds.). ACM, 111–124. https://doi.org/10.1145/3196959.3196990

[26] Hung Q. Ngo. 2022. On an Information Theoretic Approach to Cardinality Estimation (Invited Talk). In *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference) (LIPIcs, Vol. 220)*, Dan Olteanu and Nils Vortmeier (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1:1–1:21. https://doi.org/10.4230/LIPIcs.ICDT.2022.1

[27] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2012. Worst-case optimal join algorithms: [extended abstract]. In *PODS*. 37–48.

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

[29] Dan Suciu. 2023. Applications of Information Inequalities to Database Theory Problems. arXiv:2304.11996 [cs.DB]

[30] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). OpenProceedings.org, 96–106. https://doi.org/10.5441/002/icdt.2014.13

[31] Haozhe Zhang, Christoph Mayer, Mahmoud Abo Khamis, Dan Olteanu, and Dan Suciu. 2025. LpBound: Pessimistic Cardinality Estimation using ℓ_p-Norms of Degree Sequences. *SIGMOD* (2025).

A Proof of Theorem 1.3

We provide the full proof of Lemma 5.1 and Lemma 5.2. The proof of Lemma 5.3 is similar to (and much simpler than) that of Lemma 5.2 and is omitted. This will complete the proof of Theorem 1.3.

A.1 Union of acyclic and simple constraints is still hard

Proof of Lemma 5.1. We first describe the reduction. Suppose we are given an arbitrary instance I consisting of the universe $U := [n]$ and a set of degree constraints, DC. The new instance I' has $U' := \bigcup_{i\in[n]}\{x_i, y_i\}$ as universe where x_i and y_i are distinct copies of i and the following set DC$'$ of constraints. For each $i \in [n]$, we first add the following *simple* functional dependencies to DC$'$:

$$(\{x_i\}, \{x_i, y_i\}, 0) \qquad\qquad (\{y_i\}, \{x_i, y_i\}, 0)$$

Then for each constraint $(A, B, c) \in$ DC, we create a new constraint (A', B', c) by replacing each $i \in A$ with x_i and each $j \in B$ with y_j, and add it to DC$'$. By construction these degree constraints are from $\{x_1, x_2, \cdots, x_n\}$ to $\{y_1, y_2, \cdots, y_n\}$ and therefore are *acyclic*.

The following simple observation states that x_i and y_i are indistinguishable in computing the polymatroid bound for I'.

Claim 3. Let g be a submodular function that satisfies DC$'$ of I'. For any $i \in [n]$ and any $B \subseteq U'$ such that $x_i, y_i \notin B$, we have $g(B \cup \{x_i\}) = g(B \cup \{y_i\}) = g(B \cup \{x_i, y_i\})$.

By monotonicity and submodularity of g, and an FD in DC$'$ involving x_i, y_i, we have:

$$0 \le g(B \cup \{x_i, y_i\}) - g(B \cup \{x_i\}) \le g(\{x_i, y_i\}) - g(\{x_i\}) \le 0.$$

This proves $g(B \cup \{x_i, y_i\}) = g(B \cup \{x_i\})$. The other equality $g(B \cup \{x_i, y_i\}) = g(B \cup \{y_i\})$ is established analogously.

Henceforth, we will show the following to complete the proof of Lemma 5.1:

$\Rightarrow$ Given a monotone submodular function f achieving the optimum polymatroid bound for I, we can create a monotone submodular function g for I' such that $f(U) = g(U')$.

⇐ Conversely, given a monotone submodular function g achieving the optimum polymatroid bound for I', we can create a monotone submodular function f for I such that $f(U) = g(U')$.

We start with the forward direction. Define $h : 2^{U'} \to 2^U$ as follows: for any $B \subseteq U'$, we have

$$h(B) := \{i \in [n] \mid x_i \in B \text{ or } y_i \in B\}.$$

and set $g(B) := f(h(B))$.

Claim 4. For any $A, B \subseteq U'$, $h(A) \cup h(B) = h(A \cup B)$ and $h(A) \cap h(B) \supseteq h(A \cap B)$.

The first statement follows because if x_i or y_i is in any of A and B, it is also in $A \cup B$. The second statement follows because if $i \in h(A \cap B)$, we have $x_i \in A \cap B$ or $y_i \in A \cap B$ and in both cases, we have $i \in h(A) \cap h(B)$.

At the universe U and U': by definition we have $g(U') = f(h(U')) = f(U)$. Therefore, we only need to show g is monotone and submodular. Showing monotonicity is trivial and is left as an easy exercise. We can show that g is submodular as follows. For any $A, B \subseteq U'$, we have,

$$\begin{aligned} g(A) + g(B) &= f(h(A)) + f(h(B)) \geq f(h(A) \cup h(B)) + f(h(A) \cap h(B)) \\ &\geq f(h(A \cup B)) + f(h(A \cap B)) = g(A \cup B) + g(A \cap B), \end{aligned}$$

where the first inequality follows from f's submodularity and the second from f's monotonicity and Claim 4. Thus we have shown the first direction.

To show the backward direction, define

$$f(A) := g(\{x_i \mid i \in A\})$$

By definition, we have $f(U) = g(\{x_1, x_2, \cdots, x_n\})$. Further, by repeatedly applying Claim 3, we have $g(\{x_1, x_2, \cdots, x_n\}) = g(U')$. Thus we have shown $f(U) = g(U')$. Further, f is essentially identical to g restricted to $\{x_1, x_2, \ldots, x_n\}$. Thus, f inherits g's monotonicity and sumodularity.

This completes the proof of Lemma 5.1. □

A.2 Restriction to degree constraints (X, Y, c) with $|X| \leq 2$ does not help

Proof of Lemma 5.2. The high-level idea is to repeatedly replace two variables with a new variable in a degree constraint. We first discuss how to choose two variables to combine. Assume there is a degree constraint (X, Y, c) where $|X| > 2$. Then we combine an arbitrary pair of elements in X. If $|X| \leq 2$ for all degree constraints, and there is a degree constraint where $|Y| > 3$, we combine arbitrary two variables in $Y \setminus X$. If there is a degree constraint (X, Y, c) where $|X| = 2$, $|Y| = 3$ and $c > 0$, we combine the two variables in X. It is important to note that we make this replacement in only one degree constraint in each iteration.

Assume that we are to combine variables $x, y \in [n]$ into a new variable $z \notin [n]$ (say $z = n + 1$) in a degree constraint $(X, Y, c) \in \text{DC}$. Then, we create (X', Y', c) and add it to DC′ where

$$(X', Y', c) := \begin{cases} (X \setminus \{x, y\} \cup \{z\}, Y \setminus \{x, y\} \cup \{z\}, c) & \text{if } \{x, y\} \subseteq X \subset Y \\ (X, Y \setminus \{x, y\} \cup \{z\}, c) & \text{if } \{x, y\} \subseteq Y \setminus X \end{cases}$$

Further, we add functional dependencies $(\{x, y\}, \{z, x, y\}, 0)$, $(\{z\}, \{z, x\}, 0)$ and $(\{z\}, \{z, y\}, 0)$ to DC′, which we call *consistency constraints*. Intuitively, consistency constraints is to enforce the fact that variable z and the tuple of variables (x, y) are equivalent. The other constraints are called *non-trivial constraints*.

We show that the reduction process terminates by showing that each iteration reduces a potential. Define $m(X, Y, c) := |X| + |Y|$ for a non-trivial constraint (X, Y, c). The potential is defined as the sum of $m(X, Y, c)$ over all non-trivial constraints. Observe that in each iteration, either $m(X, Y, c) > m(X', Y', c)$, or $|X'| = 1$ and $|Y'| = 2$. In the latter case, the resulting non-trivial constraint (X', Y', c)

doesn't change in the subsequent iterations. In the former case the potential decreases. Further, initially the potential is at most $2n|\text{DC}|$ and the number of non-trivial constraints never increases, where DC is the set of degree constraints initially given. Therefore, the reduction terminates in a polynomial number of iterations. It is straightforward to see that we only have degree constraints of the forms that are stated in Lemma 5.2 at the end of the reduction.

Next, we prove that the reduction preserves the polymatroid bound. We consider one iteration where a non-trivial constraint (X, Y, c) is replaced according to the reduction described above. Let opt and opt$'$ be the polymatroid bounds before and after performing the iteration respectively. Let DC and DC$'$ be the sets of the degree constraints before and after the iteration respectively.

We first show opt $\geq$ opt$'$. Let $g : \{z\} \cup [n] \to [0, \infty)$ be a polymatroid function that achieves opt$'$ subject to DC$'$. Define $f : [n] \to [0, \infty)$ such that $f(A) = g(A)$ for all $A \subseteq [n]$. It is immediate that f is a polymatroid because it is a restriction of f onto $[n]$. We show below that it satisfies the degree constraints in DC and that $f([n]) = g([n] \cup \{z\}) = \text{opt}'$. We start with a claim.

Claim 5. For any set Z, $g(Z \cup \{z\}) = g(Z \cup \{z, x, y\}) = g(Z \cup \{x, y\})$.

From the consistency constraint $(\{x, y\}, \{z, x, y\}, 0)$ and the monotonicity of g, we have $0 \leq g(\{z, x, y\}) - g(\{x, y\}) \leq 0$, which means $g(\{x, y, z\}) = g(\{x, y\})$. Similarly, from the consistency constraints $(\{z\}, \{z, x\}, 0)$ and $(\{z\}, \{z, y\}, 0)$ and the monotonicity of g, we have $g(\{z\}) = g(\{z, x\}) = g(\{z, y\})$. Then, from g's submodularity and monotonicity, we have

$$g(\{x, y, z\}) \leq g(\{z, x\}) + g(\{z, y\}) - g(\{z\}) = g(\{z\}) \leq g(\{x, y, z\}),$$

This implies $g(\{z\}) = g(\{x, y, z\})$.

Now, given two sets A, B such that $g(A) = g(A \cup B)$, then for any set Z we have

$$g(Z \cup A \cup B) \leq g(Z \cup A) + g(A \cup B) - g(A \cup (Z \cap B)) \leq g(Z \cup A) + g(A \cup B) - g(A) = g(Z \cup A) \leq g(Z \cup A \cup B).$$

Therefore, $g(Z \cup A) = g(Z \cup A \cup B)$. Applying this fact with $A = \{z\}$ and $B = \{x, y, z\}$, we have $g(Z \cup \{z\}) = g(Z \cup \{x, y, z\})$. Similarly, $g(Z \cup \{x, y\}) = g(Z \cup \{x, y, z\})$. Claim 5 is thus proved.

We now check if f satisfies DC. Because we only replaced $(X, Y, c) \in \text{DC}$, we only need to show that f satisfies it. Note that, if $\{x, y\} \subseteq Z$, then from Claim 5, we have

$$f(Z) = g(Z) = g(Z \cup \{x, y\}) = g(Z \cup \{z\}) = g((Z \setminus \{x, y\}) \cup \{x, y, z\}) = g((Z \setminus \{x, y\}) \cup \{z\})$$

We need to consider two case:

- When $\{x, y\} \subseteq X \subseteq Y$. Then,

  $$f(Y) - f(X) = g(Y \cup \{z\} \setminus \{x, y\}) - g(X \cup \{z\} \setminus \{x, y\}) = g(Y') - g(X') \leq c$$

  The second inequality follows from the fact that g satisfies DC$'$.
- When $\{x, y\} \subseteq Y \setminus X$. In this case, $X' = X$ and $Y' = Y \cup \{z\} \setminus \{x, y\}$; thus

  $$f(Y) - f(X) = g(Y \cup \{z\} \setminus \{x, y\}) - g(X) = g(Y') - g(X') \leq c$$

Finally, $f([n]) = g([n]) = g([n] \cup \{z\}) = \text{opt}'$ due to Claim 5. Since we have shown f is a feasible solution for DC, we have opt $\geq f([n])$. Thus, we have opt $\geq$ opt$'$ as desired.

We now show opt $\leq$ opt$'$. Given f that achieves opt subject to DC, we construct $g : \{z\} \cup [n] \to [0, \infty)$ as follows:

$$g(A) := \begin{cases} f(A) & \text{if } z \notin A \\ f(A \setminus \{z\} \cup \{x, y\}) & \text{otherwise} \end{cases} \tag{18}$$

We first verify that g is monotone. Consider $A \subseteq B \subseteq \{z\} \cup [n]$. If $z \notin A$ and $z \notin B$, or $z \in A$ and $z \in B$, it is easy to see that is the case. So, assume $z \notin A$ but $z \in B$. By definition of g, it suffices

show $f(A) \le f(B \setminus \{z\} \cup \{x, y\})$, which follows from f's monotonicity: Since $z \notin A$ and $A \subseteq B$, we have $A \subseteq B \setminus \{z\} \cup \{x, y\}$.

Secondly we show that g is submodular. So, we want to show that $g(A)+g(B) \ge g(A \cup B)+g(A \cap B)$ for all $A, B \subseteq \{z\} \cup [n]$.

- When $z \notin A$ and $z \notin B$. This case is trivial as g will have the same value as f for all subsets we're considering.
- When $z \in A$ and $z \in B$. We need to check if $f(A \setminus \{z\} \cup \{x, y\}) + f(B \setminus \{z\} \cup \{x, y\}) \ge f(A \cup B \setminus \{z\} \cup \{x, y\}) + f(A \cap B \setminus \{z\} \cup \{x, y\})$, which follows from f's submodularity. More concretely, we set $A' = A \setminus \{z\} \cup \{x, y\}$ and $B' = B \setminus \{z\} \cup \{x, y\}$ and use $f(A') + f(B') \ge f(A' \cup B') + f(A' \cap B')$.
- When $z \in A$ and $z \notin B$ (this is symmetric to $z \notin A$ and $z \in B$). We have

$$\begin{aligned}
g(A) + g(B) &= f(A \setminus \{z\} \cup \{x, y\}) + f(B) \\
(\text{submodularity of } f) &\ge f((A \setminus \{z\} \cup \{x, y\}) \cup B) + f((A \setminus \{z\} \cup \{x, y\}) \cap B) \\
&= f(((A \cup B) \setminus \{z\}) \cup \{x, y\}) + f((A \setminus \{z\} \cup \{x, y\}) \cap B) \\
&= g(A \cup B) + f((A \cup \{x, y\}) \cap B) \\
(\text{monotonicity of } f) &\ge g(A \cup B) + f(A \cap B) \\
&= g(A \cup B) + g(A \cap B)
\end{aligned}$$

Thirdly, we show that g satisfies DC′. Suppose we replaced a non-trivial constraint (X, Y, c) with (X', Y', c). We show $g(Y') - g(X') \le c$ by showing $f(Y) = g(Y')$ and $f(X) = g(X')$. Both cases are symmetric, so we only show $f(X') = g(X)$. If $z \notin X'$, then clearly we have $g(X') = f(X)$ since $X' = X$. If $z \in X'$, then it must be the case that $X' = X \setminus \{x, y\} \cup \{z\}$. By definition of g, we have $g(X') = f(X' \setminus \{z\} \cup \{x, y\}) = f(X)$ since $X' \setminus \{z\} \cup \{x, y\} = X$.

Now we also need to check g satisfies the consistency constraints we created. So we show

- $g(\{z, x, y\}) \le g(\{z\})$. Note $g(\{z, x, y\}) = f(\{x, y\}) = g(\{z\})$ by definition of g. Due to g's monotonicity we have already shown, we have $g(\{z, x\}) \le g(\{z\})$ and $g(\{z, y\}) \le g(\{z\})$.
- $g(\{z, x, y\}) \le g(\{x, y\})$. Both sides are equal to $f(\{x, y\})$ by definition of g.

Finally, we have $g(\{z\} \cup [n]) = f([n])$. Since g is a monotone submodular function satisfying DC, we have $\text{opt}' \ge \text{opt}$ as desired.

This completes the proof of Lemma 5.2. □

B Proof of Theorem 1.4

Recall that *Normal* functions [1, 2] (also called *weighted coverage functions*, or *entropic functions with non-negative mutual information*), are defined as follows. For every $V \subsetneq [n]$, a *step function* $s_V : 2^{[n]} \to \mathbb{R}_+$ is defined by

$$s_V(X) = \begin{cases} 0 & X \subseteq V \\ 1 & \text{otherwise} \end{cases} \tag{19}$$

A function is *normal* if it is a non-negative linear combination of step functions. Let N_n denote the set of normal functions on $[n]$.

To show the hardness result, we consider the dual linear programming formulation, which is exactly $D_{\boldsymbol{\delta}}^{\text{simple}}$. For easy reference we reproduce the LP below.

$$\begin{aligned} \min \quad & \sum_{i\in[k]} c_i \cdot \delta_i \\ \text{s.t.} \quad & \sum_{i\in[k]:\ X_i\subseteq V, Y_i\not\subseteq V} \delta_i \geq 1 \qquad\qquad \forall V \subsetneq [n] \end{aligned}$$

Let $\Delta(\text{DC})$ denote the convex region over δ defined by the constraints in DC. We first show the separation problem is hard.

Lemma B.1. *Given a degree constraint set* DC *and a vector* $\hat{\delta} \in \mathbb{R}_{\geq 0}^{|\text{DC}|}$*, checking if* $\hat{\delta} \notin \Delta(\text{DC})$ *is NP-complete. Further, this remains the case under the extra condition that* $\lambda\hat{\delta} \in \Delta(\text{DC})$ *for some* $\lambda > 1$.

We prove this theorem using a reduction from the Hitting Set problem, which is well-known to be NP-complete. In the Hitting Set problem, the input is a set of n elements $E = \{e_1, \ldots, e_n\}$, a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of m subsets of E, and an integer $k > 0$. The answer is true iff there exists a subset L of k elements such that for every set $S_i \in \mathcal{S}$ is 'hit' by the set L chosen, i.e., $L \cap S_i \neq \emptyset$ for all $i \in [m]$.

Consider an arbitrary instance H to the Hitting Set. To reduce the problem to the membership problem w.r.t. $\Delta(\text{DC})$, we create an instance for computing the normal bound that has $E' = E \cup \{e^*\}$ as variables and the following set DC of degree constraints and $\hat{\delta}$ (here we do not specify the value of c associated with each degree constraint (X, Y) as it can be arbitrary and we're concerned with the hardness of the membership test).

(1) $(\emptyset, \{e_i\})$ for all $e_i \in E$ with $\hat{\delta}_{\emptyset,\{e_i\}} = 1/(k+1)$.
(2) (S_i, E') for all $S_i \in \mathcal{S}$ with $\hat{\delta}_{S_i,E'} = m$.
(3) $(\{e^*\}, E')$ with $\hat{\delta}_{\{e^*\},E'} = m$.

Notably, to keep the notation transparent, we used $\hat{\delta}_{X,Y}$ to denote the value of $\hat{\delta}$ associated with (X, Y). Let DC_1, DC_2, and DC_3 denote the degree constraints defined above in each line respectively, and let $\text{DC} := \text{DC}_1 \cup \text{DC}_2 \cup \text{DC}_3$. To establish the reduction we aim to show the following lemma.

Lemma B.2. *There exists a hitting set of size* k *in the original instance* H *if and only if* $\hat{\delta} \notin \Delta(\text{DC})$.

Proof. Let $L(V) := \sum_{(X,Y)\in\text{DC}: X\subseteq V, V\not\subseteq Y} \hat{\delta}_{Y|X}$. Let $L_{\min} := \min_{V\subset E'} L(V)$ and $V_{\min} := \arg\min_{V\subset E'} L(V)$. To put the lemma in other words, we want to show that H admits a hitting set of size k if and only if $L_{\min} < 1$.

Let $\hat{\delta}(\text{DC}') := \sum_{(X,Y)\in\text{DC}'} \hat{\delta}_{X,Y}$. Note that $L_{\min} \leq L(\emptyset) = \hat{\delta}(\text{DC}_1) = \frac{m}{k+1}$. Therefore, we can have the following conclusions about $W_{\min}$.

- $e^* \notin V_{\min}$ since otherwise $L_{\min} \geq \hat{\delta}(\text{DC}) = m$.
- For all $S_i \in \mathcal{S}$, $S_i \not\subseteq V_{\min}$ since otherwise $L_{\min} \geq \hat{\delta}(\{(S_i, E')\}) = m$.

Thus, we have shown that only the degree constraints in DC_1 can contribute to $L_{\min}$. As a result,

$$L_{\min} = L(V_{\min}) = \hat{\delta}(\{(\emptyset, \{e_i\}) \in \text{DC}_1 : \{e_i\} \not\subseteq V_{\min}\}) = \frac{1}{k+1}|E \setminus V_{\min}|.$$

As observed above, for all $S_i \in \mathcal{S}$, $S_i \not\subseteq V_{\min}$, which means $(E \setminus V_{\min}) \cap S_i \neq \emptyset$. This immediately implies that $E \setminus V_{\min}$ is a hitting set.

To recap, if $\hat{\delta} \notin \Delta(\text{DC})$, we have $\frac{1}{k+1}|E \setminus V_{\min}| < 1$ and therefore the original instance H admits a hitting set $E \setminus V_{\min}$ of size at most k.

Conversely, if the instance H admits a hitting set E' of size k, we can show that $L(E \setminus E') = \hat{\delta}(\{(\emptyset, \{e_i\}) \in \mathrm{DC}_1 \mid e_i \in E'\}) = \frac{k}{k+1} < 1$, which means $\hat{\delta} \notin \Delta(\mathrm{DC})$. This direction is essentially identical and thus is omitted. □

The above lemma shows checking $\hat{\delta} \notin \Delta(\mathrm{DC})$ is NP-hard. Further, a violated constraint can be compactly represented by V; thus the problem is in NP. Finally, if we scale up $\hat{\delta}$ by a factor of $\lambda = k + 1$, we show $\lambda \hat{q} \in \Delta(\mathrm{DC})$. We consider two cases. If $E \not\subseteq V$, we have $L(V) \geq \hat{\delta}(\{i \in [n] : e_i \notin V\}) \geq \frac{1}{k+1}\lambda = 1$. If $E \subseteq V$, it must be the case that $E = V$ since $V \neq E'$ and $E' = E \cup \{e^*\}$. In this case $L(E) = \hat{\delta}(\mathrm{DC}_2) \geq m\lambda \geq 1$. Thus, for all $V \subset E'$, we have $L(V) \geq 1$, meaning $\lambda\hat{\delta} \in \Delta(\mathrm{DC})$. This completes the proof of Theorem 6.1.

Using this theorem, we want to show that we can't solve D' in polynomial time unless P = NP. While there exist relationship among the optimization problem, membership problem and their variants [16], in general hardness of the membership problem doesn't necessarily imply hardness of the optimization problem. However, using the special structure of the convex body in consideration, we can show such an implication in our setting. The following theorem would immediately imply Theorem 1.4.

Lemma B.3. *We cannot solve D' in polynomial time unless P = NP.*

Proof. Consider an instance to the membership problem consisting of DC and $\hat{\delta}$. By Theorem 6.1, we know checking $\hat{\delta} \notin \Delta(\mathrm{DC})$ is NP-complete, even when $\lambda\hat{\delta} \in \Delta(\mathrm{DC})$ for some $\lambda > 1$. For the sake of contradiction, suppose we can solve D'_S in polynomial time for any $w \geq 0$ over the constraints defined by the same $\Delta(\mathrm{DC})$. We will draw a contradiction by showing how to exploit it to check $\hat{\delta} \notin \Delta(\mathrm{DC})$ in polynomial time.

Define $R := \{w \mid w \cdot (\delta - \hat{\delta}) > 0 \ \ \forall \delta \in \Delta(G)\}$. It is straightforward to see that R is convex.

We claim that $\hat{\delta} \notin \Delta(\mathrm{DC})$ iff $R \neq \emptyset$. To show the claim suppose $\hat{\delta} \notin \Delta(\mathrm{DC})$. Recall from Theorem 6.1 that there exists $\lambda > 1$ such that $\lambda\hat{\delta} \in \Delta(\mathrm{DC})$. Let $\lambda' > 0$ be the smallest λ'' such that $\lambda''\hat{\delta} \in \Delta(\mathrm{DC})$. Observe that $\lambda' > 1$ and $\lambda'\hat{\delta}$ lies on a facet of $\Delta(\mathrm{DC})$, which corresponds to a hyperplane $\sum_{i\in[k]:X_i\subseteq V, V\not\subseteq Y_i} \delta_i = 1$ for some $V \subset [n]$. Let w be the orthogonal binary vector of the hyperplane; so we have $w \cdot \lambda'\hat{\delta} = 1$. Then, $w \cdot (\delta - \lambda'\hat{\delta}) \geq 0$ for all $\delta \in \Delta(\mathrm{DC})$. Thus, for any $\delta \in \Delta(\mathrm{DC})$ we have $w \cdot (\delta - \hat{\delta}) \geq (\lambda' - 1)w \cdot \hat{\delta} = \frac{\lambda'-1}{\lambda} w \cdot \lambda\hat{\delta} \geq \frac{\lambda'-1}{\lambda} > 0$. The other direction is trivial to show: If $\hat{\delta} \in \Delta(\mathrm{DC})$, no w satisfies $w \cdot (\delta - \hat{\delta}) > 0$ when $\delta = \hat{\delta}$.

Thanks to the claim, we can draw a contradiction if we can test if $R = \emptyset$ in polynomial time. However, R is defined on an open set which is difficult to handle. Technically, R is defined by infinitely many constraints but it is easy to see that we only need to consider constraints for δ that are vertices of $\Delta(\mathrm{DC})$. Further, $\Delta(\mathrm{DC})$ is defined by a finite number of (more exactly at most 2^n) constraints (one for each W). This implies that the following LP,

$$\begin{aligned} &\max \epsilon \\ w \cdot (\delta - \hat{\delta}) &\geq \epsilon \quad \forall \delta \in \Delta(\mathrm{DC}) \\ w &\geq 0 \end{aligned}$$

has a strictly positive optimum value iff $R \neq \emptyset$. We solve this using the ellipsoid method. Here, the separation oracle is, given $w \geq 0$ and ϵ, to determine if $w \cdot (\delta - \hat{\delta}) \geq \epsilon$ for all $\delta \in \Delta(\mathrm{DC})$; otherwise it should find a $\delta \in \Delta(\mathrm{DC})$ such that $w \cdot (\delta - \hat{\delta}) < \epsilon$. In other words, we want to know $\min_{\delta\in\Delta(\mathrm{DC})} w \cdot (\delta - \hat{\delta})$. If the value is no smaller than ϵ, all constraints are satisfied, otherwise, we can find a violated constraint, which is given by the δ minimizing the value. But, because the oracle assumes $w \cdot \hat{\delta}$ is fixed, so this optimization is essentially the same as solving D', which can be solved

by the hypothetical polynomial time algorithm we assumed to have for the sake of contradiction. Thus, we have shown that we can decide in poly time if R is empty or not. □

C Proof of Theorem 1.5

Proof of Theorem 1.5. For part (a), To show that $\mathsf{DC}[\Gamma_n] \leq \mathsf{flow\text{-}bound}(\mathsf{DC}, \pi)$ it is sufficient to show that a solution to the linear program for $\mathsf{flow\text{-}bound}_\pi(\mathsf{DC})$ can be extended to a feasible flow for linear program D as we did for simple degree constraints in section 3.2. The only difference here is that some flow can be directly pushed to a sink t on nonsimple edges in DC_t. The fact that the flow bound is smaller than the chain bound follows immediately from the fact that the degree constraints used in the chain bound are a subset of the degree constraints used in the flow bound.

Statement (b) follows because for simple instances the linear program for $\mathsf{flow\text{-}bound}_\pi$ and D'_S are identical, and for acyclic instances the polymatroid bound equals the chain bound [25].

Finally, we prove part (c), stating the chain bound can be arbitrarily larger than the flow bound follows from the following instance. Consider an instance consisting of two elements 1 and 2 and let the permutation π follow this order. There is a cardinality constraint $(\emptyset, \{2\}, 1)$ and simple degree constraint $(\{2\}, \{1, 2\}, 1)$. The chain bound is unbounded because it cannot use the simple degree constraint that does not agree with π. Alternatively, the flow bound is bounded by using both degree constraints. □

Received December 2024; revised February 2025; accepted March 2025