Optimizing Entropic and Polymatroid Functions
Subject to Difference Constraints

Sungjin Im\textsuperscript{1}, Benjamin Moseley\textsuperscript{2}, Hung Q. Ngo\textsuperscript{3}, Kirk Pruhs\textsuperscript{4}, and Alireza Samadian\textsuperscript{4}

\textsuperscript{1} University of California, Merced CA, 95343
\textsuperscript{2} Tepper School of Business, Carnegie Mellon University, Pittsburgh PA, 15213
\textsuperscript{3} RelationalAI, Inc., Berkeley, CA, 94704.
\textsuperscript{4} Computer Science Department, University of Pittsburgh, Pittsburgh PA, 15260

Abstract. We consider a class of optimization problems that involve determining the maximum value that a function in a particular class can attain subject to a collection of difference constraints. We show that a particular linear programming technique, based on duality and projections, can be used to rederive some structural results that were previously established using more ad hoc methods. We then show that this technique can be used to obtain a polynomial-time algorithm for a certain type of simple difference constraints. Finally we give lower bound results that show that certain possible extensions of these results are probably not feasible.

Keywords: Submodular optimization · cardinality estimation · entropic implication

1 Introduction

We consider a class of optimization problems that involve determining the maximum value that a function in a particular class $\mathcal{C}$ can attain subject to a collection of difference constraints. So written as a mathematical program, these problems are of the form $\text{DC}[\mathcal{C}]$:

$$\max h([n]) - h(\emptyset)$$
$$\text{s.t. } h(Y_i) - h(X_i) \leq c_i, \quad i \in [k]$$

where each $X_i \subsetneq Y_i \subseteq [n]$, and the “variable” is the function (or vector) $h : 2^n \to \mathbb{R}^+$. 

\* S. Im was supported in part by NSF grants CCF-1617653, CCF-1844939 and CCF-2121745. B. Moseley was supported in part by NSF grants CCF-1824303, CCF-1845146, CCF-2121744 and CMMI-1938909, a Google Research Award, an Infor Research Award, and a Carnegie Bosch Junior Faculty Chair. K. Pruhs was supported in part by NSF grants CCF-1907673, CCF-2036077, CCF-2209654 and an IBM Faculty Award. A. Samadian contributed to this work while he was a PhD student at the University of Pittsburgh and is now affiliated with Google Pittsburgh.
We are primarily interested in two classes $\mathcal{C}$ of functions. The first is the class $\Gamma^*_{n}$ of entropic functions. If $h$ is an entropic function then the difference $h(Y_i) - h(X_i)$ is equal to $h(Y_i | X_i)$, the conditional entropy of $Y_i$ conditioned on the knowledge of $X_i$. Thus $DC[\Gamma^*]$ is the problem of determining the maximum possible entropy subject to conditional entropy constraints. The second is the class $\Gamma_n$ of polymatroid functions. Other classes of functions that will play a role in our story are the class $N_n$ of weighted coverage functions (normal functions in the database literature), and the class $M_n$ of modular functions.

Observe that for all collections $DC$ of difference constraints,

$$DC[M_n] \leq DC[N_n] \leq DC[\Gamma^*_n] \leq DC[\Gamma_n]$$

where here $DC[C]$ should be interpreted to mean the optimal objective value. The first inequality follows because modular functions are weighted coverage functions, the second inequality follows because weighted coverage functions are entropic functions, and last inequality follows because entropic functions are polymatroid functions. It will be convenient to refer to these quantities as the modular bound, the coverage bound, the entropic bound and the polymatroid bound, respectively. It is known that for arbitrary difference constraints that the gaps between these bounds can arbitrarily large [ANS17].

1.1 Database Applications and Background

Our main motivation for considering these types of optimization problems arises from applications in databases, for example in bounding the cardinality of a table that is formed by joining smaller tables [Ngo22]. The difference constraints express knowledge about the data that might come from some external understanding of the relation between various attributes, or information that is easily obtained from the data. So for example, consider a simple database for US postal service that contains the following three attributes (among others), attribute 1 is zip code, attribute 2 is city and attribute 3 is state. Assume that it is known that there are at most 50 states, that no state contains more than 2,598 zip codes, that no city is in more than one state, and no state contains more than 9,197 cities. Then suppose one joins together a collection of tables to get a table $T$ consisting of (zip code, city, state) tuples. Then the number of tuples in $T$ can be upper bounded using the following instance of $DC[\Gamma^*_n]$ problem:

$$\max h(\{1, 2, 3\}) - h(\emptyset)$$
$$\text{s.t. } h(\{3\} | \emptyset) \leq \lg 50$$
$$h(\{1, 3\} | \{3\}) \leq \lg 2598$$
$$h(\{2, 3\} | \{2\}) \leq \lg 1$$
$$h(\{2, 3\} | \{3\}) \leq \lg 9197$$
$$h \in \Gamma^*_n$$

In particular, if this optimal objective is $E$ then $2^E$ is an upper bound to the cardinality of the number of tuples in $T$ (essentially because uniform distributions have maximum entropy) [ANS17]. For certain common types of queries,
e.g. disjunctive datalog queries, and joins with functional dependencies and/or cardinality constraints, it is known that this entropic bound is asymptotically tight \([\text{ANS17,ANS16}]\). Unfortunately the space of entropic functions is complicated. For example, if arbitrary linear constraints are allowed (instead of just difference constraints) the problem is not even computable \([\text{Li21,KY22}]\). Thus the interest in the polymatroid bound in the database community derives from the fact that it represents a clearly computable, and potentially even efficiently computable, upper bound.

To date, database researchers have largely considered certain classes of difference constraints that commonly/naturally arise in database applications \([\text{Ngo22}]\). Of particular interest to us here are:

**Acyclic Difference Instances:** An acyclic instance is one where the dependency digraph of the difference constraints is acyclic. The vertices of the dependency digraph is the universe \([n]\) and \((u,v)\) is a directed edge if and only if there exists an \(i \in [k]\) such that \(u \in X_i\) and \(v \in Y_i - X_i\).

**Simple Difference Constraint:** A simple difference constraint is one where \(|X_i| \leq 1\). A simple instance is one where all difference constraints are simple.

**Cardinality Constraint:** A cardinality constraint is one where \(|X_i| = 0\).

**Functional Dependency** A functional dependency is one where \(c_i = 0\).

In \([\text{Ngo18}]\) it was shown that, for acyclic instances the modular bound is equal to the polymatroid bound. The proof technique was to show that every polymatroid function \(h\) that satisfies the difference constraints can be converted in a modular function \(h'\) that also satisfies the difference constraints and that has the same objective value as \(h\). As the modular bound can be computed by a polynomially sized linear program, this observation yields a polynomial time algorithm to compute the entropic/polymatroid bound for acyclic instances. In \([\text{AKNS20}]\) it was shown that, for simple instances the coverage bound is equal to the polymatroid bound. The proof technique was to show that every polymatroid function \(h\) satisfying the difference constraints could be converted in a weighted coverage function \(h'\) satisfying the difference constraints and having the same objective value as \(h\). This observation doesn’t immediately yield a polynomial time algorithm as the natural linear programming formulation of the coverage bound has exponentially many variables and exponentially many constraints.

There is a rich history of research in this area in the database theory community, that we can’t hope to do justice to here, but a good starting point for reader are the surveys \([\text{Ngo22,Ngo18}]\). Also entropy implication problems are central to the area of information theory, see for example the textbooks \([\text{Yeu02,Stu05}]\).

1.2 Our Contributions

**The Dual-Project-Dual Technique** We apply a linear programming based technique, which we will call the dual-project-dual technique, that allows us to rederive the results that \(\text{DC}[M_n] = \text{DC}[\Gamma_n]\) for acyclic instances and \(\text{DC}[N_n] = \text{DC}[\Gamma_n]\) for simple instances. We then extend these results in various ways.
Our starting point is a natural linear programming formulation $P$ for the problem of computing $DC[Γ_n]$:

$$
P : \text{max } h([n]) - h(\emptyset)$$

s.t. $h(Y \cup X) - h(X) - h(Y) + h(Y \cap X) \leq 0 \quad \forall X \forall Y X \perp Y$

$h(Y) - h(X) \geq 0 \quad \forall X \forall Y X \subseteq Y$

$h(Y_i) - h(X_i) \leq c_i \quad \forall i \in [k]$

where there is a variable $h(X)$ for each subset $X$ of the universe $[n]$, and $X \perp Y$ means $X \not\subseteq Y$ and $Y \not\subseteq X$. We will adopt the convention that all variables in our mathematical programs are constrained to be nonnegative unless explicitly mentioned otherwise. An optimal polymatroid function $h'$ would then be $h'(X) = h(X) - h(\emptyset)$, where the values on the right hand side come from the linear program $P$. Note that the linear program $P$ has both exponentially many variables and exponentially many constraints. But critically this linear program only has linearly many constraints where the constant on the right-hand-size of the constraint is nonzero (one for each difference constraint).

The first step of our dual-project-dual technique is to take the dual of $P$ to obtain a linear program $D$. If we associate a dual variables $σ_{X,Y}$, dual variables $µ_{X,Y}$ and dual variables $δ_i$ with the three types of constraints in $P$ (in that order), then the dual linear program $D$ is:

$$
D : \text{min } \sum_{i \in [k]} c_i \cdot δ_i
$$

s.t. $\text{excess}([n]) \geq 1$

$\text{excess}(\emptyset) \geq -1$

$\text{excess}(Z) \geq 0, \quad \forall Z \neq \emptyset, [n]$

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

$$
\text{excess}(Z) := \sum_{i,Z=Y_i} δ_i - \sum_{i,Z=X_i} δ_i + \sum_{I \cup J = Z} σ_{I,J} + \sum_{I \cup J' = Z} σ_{I',J'} - \sum_{J \cup L = Z} σ_{Z,J} - \sum_{X \cup X' \subseteq Z} µ_{X,Z} + \sum_{Y \cup Z \subseteq Y} µ_{Z,Y}
$$

(1)

See Figure 1 for an illustration of $\text{excess}(Z)$.

So the problem modeled by linear program $D$ can be interpreted as a min-cost $s$-$t$ flow problem on a hypergraph $L$. The vertices of $L$ are the subsets of $[n]$. Each variable $µ_{X,Y}$ represents a directed edge with no cost and infinite capacity from $Y$ to $X$, where $X \subseteq Y$. The variable $δ_i$ represents the capacity on the directed edge from $X_i$ to $Y_i$. The cost to buy this capacity is $δ_i \cdot c_i$. The variable $σ_{X,Y}$ represents a hyperedge, and $σ_{X,Y} = f$ means $f$ units of flow leave each of $X$ and $Y$, and $f$ units of flow enters each of $X \cap Y$ and $X \cup Y$. Flow can not be created at any vertex other than the empty set. 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 $t = [n]$. 
For the project step of our dual-project-dual technique, we now consider the region $\Delta$ formed by projecting the feasible region for the linear program $D$ down onto the space corresponding to the $\delta_i$ variables (and perhaps some other variables in $D$). Then one can form a (potentially smaller) linear program $D'$ equivalent to $D$ by replacing the constraints in $D$ by constraints that define $\Delta$.

The final step of the dual-project-dual technique is to take the dual of $D'$ to obtain a linear program $P'$ that is equivalent to $P$, but that is potentially smaller and/or simpler than $P$.

The Dual-Project-Dual Technique for Simple Instances In Section 3 we consider the application of the dual-project-dual technique to simple instances. For simple instances we show that this dual-project process results in the following linear program $D_S$:

$$D_S : \min \sum_{i \in [k]} c_i \cdot \delta_i$$

s.t. $$\sum_{i \in [k]} \delta_i \geq 1 \quad \forall V \neq \emptyset, V \subseteq [n]$$

$$X_i \cap V = \emptyset \quad V \cap Y_i \neq \emptyset$$

We then observe that the dual $P_S$ of the linear program $D_S$ is the natural linear program for optimizing over weighted coverage functions:

$$P_S : \max \sum_{\emptyset \neq V \subseteq [n]} \lambda_V$$

s.t. $$\sum_{V \neq \emptyset} \lambda_V \leq c_i \quad \forall i \in [k]$$

$$X_i \cap V = \emptyset \quad V \cap Y_i \neq \emptyset$$
Here $\lambda_V$ is the weight of the vertex connected to vertices in $V$ in the standard bipartite representation of a weighted coverage function (without loss of generality one can assume that there is only one such vertex). Thus we rederive the fact that $DC[N_n] = DC[\Gamma_n^*] = DC[\Gamma_n]$ for simple instances.

We then consider the separation problem for $D_S$, where the values of $\delta_i$’s are given, and the goal is to determine whether this is $D_S$-feasible. We show that this separation problem is equivalent to $n$ related $s$-$t$ flow problems on a particular subgraph $G$ of $L$. The vertices in $G$ are the empty set, the singleton sets, and the sets $Y_i$, $i \in [k]$. The edges in $G$ are all the $\delta_i$ edges from $L$, and all the $\mu_{X,Y}$ edges from $L$ where $X$ and $Y$ are vertices in $G$ and $|X| \leq 1$. We then show that a setting of the $\delta_i$ variables is feasible in $D_S$ if and only if for all $t \in [n]$ it is possible to route a unit of flow in $G$ from a source $s = \emptyset$ to the sink $\{t\}$. Note that each of these flow problems is independent, so while each flow has to respect the edge capacities, the aggregate flows over all sinks can exceed the edge capacities. As an immediate consequence, we can conclude that there is a polynomial-time combinatorial algorithm for the separation problem for $D_S$, and the following linear program $D'_S$ is equivalent to $D_S$:

$$D'_S : \quad \min \sum_{i \in [k]} c_i \cdot \delta_i \quad \text{(4)}$$

$$\text{s.t.} \quad f_{i,t} \leq \delta_i \quad \forall i \in [k] \quad \forall t \in [n]$$

$$\text{excess}_s(t) \geq 1 \quad \forall t \in [n]$$

$$\text{excess}_s(\emptyset) \geq -1 \quad \forall t \in [n]$$

$$\text{excess}_s(Z) \geq 0 \quad \forall Z \in G \setminus \{\emptyset\} \setminus \{t\} \quad \forall t \in [n]$$

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

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

Here the interpretation of $f_{i,t}$ is the flow from $X_i$ to $Y_i$ in $G$ for the flow problem where the sink is $\{t\}$. As $D'_S$ is of polynomial size, this yields a polynomial-time algorithm to compute the coverage/entropic/submodular bound for simple instances.

The Dual-Project-Dual Technique for Strongly Connected Components In section 4 we consider applying the dual-project-dual technique for computing the polymatroid bound for general difference constraints. Further we consider the effect of the strongly connected components $V_1, \ldots, V_h$ of the dependency graph for the difference constraints. In this case we consider the projection from the feasible region for $D$ onto the space spanned by the $\delta_i$ variables, the $\mu_{X,Y}$ variables where $Y$ is a subset of some connected component of the dependency graph, and the $\sigma_{X,Y}$ variables where $X \cup Y$ is a subset of some connected component of the dependency graph. Without loss of generality, assume $V_1, \ldots, V_h$ is a topological sort of the strongly connected components of the
dependency graph. We show that the result of such a projection is the following linear program $D_{SCC}$:

$$D_{SCC} : \quad \min \sum_{i \in [k]} c_i \cdot \delta_i$$

s.t. \quad \text{excess}(Z, V_j) \geq 0 \quad \forall j \in [h] \quad \forall Z \not\subsetneq Z \subsetneq V_j$$

$$\text{excess}(\emptyset, V_j) \geq -1 \quad \forall j \in [h]$$

$$\text{excess}(V_j, V_j) \geq 1 \quad \forall j \in [h]$$

where we define $\text{excess}$ as follows:

$$\text{excess}(Z, U) := \sum_{I \cup J \subseteq U \atop I \perp J} \sigma_{I, J} + \sum_{I \cup J = Z} \sigma_{I, J} - \sum_{J \in U \atop J \perp Z} \sigma_{Z, J} - \sum_{X \subsetneq Z \atop X \in U} \mu_{X, Z} + \sum_{Z \subseteq Y \subseteq U \atop Z = Y \cap U} \mu_{Z, Y} + \sum_{i \in [k] \atop Z = Y_i \cap U} \delta_i - \sum_{i \in [k] \atop Z = X_i \cap U} \delta_i$$

The way to think about the problem modeled by the linear program $D_{SCC}$ is that consists of essentially one hypergraph flow problem, as is modeled by the linear program $D$, for each connected component. Moreover, the flow problems for the connected components are independent with the exception of sharing the capacities of the projection of the difference constraints into the connected components.

The dual of the linear program $D_{SCC}$ is the following linear program $P_{SCC}$:

$$P_{SCC} : \quad \max \sum_{j=1}^h (h_j(V_j) - h_j(\emptyset))$$

s.t. \quad h_j(Y \cup X) - h_j(X) - h_j(Y) + h_j(Y \cap X) \leq 0 \quad \forall j \in [h] \quad \forall X \cup Y \in V_j$$

$$h_j(Y) - h_j(X) \geq 0 \quad \forall j \in [h] \quad \forall X \cup Y \subseteq V_j$$

$$\sum_{j=1}^h (h_j(V_j \cap Y_i) - h_j(V_j \cap X_i)) \leq c_i \quad \forall i \in [k]$$

The linear program $P_{SCC}$ models computing the optimal objective over what we call semimodular functions. A function is semimodular with respect to a partition $V_1, \ldots, V_h$ of the universe if for each $j \in [h]$ there exists a polymatroid function function $h_j$ on $V_j$ such that for all $X \subseteq [n]$, it is the case that $h(X) = \sum_{j \in [h]} h_j(X \cap V_j)$. So if the strongly connected components are singletons, then a semimodular function is modular. Thus we can conclude that there is always an optimal solution to $DC[\Gamma_n]$ that is semimodular with respect to the connected components, and we recover the result that the modular bound, the entropic bound and the polymatroid bound are equal for acyclic instances.

As the size of $P_{SCC}$ is bounded by a polynomial function in $n$ times an exponential function of the maximum number of vertices in any strongly connected component, this yields a fixed parameter tractable algorithm for computing the
polymatroid bound when the parameter is the maximum number of vertices in any strongly connected component.

Note that given these results one can view the conversion of an optimal polymatroid function to an optimal modular function for acyclic instances in \cite{Ngo18}, and the conversion of an optimal polymatroid function into an optimal weighted coverage function for simple instances in \cite{AKNS20}, as being equivalent to the dual process of projecting down to the variables in the objective in the dual space.

**Lower Bound Reductions** Whether the polymatroid bound for arbitrary difference constraints can be computed in polynomial-time is a fascinating, and seemingly challenging, open question. It is also natural to ask whether we can apply our techniques to other natural classes of difference constraints, but even this is challenging. We have a collection of results that illustrate some of the obstacles to extending our results. We show that computing the coverage bound for general difference constraints is NP-hard. We show how to efficiently reduce the problem of computing the polymatroid bound on general difference constraints to computing the polymatroid bound on difference constraints that are a union of an acyclic instance and a simple instance. And we show how to efficiently reduce the problem of computing the polymatroid bound on general difference constraints to computing the polymatroid bound on difference constraints where for all difference constraints \( i \in [k] \) it is the case that \( |X_i| \leq 2 \) and \( |Y_i| \leq 3 \). This shows that computing the polymatroid bound for such instances is as hard as computing the polymatroid bound in general.

Whether the polymatroid bound for arbitrary difference constraints can be computed in polynomial-time is a fascinating, and seemingly challenging, open question. It is also natural to ask whether we can apply our techniques to other natural classes of difference constraints, but even this is challenging. We have a collection of results that illustrate some of the obstacles to extending our results. In section \( 5 \) we show that computing the coverage bound for general difference constraints is NP-hard. In section \( 6 \) we show how to efficiently reduce the problem of computing the polymatroid bound on general difference constraints to computing the polymatroid bound on difference constraints that are a union of an acyclic instance and a simple instance. And we show how to efficiently reduce the problem of computing the polymatroid bound on general difference constraints to computing the polymatroid bound on difference constraints where for all difference constraints \( i \in [k] \) it is the case that \( |X_i| \leq 2 \) and \( |Y_i| \leq 3 \). This shows that computing the polymatroid bound for such instances is as hard as computing the polymatroid bound in general.
2 Formal Definitions

A function $h : 2^n \rightarrow \mathbb{R}^+$ is entropic if there exist discrete random variables $z_1, \ldots, z_n$ such that for all $X \subseteq [n]$ it is the case that $h(X)$ is the entropy of the marginal distribution on the variables $z_j$ where $j \in X$. A function $h : 2^n \rightarrow \mathbb{R}^+$ is a weighted coverage function if there exists a positive integer $m$, a collection of subsets $T_1, \ldots, T_n$ of $[m]$, and nonnegative weights $w_1, \ldots, w_m$ such that $h(X) = \sum_{j \in [m] : \exists i \in X, j \in T_i} w_j$. A function $h : 2^n \rightarrow \mathbb{R}^+$ is a modular function if there exists nonnegative numbers $z_1, \ldots, z_n$ such that $h(X) = \sum_{i \in X} z_i$. A function $h : 2^n \rightarrow \mathbb{R}^+$ is a polymatroid function if it is nonnegative, normalized ($h(\emptyset) = 0$), monotonically nondecreasing ($h(X) \leq h(Y)$ if $X \subseteq Y$) and submodular ($h(Y \cup X) + h(Y \cap X) \leq h(X) + h(Y)$). An algorithm $A$ is a fixed parameter tractable algorithm in the parameter $k$ if the running time of $A$ can be bounded by a polynomial in the input size times some function of $k$.

3 The Dual Project Dual Approach for Simple Instances

We show in Lemma 1 that the feasible region of the linear program $D_S$ is identical to the feasible region of $D'_S$. We then show in Lemma 2 and Lemma 3 that for simple instances the linear program $D$ is equivalent to the linear program $D'_S$.

Lemma 1. The feasible region of the linear program $D_S$ is identical to the feasible region of $D'_S$.

Proof. Assume that for some setting of the $\delta_i$ variables, that $D'_S$ is infeasible. Then there exists a $t \in [n]$ such that the max flow between $s = \emptyset$ 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 $C$ of vertices in $G$ such that $s \notin C$ and $t \in C$, where the aggregate capacities entering $C$ is less than one. Thus by taking $V := \{i \in [n] \mid \{i\} \in C\}$ we obtain a violated constraint for $D_S$.

Conversely, assume that for some setting of the $\delta_i$ variables, that $D_S$ is infeasible. Then there is a nonempty $V$ such that $\sum_{i \in [k]} \delta_i < 1$. Consider the cut $(V(G) \setminus W, W)$, where $W := \{i \mid i \in V\}$. This cut has value less than one.

Thus again by appealing to the fact that the value of the minimum $s$-$t$ cut is equal to the maximum $s$-$t$ flow, we can conclude that this setting of the $\delta_i$ variables is not feasible for $D'_S$. \qed

Lemma 2. For simple instances, if a setting of the $\delta_i$ variables can be extended to a feasible solution for the linear program $D$ then this same setting of the $\delta$ variables is feasible for the linear program $D'_S$.

\[5\] It is beyond the scope of this discussion to deal with subtleties arising from differential entropies in the continuous case.
Proof. We prove the contrapositive. Consider a setting of the $\delta_i$ variables that is not feasible for $D'_S$. Then we know that there exists a $t \in [n]$ such that there is a cut of value less than one that that separates $s = \emptyset$ and $\{t\}$ in $G$.

Let $V$ be the union of all singleton sets that are on the same side of this cut as $\{t\}$. We know that, $\sum_{i \in [k], X_i \cap V = \emptyset, Y_i \cap V \neq \emptyset} \delta_i < 1$.

Now consider the hypergraph $L$ that is the lattice as defined in Section 1.2. Let $V'$ be all nodes in $L$ which contain at least one vertex in $V$ and $C$ be the remaining vertices. Then in the hypergraph $L$ the aggregate flow into vertices in $V'$ (i.e. out of $C$) can be at most $\sum_{i \in [k], X_i \cap V = \emptyset, Y_i \cap V \neq \emptyset} \delta_i < 1$. This is because no $\mu_{X,Y}$ edge can cause flow to enter $V'$ from $C$; and no $\sigma_{X,Y}$ hyperedge can cause flow to leave $C$ as if $X \in C$ and $Y \in C$ then $X \cup Y \in C$, and if either $X \not\in C$ or $Y \not\in C$ then $\sigma_{X,Y}$ does not route any net flow out of $C$. \hfill $\Box$

Lemma 3. For simple instances, if a setting of the $\delta_i$ variables is feasible for the linear program $D'_S$ then this same setting of the $\delta_i$ variables can be extended to a feasible solution for the linear program $D$.

The rest of the section is devoted to proving Lemma 3. We constructively show how to extend a feasible solution for $D'_S$ to a feasible solution for $D$ by setting $\mu_{X,Y}$ and $\sigma_{X,Y}$ variables. Let $q$ be an integer such that the setting of every $\delta_i$, $i \in [k]$, variable is an integer multiple of $1/q$, and let $\epsilon = 1/q$. Initially $\mu_{X,Y} = \delta_i$ for each difference constraint $j \in [k]$, and all other $\mu$ and $\sigma$ variables are zero. We now give an iterative process to modify these variable settings. The outer loop of the constructive algorithm to iterates over $i \in [n]$. This loop will maintain the following outer loop invariant on the setting of the variables in $D$:

1. The excess at the vertex $[i]$ in $D$ is $1$.
2. The excess at every vertex in $D$, besides $\emptyset$ and $[i]$ is zero.
3. For every $j \in [k]$, if $Y_j \cup [i] \neq X_j \cup [i]$ then $\sum_{0 \leq t \leq 1} \mu_{\{t\} \cup X_j \cup [i] \cup Y_j} = \delta_j$.
4. Each variable is an integer multiple of $\epsilon$.

Note this inductive invariant is initially satisfied, if one interprets $[0]$ to be the empty set, and will represent a feasible solution for $D$ when $i = n$. To extend the inductive hypothesis from $i$ to $i + 1$, let $P^{i+1}$ be the collection of simple flow paths in the graph $G$ that each route an $\epsilon$ unit of flow from $s = \emptyset$ to $t = \{i + 1\}$ in $D'_S$. Our construction then iterates through the paths in $P^{i+1}$, which we call the forward path process, and then iterates through these paths again in what we call the restorative process.

The forward path process processes a path $P$ in $P^{i+1}$ with edges $(A_1, B_1), \ldots, (A_u, B_u)$, where $A_1 = \emptyset$ and $B_u = \{i + 1\}$ as follows. Let $P_{\{t\}}$ be the path in $L$ that is formed from $P$ by deleting edges $(A_h, B_h)$ where $A_h \cup \{t\} = B_h \cup \{t\}$, and replacing edges $(A_h, B_h)$ where $A_h \cup \{t\} \neq B_h \cup \{t\}$ by the edge $(A_h \cup \{t\}, B_h \cup \{t\})$. Let $P^{i+1}_{\{t\}}$ be the collection of all such $P_{\{t\}}$ over all $P \in P^{i+1}$. Our construction then iterates through the edges $(A \cup [i], B \cup [i])$ in $P[i]$ from $[i]$ to $[i + 1]$ (where $(A, B)$ is the corresponding edge in $P$), processing each edge as follows:

1. If $B \subset A$ then increase $\mu_{B \cup [i], A \cup [i]}$ by $\epsilon$. 


2. Else: Let \( t \leq i \) be such that \( \mu_{A \cup [i], B \cup [t]} \geq \epsilon. \)
   (a) If \( A \cup [i] \subset B \cup [t] \) then decrease \( \mu_{A \cup [i], B \cup [t]} \) by \( \epsilon. \)
   (b) Else:
      i. Decrease \( \mu_{A \cup [i], B \cup [t]} \) by \( \epsilon. \)
      ii. Increase \( \sigma_{B \cup [t], A \cup [i]} \) by \( \epsilon. \)
      iii. If \( (A \cup [i]) \cap (B \cup [t]) \neq A \cup [t] \) then increase \( \mu_{A \cup [i], (A \cup [i]) \cap (B \cup [t])} \) by \( \epsilon. \)

The restorative process iterates over the paths in \( P^{i+1}[i+1] \), and then iterates over the edges of each \( P'[i+1] \) in \( P^{i+1}[i+1] \). An edge \( (A \cup [i+1], B \cup [i+1]) \) in \( P[i+1] \) (where \( (A, B) \) is the corresponding edge in \( P \)) is processed as follows:

1. If \( B \subset A \) then
   (a) Decrease \( \mu_{B \cup [i], A \cup [i]} \) by \( \epsilon. \)
   (b) Increase \( \sigma_{A \cup [i], B \cup [i+1]} \) by \( \epsilon. \)
2. Else increase \( \mu_{A \cup [i+1], B \cup [i+1]} \) by \( \epsilon. \)

We will show in Lemma 4 that the outer loop of the forward path process maintains the following forward path loop invariant:

1. The excess at the vertex \( [i] \) in \( D \) is reduced by \( \epsilon \) for each path processed.
2. The excess at the vertex \( [i+1] \) in \( D \) is increased by \( \epsilon \) for each path processed.
3. The excess at every vertex in \( D \), besides \( \emptyset \) and \( [i] \) and \( [i+1] \) is zero.
4. For every \( j \in [k] \), if \( Y_j \cup [i] \neq X_j \cup [i] \) then \( \sum_{0 \leq i \leq 1} \mu_{[i], X_j \cup [i] \cup Y_j} \geq \delta_i - \epsilon \sum_{h=1}^j \tau_{j+1}^i(h) \), where \( \tau_{j+1}^i(h) \) is an indicator function that is 1 if path \( P \) contains the edge \((X_h, Y_h)\) in \( G \). Let us call the right-hand side of this constraint the remaining capacity for difference constraint \( j. \)
5. Each variable is an integer multiple of \( \epsilon \).

**Lemma 4.** The forward path process maintains the forward path loop invariant

**Proof.** First, note that at least one edge has to be processed on every path \( P[i] \). The first edge processed will reduce the excess at \( [i] \). The last edge processed will increase the excess at \( [i+1] \).

It is then sufficient to show that when an edge \( (A \cup [i], B \cup [i]) \) in a path \( Q_j^{i+1}[i] \) is processed the excess of \( A \cup [i] \) decreases by \( \epsilon \), the excess of \( B \cup [i] \) increases by \( \epsilon \), the excess of all other vertices remain unchanged, and the remaining capacity of a difference constraint \( h \) decreases by \( \epsilon \) if and only if \( A = X_h \) and \( B = Y_h \). If \( B \subset A \) then \( (A, B) \) is a \( \mu \) edge then what we want to prove is obvious. So consider the case that \( A \subset B \). In this case we know there is a difference constraint \( h \) where \( A = X_h \) and \( B = Y_h \), and thus \( \tau_{j+1}^i(h) = 1 \). The existence of such a \( t \) follows from the loop invariant and the fact the flow to \( [i+1] \) in \( D'_S \) uses difference constraint \( h \) to an extent at most \( \delta_h \). If \( A \cup [i] \subset B \cup [t] \) then this implies \( i = t \), thus the edge invariant holds and the remaining capacity for difference constraint \( h \) decreases by a most \( \epsilon \).

Otherwise, note that it must be the case that \( i > t \). Further note that \( A \cup [t] \subset (A \cup [i]) \cap (B \cup [t]) \). So consider how the various nodes who are affected.
where \( A \cup [i] \): Increasing \( \sigma_{B \cup [i], A \cup [i]} \) decreases the excess from \( \epsilon \) to 0.

- \( A \cup [t] \): The excess decreases by \( \epsilon \) due to the decrease of \( \mu_{A \cup [t], B \cup [t]} \) is decreased. If \( (A \cup [i]) \cap (B \cup [t]) = A \cup [t] \) this decrease is canceled by the increase of \( \sigma_{B \cup [i], A \cup [i]} \), and otherwise it is canceled by the increase of \( \mu_{A \cup [t], (A \cup [i]) \cap (B \cup [t])} \).

- \((A \cup [i]) \cap (B \cup [t])\): The excess increases from the increase of \( \sigma_{B \cup [i], A \cup [i]} \). If \( (A \cup [i]) \cap (B \cup [t]) = A \cup [t] \) this increase is canceled by the decrease of \( \mu_{A \cup [t], B \cup [t]} \), and otherwise it is canceled by the increase of \( \mu_{A \cup [t], (A \cup [i]) \cap (B \cup [t])} \).

- \( B \cup [t] \): The excess decreases by \( \epsilon \) from the increase of \( \sigma_{B \cup [t], A \cup [i]} \) and increases by \( \epsilon \) from the decrease of \( \mu_{A \cup [t], B \cup [t]} \), resulting in the excess staying at zero.

- \( B \cup [i] \): Increasing \( \sigma_{B \cup [t], A \cup [i]} \) increases the excess from zero to \( \epsilon \).

Notice that upon termination of the forward path process, the forward path loop invariant implies that outer loop invariant is satisfied for \( i + 1 \), with the exception of the third invariant. Lemma 5 shows the the restorative process makes this third invariant true, without affecting the other invariants.

**Lemma 5.** After the restorative process the outer loop invariant holds for \( i + 1 \).

**Proof.** First note that if an \( P_{j+1}^{i+1}[i+1] \) contains no edges, then for each difference constraint \( h \) where \( (X_h, Y_h) \) is an edge in \( P_{j+1}^{i+1} \) it is the case that \( X_h \cup [i + 1] = Y_h \cup [i + 1] \), and thus the remaining capacity for this difference does not need to be restored. So consider a path \( P_{j+1}^{i+1}[i+1] \) that contains a positive number of edges. Note that \( P_{j+1}^{i+1}[i+1] \) is a closed loop as both the first vertex (namely \( \emptyset \)) and last vertex (namely \( \emptyset \)) in \( P_{j+1}^{i+1} \) are subsets of \( [i + 1] \). Thus it will be sufficient to argue that for each edge \((A \cup [i+1], B \cup [i+1]) \) in \( P_{j+1}^{i+1}[i+1] \) it is the case that when this edge is processed the excess of \( A \cup [i+1] \) increases by \( \epsilon \), the excess of \( B \cup [i+1] \) decreases by \( \epsilon \), the excess of all other nodes does not change, and if there is a difference constraint \( h \) where \( A = X_h, B = Y_h, \) and \( X_h \cup [i + 1] \neq Y_h \cup [i + 1] \) then the remaining capacity for this difference constraint will increase by \( \epsilon \). If \( A \subset B \) then there is a difference constraint \( h \) where \( A = X_h, B = Y_h \). Further, if \( X_h \cup [i+1] \neq Y_h \cup [i+1] \) then the remaining capacity for this difference constraint will increase by \( \epsilon \). The invariants about the excesses obvious hold in this case.

So now let us consider the effects when \( B \subset A \). Note that in this case \((B \cup [i+1]) \setminus (A \cup [i]) = \{i + 1\}\). The effects on the excesses of various nodes is:

- \( B \cup [i]: \) Its excess is decreased by \( \epsilon \) due to the decrease of \( \mu_{B \cup [i], A \cup [i]} \), and its excess is increased by \( \epsilon \) due to the increase of \( \sigma_{A \cup [i], B \cup [i+1]} \). Note that the decrease of \( \mu_{B \cup [i], A \cup [i]} \) here negates the increase of \( \mu_{A \cup [i], B \cup [i]} \) when processing edge \((A \cup [i], B \cup [i]) \) in \( P_{j+1}^{i}[i] \) in the forward path process.

- \( B \cup [i + 1]: \) Its excess decreases by \( \epsilon \) due to the increase of \( \sigma_{A \cup [i], B \cup [i+1]} \).

- \( A \cup [i]: \) Its excess is decreased by \( \epsilon \) due to the increase of \( \sigma_{A \cup [i], B \cup [i+1]} \), and increases by \( \epsilon \) due to the decrease of \( \mu_{B \cup [i], A \cup [i]} \).
\( A \cup \{i + 1\} \): Its excess increases by \( \epsilon \) due to the increase of \( \sigma_{A \cup \{i\}, B \cup \{i+1\}} \).

\[ \]  

4 The Dual Project Dual Approach for Strongly Connected Components

For convenience we rewrite \( D_{SCC} \) as:

\[
\min \sum_{i \in [k]} c_i \cdot \delta_i \\
\text{s.t. } D[V_j] \quad \forall j \in [h]
\]

where \( D[U] \) are the constraints

\[
\begin{align*}
\text{excess}^{\alpha, \mu}(Z, U) &\geq -\text{excess}^\delta(Z, U) & \forall Z \subseteq U \\
\text{excess}^{\alpha, \mu}(U, U) &\geq 1 - \text{excess}^\delta(U, U)
\end{align*}
\]

where we define \( \text{excess}^{\alpha, \mu} \) as follows:

\[
\text{excess}^{\alpha, \mu}(Z, U) := \sum_{i \in [k]} \sigma_{I, J} + \sum_{i \in [k]} \sigma_{I, J} - \sum_{Z \subseteq U} \mu_{Z, Y} - \sum_{X \subseteq Z \subseteq Y \subseteq U} \mu_{X, Z}
\]

and \( \text{excess}^\delta \) as follows:

\[
\begin{align*}
\text{excess}^\delta(Z, U) &:= \sum_{i \in [k]} \delta_i - \sum_{i \in [k]} \delta_i \\
&\quad \text{where } Z = Y \cap U \\
&\quad \text{and } Z = X \cap U
\end{align*}
\]

Lemma 6. A setting of the \( \delta \) variables in the linear program \( D \) can be extended to a feasible solution for \( D \) if and only if this same setting of the \( \delta \) variables can be extended to a feasible solution in \( D_{SCC} \).

Proof. Assume that a setting of the \( \delta \) variables is not feasible for \( D_{SCC} \). Then there must exist a \( j \) such that \( D[V_j] \) is infeasible. By Farkas’ lemma, if \( D[V_j] \) is not feasible, then there is polymatroid function \( F_j \) on the lattice of subsets of \( V_j \) such that

\[
\sum_{Z \subseteq V_j} \text{excess}^\delta(Z, V_j) \cdot F_j(Z) < F_j(V_j)
\]

We now use \( F_j \) to define a poly-matroid function \( F \) on the full lattice as follows:

\[
F(Z) = F_j(Z \cap V_j)
\]
Note that then \( \sum_{Z \subseteq [n]} \text{excess}(Z, V_j) F(Z) < F([n]) \). Thus by Farkas’ lemma \( D \)
must be infeasible.

Fix a collection of variables \( \delta, \mu^{\text{SCC}} \) and \( \sigma^{\text{SCC}} \) variables that are feasible
for \( D^{\text{SCC}} \). We want to show that \( \delta \) is feasible for \( D \) by setting \( \sigma \) and \( \mu \)
appropriately. The proof is by induction on the number of connected components of the
dependency graph. The induction invariant is that after \( j \) iterations a flow
of one has been routed to the set \( \cup_{i=1}^j V_i \) in the lattice. Initially, set all \( \sigma \) and \( \mu \)
variables to 0. Say that a unit flow has reached \( V_\star := \cup_{i=1}^{j-1} V_i \) inductively.
Consider iteration \( j \). We set the new variables as follows.

For each variable \( \sigma^{\text{SCC}}_{X,Y} \) in \( D[V_j] \) where \( X \subseteq Y \subseteq V_j \), set the variable
\( \sigma^{\text{SCC}}_{X \cup V_\star, Y \cup V_\star} \) in \( D \) to \( \sigma^{\text{SCC}}_{X,Y} \). For each variable \( \mu^{\text{SCC}}_{X,Y} \) in \( D[V_j] \) where \( X, Y \subseteq V_j \)
and \( X \perp Y \), set the variable \( \mu^{\text{SCC}}_{X \cup V_\star, Y \cup V_\star} \) in \( D \) to \( \mu^{\text{SCC}}_{X,Y} \). The value of the \( \delta_i \) are
the same in \( D \) and \( D^{\text{SCC}} \).

For each \( i \in [k] \) let \( m(i) \) be a real number such that for strongly connected
components \( V_i \) with \( j < m(i) \) it is the case that \( V_j \cap (Y_i \setminus X_i) = \emptyset \), for strongly
connected components \( V_j \) with \( j > m(i) \) it is the case that \( V_j \cap X_i = \emptyset \), and if
\( m(i) \) is an integer then it is the case that \( V_{m(i)} \cap Y_i \setminus X_i \neq \emptyset \) and \( V_{m(i)} \cap X_i \neq \emptyset \).
For each \( j > m(i) \) such that \( (Y_i \setminus X_i) \cap V_j \neq \emptyset \) then \( \sigma^{\text{SCC}}_{X_i \cup V_\star, Y_i \cup V_\star} = \delta_i \). This
pushes \( \delta_i \) units of flow from \( V^\star \) to \( V^\star \cup (Y_i \cap V_j) \) in \( D \), essentially replacing the \( \delta_i \)
flow in \( D[V_j] \), and pushes \( \delta_i \) units of flow from \( Y_i \cap (V^\star \cup V_j) \) to \( V^\star \cap Y_i \) in \( D \) (call
this a down push). And if \( m(i) \) is an integer then \( \sigma^{\text{SCC}}_{X_i \cup V_\star, Y_i \cup V_\star} = \delta_i \).
This pushes \( \delta_i \) units of flow from \( V^\star \cup (V_j \cap X_i) \) to \( V^\star \cup (Y_i \cap V_j) \) in \( D \), essentially
replacing the \( \delta_i \) flow in \( D[V_j] \), and pushes \( \delta_i \) units of flow from \( Y_i \cap (V^\star \cup V_j) \) to
\( (V^\star \cap X_i) \cup (Y_i \cap V_j) \) in \( D \) (call this a down push).

For each difference constraint \( i \in [k] \) we set some \( \mu \) variables in \( D \) as follows.

Let the down pushes of flow in \( D \) involving difference constraint \( i \) that have been
considered so far be: \( B_1 \rightarrow A_1, B_{k-1} \rightarrow A_{k-1}, \ldots, B_1 \rightarrow A_1 \) such that
\[
X_i \subseteq A_1 \subseteq B_1 \subseteq A_2 \subseteq B_2 \subseteq \ldots \subseteq A_k \subseteq B_k = Y_i
\]

We then connect these down pushes up by setting \( \mu^{\text{SCC}}_{X_i, A_1} = \delta_i \) if \( X_1 \neq A_1 \), and
setting each \( \mu^{\text{SCC}}_{B_i, A_{i+1}} = \delta_i \) if \( B_i \neq A_{i+1} \) and \( i \in [k-1] \). The down pushes and
these \( \mu \) variables together route \( \delta_i \) units of flow from \( Y_i \) to \( X_i \) in \( D \).

5 Hardness of Computing Normal Bounds

This section is devoted to proving the following theorem.

**Theorem 1.** The problem of maximizing the weighted coverage function value
subject to difference constraints, DC\([N_n]\), cannot be solved in polynomial time
unless \( P = NP \).

Recall that the weighted coverage bounds are obtained over functions that
are linear combinations of coverage functions. To prove the theorem we will
have to define several collections of difference constraints. Thus, we will directly
use \((X, Y)\) or \((X, Y, c)\) to denote a difference constraint; the former hides \(c\) for brevity. For a given collection of difference constraints, \(G\), the LP, \(D_S\), can be rewritten into the following equivalent form, \(D'_S\):

\[
\min \quad \sum_{(X, Y) \in G} c_{X,Y} \cdot \delta_{X,Y}
\]

such that

\[
\sum_{X \subseteq W, W \not\subseteq Y, (X, Y) \in G} \delta_{X,Y} \geq 1 \quad \forall W \subseteq [n],
\]

where the constraints here are equivalent to those in \(D_S\) by setting \(V = [n] \setminus W\).

Let \(\Delta(G)\) denote the convex region over \(\delta\) defined by the constraints in \(G\). We first show the separation problem is hard.

**Theorem 2.** Given a difference constraint set \(G\) and a vector \(\hat{\delta} \in \mathbb{R}^{G}_{\geq 0}\), checking if \(\hat{\delta} \not\in \Delta(G)\) is NP-complete. Further, this remains the case under the extra condition that \(\lambda \delta \in \Delta(G)\) 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. \(\Delta(G)\), we create an instance for computing weighted coverage bounds that has the elements \(E' = E \cup \{e^*\}\) and the following set \(G\) of difference constraints and \(\hat{\delta}\) (here we do not specify \(c_{X,Y}\) associated with each difference 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 \(\hat{\delta}_{\{e^*\}, E'} = m\).

Let \(G_1, G_2,\) and \(G_3\) denote the difference constraints defined above in each line respectively, and let \(G := G_1 \cup G_2 \cup G_3\). To establish the reduction we aim to show the following lemma.

**Lemma 7.** There exists a hitting set of size \(k\) in the original instance \(H\) if and only if \(\hat{\delta} \not\in \Delta(G)\).

**Proof.** Let \(L(W) := \sum_{(X, Y) \in G, X \subseteq W, W \not\subseteq Y} \hat{\delta}_{Y|X}\). Let \(L_{\min} := \min_{W \subseteq E'} L(W)\) and \(W_{\min} := \arg \min_{W \subseteq E'} L(W)\). 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 \(\hat{\delta}(G') := \sum_{(X, Y) \in G'} \hat{\delta}_{X,Y}\). Note that \(L_{\min} \leq L(\emptyset) = \hat{\delta}(G_1) = \frac{m}{k+1}\).

Therefore, we can have the following conclusions about \(W_{\min}\).
e* ∉ W min since otherwise L min ≥ ³(G3) = m.
- For all S_i ∈ S, S_i ∉ W min since otherwise L min ≥ ³(\{S_i, E'\}) = m.

Thus, we have shown that only the difference constraints in G_1 can contribute to L min. As a result,

\[ L_{min} = L(W_{min}) = ³(\{\emptyset, \{e_i\}\} \in G_1 : \{e_i\} \not\subseteq W_{min}) = \frac{1}{k+1} |E \setminus W_{min}|. \]

As observed above, for all S_i ∈ S, S_i ∉ W min, which means (E \ W min) \cap S_i ≠ ∅. This immediately implies that E \ W min is a hitting set.

To recap, if ³ ∉ Δ(G), we have \(\frac{1}{k+1}|E \setminus W_{min}| < 1\) and therefore the original instance H admits a hitting set E \ W min of size at most k.

Conversely, if the instance H admits a hitting set E' of size k, we can show that \(L(E \setminus E') = ³(\{\emptyset, \{e_i\}\} \in G_1 : \{e_i\} \not\subseteq E') = \frac{k}{k+1} < 1\), which means ³ ∉ Δ(G). This direction is essentially identical and thus is omitted. □

The above lemma shows checking ³ ∉ Δ(G) is NP-hard. Further, a violated constraint can be compactly represented by W_i; thus the problem is in NP. Finally, if we scale up ³ by a factor of λ = k+1, we show λ³ ∉ Δ(G). We consider two cases. If E ∉ W, we have \(L(W) ≥ ³(\{i \in [n] : e_i ∉ W\}) ≥ \frac{1}{k+1}λ = 1\). If E ⊆ W, it must be the case that E = W since W ≠ E' and E' = E \ {e*}. In this case \(L(E) = ³(G_2) ≥ mλ ≥ 1\). Thus, for all W ⊆ E', we have L(W) ≥ 1, meaning λ³ ∉ Δ(G). This completes the proof of Theorem 2.

Using this theorem, we want to show that we can’t solve D_2 in polynomial time unless P = NP. While there exist relationships among the optimization problem, membership problem and their variants [GLS12], 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

**Theorem 3.** We cannot solve LP normal in polynomial time unless P = NP.

**Proof.** Consider an instance to the membership problem consisting of G and ³. By Theorem 2 we know checking ³ ∉ Δ(G) is NP-complete, even when λ³ ∉ Δ(G) for some λ > 1. For the sake of contradiction, suppose we can solve LP normal in polynomial time for any w ≥ 0 over the constraints defined by the same Δ(G). We will draw a contradiction by showing how to exploit it to check ³ ∉ Δ(G) in polynomial time.

Define R := \{w | w \cdot (³ - ³) > 0 \ ∀³ \in Δ(G)\}. It is straightforward to see that R is convex.

We claim that ³ ∉ Δ(G) iff R ≠ ∅. To show the claim suppose ³ ∉ Δ(G). Recall from Theorem 2 that there exists λ > 1 such that λ³ ∉ Δ(G). Let λ' > 0 be the smallest λ" such that λ'³ ∈ Δ(G). Observe that λ' > 1 and λ³ lies on a facet of Δ(G), which corresponds to a hyperplane \(\sum_{X⊂ W, W ∋ Y, (X,Y) ∈ G} δ_{X,Y} = 1\)
for some $W \subset [n]$. Let $w$ be the orthogonal binary vector of the hyperplane; so we have $w \cdot \lambda' \hat{\delta} = 1$. Then, $w \cdot (\delta - \hat{\delta}) \geq 0$ for all $\delta \in \Delta(G)$. Thus, for any $\delta \in \Delta(G)$ we have $w \cdot (\delta - \hat{\delta}) \geq (\lambda' - 1)w \cdot \lambda \hat{\delta} = \frac{\lambda' - 1}{\lambda} > 0$. The other direction is trivial to show: If $\hat{\delta} \in \Delta(G)$, 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 $\hat{\delta}$ that are vertices of $\Delta(G)$. Further, $\Delta(G)$ is defined by a finite number of (more exactly at most $2^n$) constraints (one for each $W$). This implies that the following LP,

$$\max \epsilon \quad w \cdot (\delta - \hat{\delta}) \geq \epsilon \quad \forall \delta \in \Delta(G)$$

$$w \geq 0$$

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 $\epsilon$, to determine if $w \cdot (\delta - \hat{\delta}) \geq \epsilon$ for all $\delta \in \Delta(G)$; otherwise it should find a $\delta \in \Delta(G)$ such that $w \cdot (\delta - \hat{\delta}) < \epsilon$. In other words, we want to know $\min_{\delta \in \Delta(G)} w \cdot (\delta - \hat{\delta})$. If the value is no smaller than $\epsilon$, all constraints are satisfied, otherwise, we can find a violated constraint, which is given by the $\delta$ minimizing the value. But, because the oracle assumes $w \cdot \hat{\delta}$ is fixed, so this optimization is essentially the same as solving $LP_{normal}$, 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.

\section{Hard Special Cases}

In this section we present two classes of seemingly simple instances which turn out to be as hard as general instances.

\subsection{Reduction from General DCs to Acyclic DCs and Simple FDs}

\textbf{Theorem 4.} 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 difference constraints (DCs) can be divided into two subsets of acyclic DCs and simple functional dependencies (FDs)—further, each FD contains exactly two elements.

\textbf{Reduction:} Suppose we are given an arbitrary instance $I$ consisting of the universe $U := [n]$ and a set $G$ of DCs. 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 $G'$ of DCs. For each $i \in [n]$, we first add the following simple functional dependencies
to $G'$:

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

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

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

**Lemma 8.** Let $g$ be a submodular function that satisfies $G'$ of $I'$. For any $i \in [n]$ and any $B \subseteq U'$ such that $x_i, y_i \notin B$, we have $g(B \cup \{x_i\}) = g(B \cup \{y_i\}) = g(B \cup \{x_i, x_2\})$.

**Proof.** By submodularity and a FD in $G'$ involving $x_i, y_i$, we have:

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

Mototonicity implies $g(B \cup \{x_i, y_i\}) - g(B \cup \{x_i\}) \geq 0$ which means $f(B \cup \{x_i, y_i\}) = f(B \cup \{x_i\})$. 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 Theorem 3.

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

We first show the first direction. Let $h : 2^U \to 2^U$ be a set function that converts a subset of $U'$ to a subset of $U$ by counting $x_i$ and $y_i$ only once for each $i \in [n]$. Formally, $i \in h(B)$ iff $x_i \in B$ or $y_i \in B$. Then, we set $g(B) := f(h(B))$.

**Lemma 9.** For any $A, B \subseteq U'$, $h(A) \cup h(B) = h(A \cup B)$ and $h(A) \cap h(B) \geq h(A \cap B)$.

**Proof.** The first claim follows because if $x_i$ or $y_i$ is in any of $A$ and $B$, it is also in $A \cup B$. The second claim 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 $i \in h(A) \cap h(B)$.

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,

$$g(A) + g(B) = f(h(A)) + f(h(B)) \geq f(h(A) \cup h(B)) + f(h(A) \cap h(B)) \geq g(h(A \cup B)) + g(h(A \cap B)) = f(A \cup B) + f(A \cap B),$$
where the first inequality follows from \( f \)'s submodularity and the second from \( f \)'s monotonicity and Lemma 3. Thus we have shown the first direction.

To show the other direction, we define \( f(A) \) to be \( g(B) \) where \( i \in A \) iff \( x_i \in B \). By definition, we have \( f(U) = g({x_1, x_2, \ldots, x_n}) \). Further, by repeatedly applying Lemma 8 we have \( g({x_1, x_2, \ldots, x_n}) = g(U) \). 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 Theorem 3.

6.2 Reduction from General DCs to DCs \((X, Y, d)\) with \(|X| \leq 2\)

**Theorem 5.** There is a polynomial-time reduction from a general instance to an instance preserving the polymatroid bound, where for each difference constraint \((X, Y, d)\) we have \(|Y| \leq 3\) and \(|X| \leq 2\). Further, the new instance satisfies the following:

- If \(|Y| = 3\), then \(|X| = 2\) and \(d = 0\).
- If \(|Y| = 2\), then \(|X| = 1\).

**Reduction:** The high-level idea is to repeatedly replace two variables with a new variable in a difference constraint. We first discuss how to choose two variables to combine. Assume there is a difference constraint \((X, Y, d)\) over all non-trivial constraints. Observe that

\[
\{x_1, x_2, \ldots, x_n\} = \{0\} + \{n-1, n\}.
\]

By renaming, we can assume wlog that we combine variables \(n-1\) and \(n\) into a new variable \(0\) in a difference constraint \((X, Y, d) \in G\). Then, we create \((X', Y', d)\) and add it to \(G'\) where

\[
(X', Y', d) := \begin{cases} 
(X \setminus \{n-1, n\} \cup \{0\}, Y \setminus \{n-1, n\} \cup \{0\}, d) & \text{if } \{n-1, n\} \subseteq X, Y \\
(X, Y \setminus \{n-1, n\} \cup \{0\}, d) & \text{if } \{n-1, n\} \subseteq Y \setminus X
\end{cases}
\]

Further, we add functional dependencies \((\{n-1, n\}, \{0, n-1, n\}, 0), (\{0\}, \{0, n-1\}, 0)\) and \((\{0\}, \{0, n\}, 0)\) to \(DC'\), which we call consistency constraints. Intuitively, consistency constraints imply we have variable \(0\) if and only if we have both \(n-1\) and \(n\). The other constrains are called non-trivial constraints.

**Observation 6** If \(X \neq X'\), then we have \(X' = X \setminus \{n-1, n\} \cup \{0\}\) and \(X = X' \setminus \{0\} \cup \{n-1, n\}\). A similar observation holds for \(Y\) and \(Y'\).

**What do we have after repeatedly applying this reduction?** Let's first see why the reduction process terminates. For a difference constraint \((X, Y, d)\), Define \(c(X, Y, d) = |X| + |Y|\) for a non-trivial constraint \((X, Y, d)\). The potential is defined as the total sum of \(c(X, Y, d)\) over all non-trivial constraints. Observe that
in each iteration, either \(c(X, Y, d) > c(X', Y', d)\), or \(|X'| = 1\) and \(|Y'| = 2\). In the latter case, the resulting non-trivial constraint \((X', Y', d)\) doesn’t change in the subsequent iterations. In the former case the potential decreases. Further, initially the potential is at most \(2n|G|\) and the number of non-trivial constraints never increases, where \(G\) is the set of difference constraints initially given. Therefore, the reduction terminates in a polynomial number of iterations. It is now straightforward to see that we only have difference constraints of the forms that are stated in Theorem 5 at the end of the reduction.

**Reduction Preserves the Polymatroid Bound.** We consider one iteration where a non-trivial constraint \((X, Y, d)\) is replaced according to the reduction described above. Let \(\text{opt}\) and \(\text{opt}'\) be the polymatroid bounds before and after performing the iteration respectively. Let \(G\) and \(G'\) be the sets of the difference constraints before and after the iteration respectively.

We first show \(\text{opt} \geq \text{opt}'\). Let \(g : \{0\} \cup [n] \to [0, \infty)\) be a monotone submodular function that achieves \(\text{opt}'\) subject to \(G'\). Define \(f : [n] \to [0, \infty)\) such that \(f(A) = g(A)\) for all \(A \subseteq [n]\). It is immediate that \(f\) is monotone and submodular from \(g\) being monotone and submodular, as we only restricted the function to \([n]\). The following claim shows that having 0 is equivalent to having \(n - 1\) and \(n\) in evaluating \(g\).

**Claim.** For any \(X\), \(g(X \cup \{0\}) = g(X \cup \{0, n - 1, n\}) = g(X \cup \{n - 1, n\})\).

**Proof.** Due to the consistency constraints and \(g\)’s monotonicity, we have \(g(\{0, n - 1, n\}) = g(\{n - 1, n\})\). Because of the consistency constraints we added and \(g\)’s submodularity, we have \(0 \geq g(\{0, n - 1\}) - g(\{0\}) \geq g(\{0, n - 1, n\}) - g(\{0, n\})\).

Then due to the monotonicity, we have \(g(\{0\}) = g(\{0, n - 1\})\) and \(g(\{0, n - 1, n\}) = g(\{0, n - 1\})\). Thus, we have shown that \(g(\{0\}) = g(\{0, n - 1\})\).

The first equality in the claim follows since \(0 = g(\{0, n - 1, n\}) - g(\{0\}) \geq g(X \cup \{0, n - 1, n\}) - g(X \cup \{0\}) \geq 0\). The second equality can be shown similarly. \(\Box\)

We now check if \(f\) satisfies \(G\). Because we only replaced \((X, Y, d) \in G\), we only need to show that \(f\) satisfies it. We need to consider two case:

- When \(\{n - 1, n\} \subseteq X \subseteq Y\). Then, we have \(g(Y \cup \{0\} \setminus \{n - 1, n\}) - g(X \cup \{0\} \setminus \{n - 1, n\}) \leq d\). By Claim 6.2, we have \(g(Y) - g(X) = g(Y \cup \{n - 1, n\}) - g(X \cup \{n - 1, n\}) \leq d\). By definition of \(f\), we have \(f(Y) - f(X) \leq d\).
- When \(\{n - 1, n\} \subseteq Y \setminus X\). In this case, \(X' = X\) and \(Y' = Y \cup \{0\} \setminus \{n - 1, n\}\); thus we have \(g(Y \cup \{0\} \setminus \{n - 1, n\}) - g(X) \leq d\). Thanks to Claim 6.2 and \(f\)'s definition, we have \(f(Y) - f(X) \leq d\), as desired.

Finally, \(f([n]) = g([n]) = g([n] \cup \{0\}) = \text{opt}'\) due to Claim 6.2. Since we have shown \(f\) is a feasible solution for \(G\), we have \(\text{opt} \geq f([n])\). Thus, we have \(\text{opt} \geq \text{opt}'\) as desired.
We now show $\text{opt} \leq \text{opt}'$. Given $f$ that achieves $\text{opt}$ subject to $G$, we construct $g : \{0\} \cup [n] \rightarrow [0, \infty)$ as follows:

$$g(A) := \begin{cases} f(A) & \text{if } 0 \not\in A \\ f(A \setminus \{0\} \cup \{n-1, n\}) & \text{otherwise} \end{cases} \quad \text{(5)}$$

We first verify that $g$ is monotone. Consider $A \subseteq B \subseteq \{0\} \cup [n]$. If $0 \not\in A$ and $0 \not\in B$, or $0 \in A$ and $0 \in B$, it is easy to see that is the case. So, assume $0 \not\in A$ but $0 \in B$. By definition of $g$, it suffices show $f(A) \leq f(B \setminus \{0\} \cup \{n-1, n\})$, which follows from $f$’s monotonicity: Since $0 \not\in A$ and $A \subseteq B$, we have $A \subseteq B \setminus \{0\} \cup \{n-1, n\}$.

Secondly we show that $g$ is submodular. So, we want to show that $g(A) + g(B) \geq g(A \cup B) + g(A \cap B)$ for all $A, B \subseteq \{0\} \cup [n]$.

- When $0 \not\in A$ and $0 \not\in B$. This case is trivial as $g$ will have the same value as $f$ for all subsets we’re considering.
- When $0 \in A$ and $0 \in B$. We need to check if $f(A \setminus \{0\} \cup \{n-1, n\}) + f(B \setminus \{0\} \cup \{n-1, n\}) \geq f(A \cup B \setminus \{0\} \cup \{n-1, n\})$, which follows from $f$’s submodularity. More concretely, we set $A' = A \setminus \{0\} \cup \{n-1, n\}$ and $B' = B \setminus \{0\} \cup \{n-1, n\}$ and use $f(A') + f(B') \geq f(A' \cup B') + f(A' \cap B')$.
- When $0 \in A$ and $0 \not\in B$ (this is symmetric to $0 \not\in A$ and $0 \in B$). We need to check if

$$f(A \setminus \{0\} \cup \{n-1, n\}) + f(B) \geq f(A \cup B \setminus \{0\} \cup \{n-1, n\}) + f(A \cap B).$$

For $A' = A \setminus \{0\} \cup \{n-1, n\}$, we have $f(A') + f(B) \geq f(A' \cup B) + f(A' \cap B)$. So, it suffices to show

$$f(A' \cup B) + f(A' \cap B) \geq f(A \cup B \setminus \{0\} \cup \{n-1, n\}) + f(A \cap B)$$

Because $A' \cup B = A \cup B \setminus \{0\} \cup \{n-1, n\}$, this is equivalent to showing:

$$f(A' \cap B) \geq f(A \cap B)$$

$$\iff f(A' \cap (B \setminus \{0\})) \geq f((A \setminus \{0\}) \cap (B \setminus \{0\}))$$

$$\iff A' \supseteq (A \setminus \{0\}) \quad \text{[Due to } f’s \text{ monotonicity]}$$

Thirdly, we show that $g$ satisfies $G'$. Suppose we replaced a non-trivial constraint $(X, Y, d)$ with $(X', Y', d)$. We show $g(Y') - g(X') \leq d$ by showing $f(Y) = g(Y')$ and $f(X) = g(X')$. Both cases are symmetric, so we only show $f(X') = g(X)$. If $0 \not\in X'$, then clearly we have $g(X') = f(X)$ since $X' = X$. If $0 \in X'$, then it must be the case that $X' = X \setminus \{n-1, n\} \cup \{0\}$. By definition of $g$, we have $g(X') = f(X' \setminus \{0\} \cup \{n-1, n\}) = f(X)$ since $X' \setminus \{0\} \cup \{n-1, n\} = X$.

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

- $g(\{0, n - 1, n\}) \leq g(\{0\})$. Note $g(\{0, n - 1, n\}) = f(\{n - 1, n\}) = g(\{0\})$ by definition of $g$. Due to $g$’s monotonicity we have already shown, we have $g(\{0, n - 1\}) \leq g(\{0\})$ and $g(\{0, n\}) \leq g(\{0\})$. 

\[ g(\{0, n - 1, n\}) \leq g(\{n - 1, n\}) \]. Both sides are equal to \( f(\{n - 1, n\}) \) by definition of \( g \).

Finally, we have \( g(\{0\} \cup [n]) = f([n]) \). Since \( g \) is a monotone submodular function satisfying \( G \), we have \( \text{opt}' \geq \text{opt} \) as desired.

This completes the proof of Theorem 5.

References

AKNS20. Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. Bag query containment and information theory. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020, pages 95–112. ACM, 2020.

ANS16. Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In Tova Milo and Wang-Chiew Tan, editors, Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016, pages 327–342. ACM, 2016.

ANS17. Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017, pages 429–444. ACM, 2017.

GLS12. Martin Grötschel, László Lovász, and Alexander Schrijver. Geometric algorithms and combinatorial optimization, volume 2. Springer Science & Business Media, 2012.

KY22. Lukas Kühne and Geva Yashfe. On entropic and almost multilinear representability of matroids, 2022.

Li21. Cheuk Ting Li. The undecidability of conditional affine information inequalities and conditional independence implication with a binary constraint. In 2021 IEEE Information Theory Workshop (ITW), page 1–6. IEEE Press, 2021.

Ngo18. Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In Jan Van den Bussche and Marcelo Arenas, editors, Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018, pages 111–124. ACM, 2018.

Ngo22. Hung Q. Ngo. On an information theoretic approach to cardinality estimation (invited talk). In Dan Olteanu and Nils Vortmeier, editors, 25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference), volume 220 of LIPIcs, pages 1:1–1:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

Stu05. Milan Studený. Probabilistic conditional independence structures. Information Science and Statistics. Springer, London, 2005.

Yeu02. Raymond W. Yeung. A first course in information theory. Kluwer Academic Publishers, 2002.