ELSEVIER

Theoretical Computer Science 193 (1998) 149–179

Theoretical Computer Science

Foundations of aggregation constraints[1]

Kenneth A. Ross[a,2], Divesh Srivastava[b,*], Peter J. Stuckey[c,3], S. Sudarshan[d,4]

[a] *Columbia University, New York, NY 10027, USA*
[b] *AT&T Labs – Research, 600–700 Mountain Avenue, Murray Hill, NJ 07974, USA*
[c] *University of Melbourne, Parkville 3052, Australia*
[d] *Indian Institute of Technology, Powai, Mumbai 400 076, India*

Received June 1995; revised November 1996
Communicated by M. Nivat

Abstract

We introduce a new constraint domain, *aggregation constraints*, that is useful in database query languages, and in constraint logic programming languages that incorporate aggregate functions. We formally study the fundamental problem of determining if a conjunction of aggregation constraints is *satisfiable*, and show that, for many classes of aggregation constraints, the problem is undecidable. We describe a complete and minimal axiomatization of aggregation constraints, for the SQL aggregate functions *min*, *max*, *sum*, *count* and *average*, over a non-empty, finite multiset on several domains. This axiomatization helps identify classes of aggregation constraints for which the satisfiability check is efficient. We present a polynomial-time algorithm that directly checks for satisfiability of a conjunction of aggregation range constraints over a single multiset; this is a practically useful class of aggregation constraints. We discuss the relationships between aggregation constraints over a non-empty, finite multiset of reals, and constraints on the elements of the multiset. We show how these relationships can be used to push constraints through aggregate functions to enable compile-time optimization of database queries involving aggregate functions and constraints.

1. Introduction

Database query languages such as SQL use aggregate functions (such as *min*, *max*, *sum*, *count* and *average*) to obtain summary information from the database. Aggregate

* Corresponding author. E-mail: divesh@research.att.com.
[1] A preliminary version of this paper appeared in [5].
[2] Supported by NSF grant IRI-9209029, by a grant from the AT&T Foundation, by a David and Lucile Packard Foundation Fellowship in Science and Engineering, by a Sloan Foundation Fellowship, and by an NSF Young Investigator Award.
[3] Partially supported by the Centre for Intelligent Decision Systems and ARC Grant A49130842.
[4] Work was performed while he was at AT&T Bell Laboratories, Murray Hill, NJ 07974, USA.

functions are typically used in combination with a grouping facility: values are partitioned into groups and aggregate functions are applied to the multiset of values within each group. Database query languages also allow constraints (e.g., $M1>0$, $M2\leqslant 10\,000$) to be specified on values, in particular on the results of aggregate functions, to restrict the answers to a query.

In this paper, we formally study constraints on the results of aggregate functions on multisets; we refer to this constraint domain as *aggregation constraints*. This is a novel constraint domain that is useful in database query languages, and in constraint logic programming languages that incorporate aggregate functions [3]. We make the following contributions in this paper:

1. We study the fundamental problem of determining if a conjunction of aggregation constraints is *satisfiable*, and show that, for many classes of aggregation constraints, the problem is undecidable (Section 3).
2. We describe a *complete* and *minimal* axiomatization of aggregation constraints, for the aggregate functions *min*, *max*, *sum*, *count* and *average*, over a non-empty, finite multiset on several domains (Section 4). These aggregate functions are exactly those supported in SQL-92 [4]. The axiomatization enables a natural reduction from this class of aggregation constraints to the class of mixed integer/real, non-linear arithmetic constraints. This axiomatization also helps identify interesting classes of aggregation constraints for which the satisfiability check is efficient.
3. We present a polynomial-time algorithm that checks for satisfiability of a conjunction of aggregation *range* constraints, for the SQL aggregate functions, over a non-empty, finite multiset of reals (Section 5 and Appendix). Our algorithm operates directly on the aggregation constraints, rather than on the reduced form obtained using the axiomatization; it is not clear how to operate directly on the reduced form to attain the same complexity.
4. We discuss the relationships between aggregation constraints over a non-empty, finite multiset of reals, and constraints on the elements of the multiset. In Section 6, we describe how to infer aggregation constraints on a multiset, given constraints on the elements of the multiset. In Section 7, we describe how to infer constraints on multiset elements, given aggregation constraints on the multiset.
5. We show how aggregation constraints on queries (i.e., *query constraints* involving aggregation) can be used for compile-time database query optimization. (Section 8).

Example 1.1 (*Illustrative Example*). Let `E` denote an employee relation with attributes `Emp` denoting the employee identifier, `Dept` denoting the employee's department, and `Salary` denoting the employee's salary. The following view `V` defines departments (and aggregates of their employees' salaries) where the minimum salary is greater than 0, where the maximum salary is less than or equal to 10 000 and where the number of employees is less than or equal to 10:

```
CREATE   VIEW V(Dept, Min-Sal, Max-Sal, Sum-Sal, Count-Emp) AS
SELECT   Dept, MIN(Salary), MAX(Salary), SUM(Salary), COUNT(Salary)
FROM     E
GROUP BY Dept
HAVING   COUNT(Salary)⩽10 AND MIN(Salary)>0
         AND MAX(Salary)⩽10000
```

Consider the query Q given by

```
SELECT   *
FROM     V
WHERE    Sum-Sal>100000
```

To determine (at compile-time, by examining only the view definition and the query, but not the database) that there are no answers to this query, we need to determine that, independent of the actual tuples in the employee relation E, the conjunction of aggregation constraints: $min(S)>0 \wedge count(S) \leqslant 10 \wedge max(S) \leqslant 10\,000 \wedge sum(S) > 100\,000$ is unsatisfiable, where S is a non-empty, finite multiset of salaries. This can be determined by observing that the results of different aggregate functions on a multiset S are not independent of each other. For example, the results of the *sum*, *count* and *max* aggregate functions are related as follows:

$$sum(S) \leqslant count(S) * max(S).$$

This inequality can be used to infer the unsatisfiability of the previous conjunction of aggregation constraints, and hence determine that the query Q has no answers. The techniques described in this paper can be used to efficiently check for satisfiability of such aggregation constraints.

Checking satisfiability of aggregation constraints can be used much like checking satisfiability of ordinary arithmetic constraints in a constraint logic programming system like CLP($\mathscr{R}$) [1]. Aggregate functions are typically applied only after multisets have been constructed. However, checking satisfiability of aggregation constraints even before the multisets have been constructed can be used to restrict the search space by not generating subgoals that are guaranteed to fail, as illustrated by the above view and query.

Our work provides the *foundations* of the area of aggregation constraints. We believe there is a lot of interesting research to be done in the further study of aggregation constraints, e.g., the relationships between aggregation constraints on different multisets that are related by multiset functions and predicates such as $\cup, \cap, \subseteq$, applications of aggregation constraints to query optimization, database integrity constraints and constraint logic programming.

2. Aggregation constraints

The constraint domain we study is specified by the class of first-order languages $L(J)$, where $J \subseteq \mathscr{R}$ is an arithmetic domain, and $\mathscr{R}$ denotes the reals. For example,

J can denote the reals, the integers, the non-negative integers, etc. The distinguished sorts in $L(J)$ are:

- the *atomic* sorts, which include J, the non-negative integers $\mathcal{N}$, the positive integers $\mathcal{N}^+$, and the sort $J/\mathcal{N}^+$ (e.g., $\mathcal{N}/\mathcal{N}^+$ denotes the non-negative rationals, and $\mathcal{R}/\mathcal{N}^+ = \mathcal{R}$), and
- the *multiset* sorts, which include finite multisets of elements from J, denoted by $\mathcal{M}(J)$, and non-empty, finite multisets of elements from J, denoted by $\mathcal{M}^+(J)$. Clearly, $\mathcal{M}(J)$ contains $\mathcal{M}^+(J)$.

Apart from the set J over which we are defining multisets, we need to introduce the sorts $\mathcal{N}$ and $J/\mathcal{N}^+$ in order to define the return values for the aggregate functions *count* and *average*, respectively.

Constants of the atomic sorts are in $L(J)$. Variables of sort $\mathcal{M}(J)$ and $\mathcal{M}^+(J)$ are called multiset variables, and are usually denoted by S, S_1, etc. For simplicity, we do not consider variables of the atomic sorts in our treatment.

Multiplication and addition functions on the atomic sorts $J, \mathcal{N}, \mathcal{N}^+$ and $J/\mathcal{N}^+$ (and between these sorts) are in $L(J)$. We require that each of J, $\mathcal{N}$, $\mathcal{N}^+$, and $J/\mathcal{N}^+$ is closed under addition and multiplication, as is any union of these sorts.

The *aggregate functions* are the functions *sum*, *min*, *max*, *count* and *average* in $L(J)$. The functions *sum*, *min*, and *max* take arguments from $\mathcal{M}^+(J)$ and return a value of sort J. The function *count* takes arguments from $\mathcal{M}(J)$ and returns a value of sort $\mathcal{N}$. The function *average* takes arguments from $\mathcal{M}^+(J)$ and returns a value of sort $J/\mathcal{N}^+$.

The *primitive terms* of $L(J)$ are constants of the atomic sorts, and *aggregation terms*, which are formed using aggregate functions on multiset variables. Thus, 7, 3.142 and $max(S)$, where S is a multiset variable of type $\mathcal{M}^+(\mathcal{R})$, are primitive terms of $L(\mathcal{R})$. *Complex terms* are constructed using primitive terms and arithmetic functions such as $+$ and $*$. Thus, $min(S_1) * max(S_2) + (-3.142) * count(S_2)$ is a complex term in $L(\mathcal{R})$.

A *primitive aggregation constraint* in $L(J)$ is constructed using complex terms and arithmetic predicates such as $\leqslant, <, =, \neq, >$ and $\geqslant$, which take arguments of the atomic sorts $J, \mathcal{N}, \mathcal{N}^+$ and $J/\mathcal{N}^+$. Thus, $sum(S_1) \leqslant min(S_1) + max(S_2) + 3.1$ is a primitive aggregation constraint in $L(\mathcal{R})$. Complex aggregation constraints can be constructed using conjunction, disjunction and complementation, in the usual manner. However, in this paper, we shall deal *only* with conjunctions of primitive aggregation constraints. Thus, for our purposes, an *aggregation constraint* is a conjunction of primitive aggregation constraints. Note that the multiset variables cannot be quantified in $L(J)$.

Given an aggregation term E, an *aggregation range constraint* on E is a conjunction of primitive aggregation constraints, where each primitive constraint is of the form E *op* c or of the form c *op* E, *op* is one of $<$ and $\leqslant$, and c is a constant of an atomic sort.

2.1. Satisfiability

Given a sort J for multiset elements, an argument of an aggregate function in $\{min,\ max,\ sum,\ count,\ average\}$ is said to be *well-typed*, if it matches the signature of the aggregate function. Thus, S in $max(S)$ is well-typed if it is of type $\mathcal{M}^+(J)$.

The notion of multiset assignments, θ, of values to free variables (here, the multiset variables) is defined in the usual way. Given a sort J, an assignment is said to be well-typed if each of the variables in the assignment is well-typed for the aggregate functions it participates in.

We are interested in the following fundamental problem.

Satisfiability: Given a conjunction $\mathscr{C}$ of primitive aggregation constraints, does there exist a well-typed assignment θ of multisets to the multiset variables in $\mathscr{C}$, such that $\mathscr{C}\theta$ is satisfied?

Checking for satisfiability of more complex aggregation constraints can be reduced to this fundamental problem. The other important problems of checking *implication* (or entailment) and *equivalence* of pairs of aggregation constraints can be reduced to checking satisfiability of other aggregation constraints, in polynomial-time.

Example 2.1. Consider the aggregation constraint

$$max(S) = 2 * min(S) \wedge count(S) \geqslant 3$$

where S is of type $\mathcal{M}^+(\mathcal{N})$. A multiset assignment that satisfies the constraint is $\{S \mapsto \{2,3,4\}\}$. Hence, the above aggregation constraint is satisfiable.

We can determine that one aggregation constraint implies ($\rightarrow$) another using satisfiability. For example, we can determine whether

$$max(S) = 2 * min(S) \wedge count(S) \geqslant 3 \rightarrow sum(S) \geqslant 0$$

where S is of type $\mathcal{M}^+(\mathcal{N})$, by testing the satisfiability of

$$max(S) = 2 * min(S) \wedge count(S) \geqslant 3 \wedge sum(S) < 0.$$

As this aggregation constraint is unsatisfiable, the implication holds.

2.2. A taxonomy

We present below several factors that affect the complexity of checking for satisfiability, and in later sections present algorithms for checking satisfiability of special cases of aggregation constraints, defined on the basis of these factors.

Domain of multiset elements: This determines the feasible assignments to the multiset variables in checking for satisfiability. Possibilities include integers and reals; correspondingly, the multiset variables range over finite multisets of integers or finite multisets of reals. In general, restricting the domain of the multiset elements to integers increases the difficulty of the problem.

Operations: If we allow just addition and multiplication, solving constraints may be easier than if we also allowed exponentiation, for example.

Aggregate functions: This determines the possible aggregate functions that are allowed in constructing aggregation terms. In general, the complexity of checking for satisfiability increases if more aggregate functions are allowed.

Class of constraints: This determines the form of the primitive aggregation constraints considered. There are at least two factors that are relevant:

1. *Linear* vs. *Non-linear* constraints: Checking for satisfiability of linear constraints is, in general, easier than for non-linear constraints. By restricting the form even further, such that each primitive aggregation constraint has at most one or two aggregation terms, the problem can become even simpler.
2. *Constraint predicates* allowed: The complexity of checking for satisfiability also depends on which types of constraint predicates are allowed. We can choose to allow only equational constraints ($=$) or add inequalities ($<, \leqslant$) or possibly even disequalities ($\neq$). In general, the difficulty of the satisfiability problem increases with each new type.

Separability. This also determines the form of the primitive aggregation constraints considered. The two possible dimensions in this case are:

1. *Multiset variables*: A conjunction of primitive aggregation constraints is said to be *multiset-variable-separable* if each primitive aggregation constraint involves only one multiset variable. For example, the conjunction of primitive aggregation constraints $min(S_1) + max(S_1) \leqslant 5 \wedge sum(S_2) \geqslant 10$ is multiset-variable-separable, while $min(S_1) + min(S_2) \leqslant 10$ is not. In general, multiset-variable-separability makes the satisfiability problem easier since one can check satisfiability of the aggregation constraints separately for each multiset variable.
2. *Aggregate functions*: A conjunction of primitive aggregation constraints is said to be *aggregate-function-separable* if each primitive aggregation constraint involves only one aggregate function. For example, the conjunction $min(S_1) \leqslant min(S_2) \wedge sum(S_1) \geqslant sum(S_2) + 2$ is aggregate-function-separable. Note that this conjunction is not multiset-variable-separable.

3. Undecidability results

We show undecidability of checking satisfiability of conjunctions of primitive aggregation constraints by a linear-time, linear-space reduction from quadratic arithmetic constraints over the positive integers to *linear* aggregation constraints over non-empty, finite multisets of reals. The reduction makes essential use of the relationships $sum(S) = count(S) * average(S)$, and $min(S) = max(S)$ implies $sum(S) = count(S) * min(S)$.

Theorem 3.1. *Checking satisfiability of a linear aggregation constraint $\mathscr{C}$ over non-empty, finite multisets of reals is undecidable if*

1. *$\mathscr{C}$ involves the sum, count and average aggregate functions, or*
2. *$\mathscr{C}$ involves the sum, min, max and count aggregate functions.*

Proof. Consider a conjunction $\mathscr{C}$ of quadratic primitive arithmetic constraints over the positive integers. Replace each quadratic term $X_j * X_k$ (where X_j and X_k are not necessarily distinct variables) in $\mathscr{C}$ by a "new" positive integer variable X_i, and conjoin a quadratic equation of the form $X_i = X_j * X_k$ to $\mathscr{C}$. The resulting conjunction of constraints $\mathscr{C}_1$ is equivalent to $\mathscr{C}$ (on the variables of $\mathscr{C}$). Further, $\mathscr{C}_1$ contains only linear arithmetic constraints and quadratic equations of the form $X_i = X_j * X_k$ over the positive integers.

For each variable X_i in $\mathscr{C}_1$, the reduction algorithm creates a new multiset variable S_i of type $\mathscr{M}^+(\mathscr{R})$, and replaces each occurrence of X_i in the linear arithmetic constraints of $\mathscr{C}_1$ by the aggregation term $count(S_i)$. For each quadratic equation of the form $X_i = X_j * X_k$ in $\mathscr{C}_1$, the reduction algorithm creates a new multiset variable S_{ijk} of type $\mathscr{M}^+(\mathscr{R})$, and replaces the above quadratic equation by the following three linear aggregation equations:

$$count(S_i) = sum(S_{ijk})$$

$$count(S_j) = count(S_{ijk})$$

$$count(S_k) = average(S_{ijk})$$

The resulting conjunction of linear aggregation constraints $\mathscr{C}_2$ is satisfiable over non-empty, finite multisets of reals if and only if the original conjunction of quadratic constraints $\mathscr{C}$ is satisfiable over the positive integers.

There is a similar reduction using the aggregate functions *sum*, *min*, *max* and *count*, where the quadratic arithmetic equation $X_i = X_j * X_k$ is replaced by the following four linear aggregation equations: $count(S_i) = sum(S_{ijk})$, $count(S_j) = count(S_{ijk})$, $count(S_k) = min(S_{ijk})$ and $count(S_k) = max(S_{ijk})$. Again, the resulting conjunction of linear aggregation constraints is satisfiable over non-empty, finite multisets of reals if and only if the original conjunction of quadratic constraints is satisfiable over the positive integers.

The theorem follows from the undecidability of the satisfiability of quadratic arithmetic constraints over the positive integers (e.g., Diophantine equations). □

The proof of the above theorem can be easily modified to establish the following result.

Corollary 3.1. *Checking satisfiability of a linear aggregation constraint $\mathscr{C}$ over non-empty, finite multisets of integers is undecidable if*
1. *$\mathscr{C}$ involves the sum, count and average aggregate functions, or*
2. *$\mathscr{C}$ involves the sum, min, max and count aggregate functions.*

A natural question that can be raised is the complexity of checking for satisfiability when fewer aggregate functions occur in the aggregation constraints. The following result establishes the hardness of some simple special cases.

Theorem 3.2. *Checking satisfiability of a linear aggregation constraint over finite multisets of values drawn from any domain, involving just the count aggregate function is* NP*-complete.*

Checking satisfiability of a linear aggregation constraint over finite multisets of integers, involving either min or max or sum is NP*-complete.*

Proof. For integer linear arithmetic constraints, there is a reduction to linear aggregation constraints, where integer variable X_i is replaced by either of:

- $count(S_{i1}) - count(S_{i2})$, where S_{i1} and S_{i2} are new multiset variables ranging over finite multisets of values drawn from any domain, or
- any of the aggregation terms $min(S_i)$, $max(S_i)$ or $sum(S_i)$, where S_i is a new multiset variable ranging over non-empty, finite multisets of integers.

There is a similar reduction from linear aggregation constraints to integer linear arithmetic constraints as well. Checking for satisfiability of linear arithmetic constraints over the integers is NP-complete [6]. The result follows. □

4. An axiomatization

In this section, we present a *complete* and *minimal* set of relationships between the values of the aggregate functions on a *single* multiset. The intuition here is that the domain of aggregation constraints only allows aggregation terms on individual multisets. Interactions between different multisets is possible only via arithmetic constraints between the results of the aggregate functions on individual multisets. Consequently, relationships between the results of aggregate functions on different multisets can be inferred using techniques from the language of ordinary arithmetic constraints (see [6], for example).

Definition 4.1 (*Aggregate assignment and aggregate satisfiability*). An *aggregate assignment* maps each aggregation term of the form $F(S)$, where F is an aggregate function and S is a multiset variable, to a value.

An aggregate assignment is said to be *well-typed* if each term $F(S)$ is mapped to a value that is in the sort of the result of $F(S)$.

An aggregation constraint is said to be *satisfied* by an aggregate assignment if the aggregate assignment is well-typed and the constraint obtained by replacing each $F(S)$ by its value in the aggregate assignment is satisfiable.

An aggregation constraint is said to be *aggregate satisfiable* if there exists an aggregate assignment that satisfies the constraint.

To each multiset assignment there corresponds a unique aggregate assignment obtained by setting the aggregation terms to the values given by the application of the aggregate function to the assigned multiset. However some aggregate assignments do not correspond to any multiset assignment since they do not satisfy necessary relationships between aggregation terms.

Example 4.1. Consider the aggregation constraint $\mathscr{C}_1$

$$min(S) + max(S) = sum(S) \wedge average(S) = sum(S) - 3,$$

where S is of type $\mathscr{M}^+(\mathscr{R})$. Two aggregate assignments that satisfy this constraint are

$$\theta_1 \equiv \{min(S) \mapsto 1, max(S) \mapsto 5, sum(S) \mapsto 6, average(S) \mapsto 3\}$$

and

$$\theta_2 \equiv \{min(S) \mapsto 5, max(S) \mapsto 2, sum(S) \mapsto 7, average(S) \mapsto 4\}.$$

Hence, the aggregation constraint is aggregate satisfiable. Note that, while for θ_1 there exists a multiset $S = \{1, 5\}$ for which the aggregation terms will take the appropriate values, no such multiset corresponds to θ_2.

An aggregation constraint $\mathscr{A}(S)$ that defines the relationships between the results of aggregate functions on a single multiset S is said to be an *axiomatization* of the aggregate functions on S. We can use an axiomatization to reduce the problem of satisfiability of an aggregation constraint to a problem of aggregate satisfiability.

Intuitively, to ensure satisfiability of a given aggregation constraint, we must check the aggregate satisfiability of the conjunction of the aggregation constraint with the axiomatizations $\mathscr{A}(S_i)$ for each multiset S_i in the aggregation constraint. (The axiomatization may depend on the sort of S_i.) Checking for aggregate satisfiability amounts to treating each $F(S_i)$ as a distinct variable (of the appropriate sort), and using techniques from the domain of ordinary arithmetic constraints.

Definition 4.2 (*Soundness and completeness*). An axiomatization $\mathscr{A}(S)$ is *sound* for a given sort of multisets if every finite multiset S of the appropriate sort satisfies $\mathscr{A}(S)$.

An axiomatization $\mathscr{A}(S)$ is *complete* for a given sort of multisets and a given collection of aggregate functions if for every aggregate assignment that assigns values to the given aggregate functions on S, and that satisfies $\mathscr{A}(S)$, there exists a finite multiset S of the appropriate sort, with the corresponding aggregate values.

Theorem 4.1. *Suppose an axiomatization $\mathscr{A}(S)$ is sound and complete for a given sort of multisets and a given collection of aggregate functions. An aggregation constraint $\mathscr{C}$ using the given aggregate functions on multisets $S_1, \ldots, S_n$ of the given sort is satisfiable iff $\mathscr{C} \wedge \mathscr{A}(S_1) \wedge \cdots \wedge \mathscr{A}(S_n)$ is aggregate satisfiable.*

Proof. For the "only if" direction, if the aggregation constraint is satisfiable by an assignment to the multiset variables $S_1, \ldots, S_n$, we can assign to each aggregation term $F(S_i)$ the value defined by the assignment to S_i. The soundness of the axiomatization implies aggregate satisfiability.

For the "if" direction, suppose we have an aggregate assignment that satisfies $\mathscr{C} \wedge \mathscr{A}(S_1) \wedge \cdots \wedge \mathscr{A}(S_n)$. For each variable S_i, the completeness of the axiomatization implies that there is a multiset S_i' of the appropriate sort such that $\mathscr{A}(S_i)$ is satisfiable using S_i', and the results of the aggregate functions on S_i' are the same as in the aggregate assignment. Hence, $\mathscr{C}$ is satisfiable. □

For the SQL aggregate functions *sum*, *min*, *max*, *count* and *average*, on the sorts $\mathscr{M}^+(J)$ for several different J, there is a sound and complete axiomatization as shown by the following theorem. The only aggregate function in the above set applicable to $\mathscr{M}(J)$, for any J, is *count*. The axiomatization for this case is trivial.

Theorem 4.2. *The conjunction of the following primitive aggregation constraints* (axioms) *provides a* sound, complete *and* minimal *axiomatization of the relationships between aggregate functions min, max, sum, count and average on a finite multiset S from $\mathscr{M}^+(J)$, where J is either the reals, the rationals, the integers, the non-negative integers, or the integers divisible by any fixed number k.*
1. $min(S) \leqslant max(S)$.
2. $count(S) * min(S) + max(S) \leqslant sum(S) + min(S)$.
3. $sum(S) + max(S) \leqslant min(S) + count(S) * max(S)$.
4. $sum(S) = average(S) * count(S)$.

Proof. That the axiomatization is sound follows from the mathematical properties of the various aggregate functions. We now consider completeness.

Consider an arbitrary non-empty, finite multiset $S = \{X_1, \ldots, X_n\}$ where $n \geqslant 1$ and $X_1 \leqslant X_2 \leqslant \cdots \leqslant X_n$. By definition, we have $min(S) = X_1$, $max(S) = X_n$, $sum(S) = X_1 + \cdots + X_n$, $count(S) = n$, and $average(S) = (X_1 + \cdots + X_n)/n$. We consider several cases.

$count(S) = 1$: The axiomatization implies $min(S) = max(S) = sum(S) = average(S)$. For any choice of $min(S)$, we let $X_1 = min(S)$, and we have the required multiset.

$count(S) = 2$: The axiomatization implies that $min(S) \leqslant max(S)$, $sum(S) = min(S) + max(S)$, $sum(S) = 2 * average(S)$. Choose $X_1 = min(S)$, $X_2 = max(S)$, and we have the required multiset.

$count(S) = 3$: The axiomatization implies that $min(S) \leqslant max(S)$, $sum(S) \leqslant min(S) + 2 * max(S)$, $sum(S) \geqslant 2 * min(S) + max(S)$, $sum(S) = 3 * average(S)$. Choose $X_1 = min(S)$, $X_3 = max(S)$, $X_2 = sum(S) - min(S) - max(S)$ and we have the required multiset.

$count(S) \geqslant 4$: The axiomatization implies that $min(S) \leqslant max(S)$, $sum(S) \leqslant min(S) + (n-1) * max(S)$, $sum(S) \geqslant (n-1) * min(S) + max(S)$, $sum(S) = n * average(S)$. We choose $X_1 = min(S)$, $X_n = max(S)$. We now subdivide into several cases:
1. J is the reals or the rationals. Choose $X_2 = \cdots = X_{n-1} = (sum(S) - min(S) - max(S))/(n-2)$, and we have the required multiset.
2. J is the integers. Let $x = (sum(S) - min(S) - max(S))/(n-2)$. Choose $X_2 = \cdots = X_j = \lfloor x \rfloor$ and $X_{j+1} = \cdots = X_{n-1} = \lceil x \rceil$, where $j = 1 + (n-2)(\lceil x \rceil - x)$, and we have the required multiset.

3. If J is the non-negative integers, or the integers divisible by k for any fixed k, then a construction similar to that of the previous case applies.

This completes the proof of completeness. Minimality follows from the fact that none of the primitive constraints is entailed by the remaining primitive constraints.[5] □

Other relationships between the results of aggregate functions can be inferred using these basic relationships. For example, we can infer that $count(S) = 1$ implies that $min(S) = max(S)$. Similarly, we can infer that the constraint $max(S) < average(S)$ is unsatisfiable.

Example 4.2. Consider the aggregation constraint $\mathscr{C}_1$ from Example 4.1:

$$min(S) + max(S) = sum(S) \wedge average(S) = sum(S) - 3$$

where S is of type $\mathscr{M}^+(\mathscr{R})$. Then for the aggregation constraint $\mathscr{C}_1 \wedge \mathscr{A}(S)$

$$\theta_1 \equiv \{min(S) \mapsto 1, max(S) \mapsto 5, sum(S) \mapsto 6, average(S) \mapsto 3\}$$

is an aggregate assignment that satisfies the constraint. But

$$\theta_2 \equiv \{min(S) \mapsto 5, max(S) \mapsto 2, sum(S) \mapsto 7, average(S) \mapsto 4\}$$

does not. For example, the axiom $min(S) \leqslant max(S)$ in $\mathscr{A}(S)$ is not satisfied.

The above axiomatization contains non-linear constraints. We now show that linear constraints are not sufficient to axiomatize aggregation constraints.

Theorem 4.3. *There is no finite linear aggregation constraint over non-empty, finite multisets of reals and integers that soundly and completely axiomatizes the relationships between the aggregate functions min, max, sum and count.*

Proof. From axioms (1)–(3), the following statement Q is provable:

$$min(S) = max(S) \wedge min(S) = count(S) \rightarrow sum(S) = count(S) * count(S).$$

Given the linear aggregation constraint $min(S) = max(S) \wedge min(S) = count(S)$, the set of possible values for $sum(S)$ is $\{1, 4, 9, 16, \ldots\}$, which cannot be expressed as the solution of a linear constraint. Thus, Q cannot be entailed by a finite linear aggregation constraint.

For any sound finite linear axiomatization $\mathscr{A}$, Q is not entailed by $\mathscr{A}$. It follows that it is possible to choose values of $min(S)$, $max(S)$, $sum(S)$, and $count(S)$ such that $min(S) = max(S)$, $min(S) = count(S)$ and $sum(S) \neq count(S) * count(S)$, but for which these values satisfy the axioms of $\mathscr{A}$. Since no such multiset S exists, $\mathscr{A}$ is not complete. □

[5] Axiom (1) is implied by axioms (2) and (3) only for the case that $count(S) \geqslant 3$.

5. Satisfiable special cases

In this section, we present some special cases of aggregation constraints where checking for satisfiability is tractable, i.e., satisfiability can be checked in time polynomial in the size of the representation of the constraints.

5.1. Directly using the axiomatization

We briefly describe two cases where the axiomatization presented in Section 4 can be used to obtain polynomial-time algorithms for checking satisfiability. The intuition here is that in each of the two cases the axiomatization of the relationships between the results of the various aggregate functions can be simplified to a conjunction of linear arithmetic constraints. These simplified axioms can then be conjoined with the given aggregation constraints, each distinct aggregation term can be replaced by a distinct arithmetic variable (of the appropriate sort) and satisfiability can be determined using techniques from existing constraint domains.

The first case is when the aggregation constraint involves only *min* and *max*. In this case, only the axiom $min(S) \leqslant max(S)$ needs to be conjoined for each multiset variable S involved. If the original aggregation constraint is linear and the multiset elements are drawn from the reals, the transformed arithmetic constraint is also linear over the reals; satisfiability can now be checked in time polynomial in the size of the aggregation constraint, using any of the standard techniques (see [6], for example) for solving linear arithmetic constraints over the reals.

Example 5.1. Consider the aggregation constraint $\mathscr{C}_2$

$$max(S_1) + max(S_2) = 2 \wedge min(S_1) + 2 * min(S_2) \geqslant 5,$$

where S_1 and S_2 are of type $\mathscr{M}^+(\mathscr{R})$. Then this can be checked for satisfiability by simply finding an aggregate assignment that aggregate satisfies the constraint

$$\mathscr{C}_2 \wedge min(S_1) \leqslant max(S_1) \wedge min(S_2) \leqslant max(S_2).$$

This can be achieved by replacing each aggregation term by a real variable, e.g., replacing $min(S_i)$ by x_i and $max(S_i)$ by y_i we obtain

$$y_1 + y_2 = 2 \wedge x_1 + 2 * x_2 \geqslant 5 \wedge x_1 \leqslant y_1 \wedge x_2 \leqslant y_2.$$

Then a solution for the resulting linear real constraint is found, e.g., $\{x_1 \mapsto -1, y_1 \mapsto -1, x_2 \mapsto 3, y_2 \mapsto 3\}$. And finally this solution is mapped back to form an aggregate assignment. For example,

$$\{min(S_1) \mapsto -1, max(S_1) \mapsto -1, min(S_2) \mapsto 3, max(S_2) \mapsto 3\}$$

is such an aggregate assignment.

Any such aggregate assignment can be extended to a multiset assignment, by allowing S_i to be the multiset $\{min(S_i), max(S_i)\}$. Hence, $\{S_1 \mapsto \{-1, -1\}, S_2 \mapsto \{3, 3\}\}$ is a multiset assignment satisfying the aggregation constraint $\mathscr{C}_2$. Similarly we can show that the aggregation constraint $\mathscr{C}_2 \wedge min(S_1) \geqslant 0 \wedge min(S_2) \geqslant 0$ is unsatisfiable, since there is no aggregate assignment for

$$\mathscr{C}_2 \wedge min(S_1) \geqslant 0 \wedge min(S_2) \geqslant 0 \wedge min(S_1) \leqslant max(S_1) \wedge min(S_2) \leqslant max(S_2). \qquad \square$$

The second case where we can obtain polynomial-time algorithms is when the linear aggregation constraint explicitly specifies the cardinality of each multiset, i.e., for each multiset variable S_i, we know that $count(S_i) = k_i$, where k_i is a constant. In this case, each of the non-linear axioms can be simplified to a linear primitive constraint; checking for satisfiability again takes time polynomial in the size of the aggregation constraints if the multiset elements are drawn from the reals.

Example 5.2. Consider the aggregation constraint $\mathscr{C}_3$

$$count(S) = 2 \wedge sum(S) = average(S) \wedge min(S) \geqslant 0,$$

where S is of type $\mathscr{M}^+(\mathscr{R})$. The axiomatization $\mathscr{A}(S)$ for multiset S simplifies to the following *linear* aggregation constraint given $count(S) = 2$:

$$\begin{aligned} &min(S) \leqslant max(S) \\ &\quad \wedge min(S) + max(S) \leqslant sum(S) \\ &\quad \wedge sum(S) \leqslant min(S) + max(S) \\ &\quad \wedge sum(S) = 2 * average(S). \end{aligned}$$

The conjunction of $\mathscr{C}_3$ with the above aggregation constraint has a single aggregate assignment that aggregate satisfies it

$$\{min(S) \mapsto 0, max(S) \mapsto 0, sum(S) \mapsto 0, average(S) \mapsto 0, count(S) \mapsto 2\}$$

We can deduce this using standard methods for linear real arithmetic. Then by Theorem 4.2 there must exist at least one multiset assignment that satisfies aggregation constraint $\mathscr{C}_3$, in this case $\{S \mapsto \{0, 0\}\}$.

5.2. Linear separable aggregation constraints

In this section, we examine a very useful class of aggregation constraints, and present a polynomial-time algorithm to check for satisfiability of constraints in the class. Our technique operates directly on the aggregation constraints, rather than on their reduction to arithmetic constraints. The reduced form of this class includes mixed integer/real constraints, and is non-linear; it is not clear how to operate directly on the reduced form and attain the same complexity as our algorithm. We specify the class of constraints in terms of the factors, described in Section 3, that affect the complexity of checking for satisfiability. We require the following:

1. The domain of multiset elements is $\mathscr{R}$, the reals.

2. The constraints are linear and specified using $\leqslant, <, =, >$ and $\geqslant$.
3. The constraints are multiset-variable-separable and aggregate-function-separable.

The above restrictions ensure that we can simplify the given conjunction of aggregation constraints to *range constraints* on each aggregation term. We refer to this class of aggregation constraints as $\mathscr{L}\mathscr{S}$-aggregation-constraints.[6]

Most aggregation constraints occurring in queries are multiset-variable-separable. Only when we consider constraint propagation or fold/unfold transformations are we likely to obtain non-multiset-variable-separable aggregation constraints. The further restrictions for $\mathscr{L}\mathscr{S}$-aggregation-constraints are not onerous; Example 1.1 uses such constraints.

The general algorithm along with a proof of correctness is presented in the appendix. Here, to present the main ideas underlying the general algorithm, we describe the algorithm for the simpler case when the only aggregate functions present are *min, max*, *sum* and *count*, i.e., there are no aggregation constraints involving *average*.

5.2.1. Multiset ranges: No average

The heart of our algorithm is a function Multiset_Ranges that takes four finite and closed ranges, $[m_l, m_h]$, $[M_l, M_h]$, $[s_l, s_h]$, and an integer range $[k_l, k_h]$, and answers the following question:

Do there exist $k > 0$ numbers, k between k_l and k_h, such that the minimum of the k numbers is between m_l and m_h, the maximum of the k numbers is between M_l and M_h, and the sum of the k numbers is between s_l and s_h?

When $a > b$, the closed range $[a, b]$ is empty. We use operations such as "overlaps" on pairs of ranges; these can be defined easily in terms of the primitive comparison operations between endpoints of the two ranges. Note that the empty range does not overlap with any range.

```
function Multiset_Ranges $(m_l, m_h, M_l, M_h, s_l, s_h, k_l, k_h)$ {
/* We assume finite and closed ranges. */
(1) /* Tighten $min, max$ and $count$ bounds. */
    (a) if $(M_l < m_l)$ then $M_l = m_l$.
    (b) if $(m_h > M_h)$ then $m_h = M_h$.
    (c) if $(k_l < 1)$ then $k_l = 1$.
(2) /* Obviously unsatisfiable cases. */
    (a) if $(k_l > k_h$ or $m_l > m_h$ or $M_l > M_h$ or $s_l > s_h)$ then
          /* infeasible ranges */
          return 0.
/* Case A: Elements can be negative, positive, or 0. */
```

[6] $\mathscr{L}\mathscr{S}$ = linear, separable.

```
    (3) if ($[m_l,M_h]$ overlaps $[0,0]$) then
        (a) if ($[s_l,s_h]$ does not overlap $[(k_h-1)*m_l+M_l,$
              $m_h+(k_h-1)*M_h]$) then return 0.
        (b) else return 1.
    /* Case B: All elements are negative. Switch everything. */
    (4) if ($M_h<0$) then
        (a) $[t1,t2]=[-M_h,-M_l]$; $[M_l,M_h]=[-m_h,-m_l]$; $[m_l,m_h]=[t1,t2]$.
        (b) $t=-s_l$; $s_l=-s_h$; $s_h=t$.
        /* Continue with Case C */
    /* Case C: All elements are positive. */
    (5) /* $m_l>0$. */
        (a) if ($[s_l,s_h]$ does not overlap $[(k_l-1)*m_l+M_l,$
              $m_h+(k_h-1)*M_h]$) then return 0.
              /* sum is too low or too high. */
        (b) define integers $k_1$ and $k_2$ by $s_l=m_h+(k_1-1)*M_h-k_2$,
              $0\leqslant k_2<M_h$.
              /* Multiset cardinality must be $\geqslant k_1$, for $sum\geqslant s_l$. */
        (c) define integers $k_3$ and $k_4$ by $s_h=(k_3-1)*m_l+M_l+k_4$,
              $0\leqslant k_4<m_l$.
              /* Multiset cardinality must be $\leqslant k_3$, for sum $\leqslant s_h$. */
        (d) if ($[k_1,k_3]$ overlaps $[k_l,k_h]$) then
              return 1. /* any $k$ in the intersection is a witness. */
        (e) else return 0.
}
```

Theorem 5.1. *Function* Multiset_Ranges *returns* 1 *iff there exist* $k>0$ (*real or integer*) *numbers*, $k_l\leqslant k\leqslant k_h$, *such that the minimum of the k numbers is in* $[m_l,m_h]$, *the maximum of the k numbers is in* $[M_l,M_h]$, *and the sum of the k numbers is in* $[s_l,s_h]$.

Further, Multiset_Ranges *has polynomial time complexity in the size of the representation of the input.*

Proof. We prove the first part of the theorem by showing that the algorithm returns 1 if and only if the given constraints along with the four axioms of Theorem 4.2 are satisfiable.

Steps (1a) and (1b) generate all constraints on *min* and *max* that can be inferred from the given range constraints on *min* and *max* and the axioms. If Step (2) returns 0, the resultant set of constraints is clearly unsatisfiable. Else, the conjunction of the given range constraints on *min*, *max* and *count* along with all the axioms is satisfiable. We now have to consider only the constraints on *sum*.

All elements in the multiset have to lie in the range $[m_l,M_h]$; the minimum and maximum elements are additionally constrained to lie in the ranges $[m_l,m_h]$ and $[M_l,M_h]$, respectively. Axioms (2) and (3) are satisfied if and only if the sum is in the union of the ranges:

$$\bigcup_{i=k_l}^{k_h} [(i-1)*m_l + M_l, m_h + (i-1)*M_h].$$

In general, this union of ranges need not be convex; there may be gaps.

Thus, the conjunction of the given constraints and axioms (1)–(4) is satisfiable if and only if there is an i such that the given range on sum $[s_l, s_h]$ overlaps with the range: $[(i-1)*m_l + M_l, m_h + (i-1)*M_h]$. The algorithm for testing the above has three cases, based on the location of the $[m_l, M_h]$ range with respect to zero.

The first case is when the $[m_l, M_h]$ range includes zero; in this case, the union of the ranges from which the sum can take values is convex, and is given by

$$[(k_h-1)*m_l + M_l, m_h + (k_h-1)*M_h].$$

Step (3) checks that $[s_l, s_h]$ overlaps with this range.

The second case is when the $[m_l, M_h]$ range includes only negative numbers, and the third case is when the $[m_l, M_h]$ range includes only positive numbers. These two cases are symmetric, and we transform the second case into the third case in Step (4), and consider only the third case in detail.

In the third case, the sum lies within the range $[(k_l-1)*m_l+M_l, m_h+(k_h-1)*M_h]$, but not all values in this range are feasible — there may be gaps. The conjunction of constraints is unsatisfiable if and only if the $[s_l, s_h]$ range lies outside $[(k_l-1)*m_l + M_l, m_h + (k_h-1)*M_h]$, or entirely within one of the gaps. Step (5a) checks for the first possibility, and Steps (5b)–(5e) check for the second possibility. The number k_1 gives the smallest cardinality that the multiset can have subject to the constraints on *min* and *max*, such that its *sum* is $\geqslant s_l$. Similarly, the number k_3 gives the largest cardinality that the multiset can have subject to the constraints on *min* and *max*, such that its *sum* is $\leqslant s_h$.

Clearly, if $[k_1, k_3]$ is infeasible, and hence by definition $[k_1, k_3]$ does not overlap $[k_l, k_h]$, then the constraints are unsatisfiable. If $[k_1, k_3]$ is feasible, let j be any integer in $[k_1, k_3]$. The possible values of *sum* for this j are all values in $[(j-1)*m_l+M_l, m_h+(j-1)*M_h]$. Now by the definition of k_1 the range for $j=k_1$ is not entirely to the left of $[s_l, s_h]$, and the range for $j=k_3$ is not entirely to the right of $[s_l, s_h]$. But since $k_1 \leqslant k_3$, both these ranges must overlap $[s_l, s_h]$. It is then easy to show that for all j in $[k_1, k_3]$ the range for j overlaps $[s_l, s_h]$. Since $[k_1, k_3]$ overlaps $[k_l, k_h]$, there is a j element multiset that satisfies all the constraints. This concludes the proof of the first part of the theorem.

The proof of the second part of the theorem is straightforward because the number of steps in Multiset_Ranges is bounded above by a constant, and each step is polynomial in the size of representation of the input. □

Checking for satisfiability of a conjunction of $\mathscr{LS}$-aggregation constraints proceeds as follows. Since the aggregation constraints are multiset-variable-separable, the primitive aggregation constraints can be partitioned based on the multiset variable, and the conjunction of aggregation constraints in each partition can be solved separately. The overall conjunction is satisfiable iff the conjunction in each partition is separately satisfiable.

Though $\mathscr{LS}$-aggregation-constraints are restricted, they are strong enough to infer useful new aggregation constraint information. They can be used to infer some information about an *arbitrary* aggregation constraint $\mathscr{C}$ by determining an $\mathscr{LS}$-aggregation-constraint H that is implied by $\mathscr{C}$; any aggregation constraints implied by H are then also implied by $\mathscr{C}$.

5.2.2. *Dealing with average in Multiset Ranges*

In the appendix we describe Gen_Multiset_Ranges, which is a generalization of the function Multiset_Ranges, described in the previous section. It takes a finite and closed range $[a_l, a_h]$ for *average*, in addition to the ranges for *min, max, sum* and *count*, and determines in polynomial-time if there is a non-empty, finite multiset of real numbers that satisfies all the aggregation constraints. Gen_Multiset_Ranges is based on three key observations, presented here.

- Requiring the minimum value of a multiset to be in the (consistent) range $[m_l, m_h]$, and the maximum value of the multiset to be in the (consistent) range $[M_l, M_h]$, allows us to *infer* that the sum of the values of an i element multiset must be in the range:

  $$[(i-1) * m_l + M_l, m_h + (i-1) * M_h].$$

  Given that the average value of a multiset is in the (consistent) range $[a_l, a_h]$, we can infer that the sum of the values of an i element multiset must be in the range:

  $$[i * a_l, i * a_h].$$

  The *first* key observation used in Gen_Multiset_Ranges combines these two ideas as follows. Given range constraints on the minimum value, on the maximum value, and on the average value of a multiset, the sum of the values of an i element multiset must be in the *intersection* of the inferred ranges for sum, based on min and max, on the one hand, and based on average, on the other. When the count of the multiset is known to be in the range $[k_l, k_h]$, we can infer that the sum must be in the following union of ranges:

  $$\bigcup_{i=k_l}^{k_h} ([(i-1) * m_l + M_l, m_h + (i-1) * M_h] \cap [i * a_l, i * a_h]).$$

- The *second* key observation used in Gen_Multiset_Ranges is as follows: If i_1 is the smallest integer $i \geqslant k_l$ for which the ranges $[(i-1) * m_l + M_l, m_h + (i-1) * M_h]$ and $[i * a_l, i * a_h]$ overlap, then for all $i \geqslant i_1$, the two ranges overlap.

  This observation can be inferred from the following facts: (a) the maximum value of a multiset can be no smaller than the minimum value (i.e., $M_l \geqslant m_l$ and $M_h \geqslant m_h$), (b) the average value of a multiset can be no smaller than the minimum value (i.e., $a_l \geqslant m_l$), and no larger than the maximum value of the multiset (i.e., $a_h \leqslant M_h$).

- The *third* key observation, repeatedly used in Gen_Multiset_Ranges, involves two properties of ranges: (a) given three ranges such that every pair from this collection overlap, there exists at least one point that is common to all three ranges, and (b) given two ranges that overlap, a third range does not overlap with the intersection of the two ranges if and only if the third range does not overlap with at least one of the two ranges.

  Thus, in checking that the given range $[s_l, s_h]$ on the sum of the values of a multiset overlaps with the inferred union of ranges for sum (see first observation above), it suffices to check that there exists at least one i in $[i_1, k_h]$ such that $[s_l, s_h]$ overlaps with $[(i-1)*m_l + M_l, m_h + (i-1)*M_h]$, as well as with $[i*a_l, i*a_h]$. Each of these checks can be independently done using the technique described in Multiset_Ranges.

6. Using constraints on multiset elements

By using the constraints that are known on the elements of a multiset, we can infer constraints on the results of aggregate functions on the multiset. The following example illustrates this:

Example 6.1 (*Multiset element constraints*). Consider again the view V from Example 1.1:

```
CREATE   VIEW V(Dept, Min-Sal, Max-Sal, Sum-Sal, Count-Emp) AS
SELECT   Dept, MIN(Salary), MAX(Salary), SUM(Salary), COUNT(Salary)
FROM     E
GROUP BY Dept
HAVING   COUNT(Salary) ≤ 10 AND MIN(Salary) > 0
         AND MAX(Salary) ≤ 10 000
```

In addition to the constraints on the results of the aggregate functions present in the HAVING clause, constraints may be known on tuples of the employee relation E; for example, each employee may be known to have a salary between 1000 and 5000. If the employee relation is a database relation, these constraints may be specified as integrity constraints on the database. If the employee relation is a derived view relation, these constraints may be computed using the integrity constraints on the database relations and the definition of the employee relation (see [7], for example).

Constraints on the tuples of the employee relation can be used to infer constraints on the results of the aggregate functions (and hence on the tuples of V). For example, if each employee is known to have a salary between 1000 and 5000, then the minimum salary and the maximum salary of each department in the view can be inferred to be between 1000 and 5000.

Consider the query

```
SELECT   *
FROM     V
WHERE    Sum-Sal > 50000
```

Given the constraints in the WHERE clause of the above query and in the view definition, it is possible for this query to have answers. However, if we take the constraints on the salaries of each employee into account, we can determine that $min(S) \geqslant 1000 \wedge max(S) \leqslant 5000$, where S is the multiset of salaries of employees in some department. In conjunction with the aggregation constraint $count(S) \leqslant 10$, it is now possible to determine that the query can have no answers.

Let each element E of multiset S satisfy constraint $\mathscr{C}(E)$, i.e., $\forall E \in S, \mathscr{C}(E)$. The following result provides a technique to infer constraints that hold on the results of aggregate functions on multiset S.

Theorem 6.1. *Let $\mathscr{C}(E)$ be an arithmetic constraint (in disjunctive normal form, for simplicity). Consider a non-empty, finite multiset S of reals. Let $\mathscr{A}(S)$ be the axiomatization relating the results of aggregate functions min, max, sum, count and average on multiset S. Suppose $\forall E \in S, \mathscr{C}(E)$. Then, the following constraint holds*:

$$\mathscr{C}(min(S)) \wedge \mathscr{C}(max(S)) \wedge count(S) > 0 \wedge \mathscr{A}(S).$$

Proof. We show soundness by showing the soundness of each conjunct in $\mathscr{C}(min(S)) \wedge \mathscr{C}(max(S)) \wedge count(S) > 0 \wedge \mathscr{A}(S)$. Since $min(S)$ and $max(S)$ are both elements of multiset S, they must satisfy the constraint $\mathscr{C}$, by assumption. The constraint $count(S) > 0$ is equivalent to the assumption that the multiset S is non-empty. The soundness of $\mathscr{A}(S)$ follows from Theorem 4.2. □

Although the constraint $\mathscr{C}(min(S)) \wedge \mathscr{C}(max(S)) \wedge count(S) > 0 \wedge \mathscr{A}(S)$ is sound, it may not, in general, be the tightest possible constraint that holds on the results of the aggregate functions, i.e., the above constraint may be *incomplete*. The following examples present several classes of constraints for which the above constraint is incomplete. Subsequently, we describe a constraint class for which the above constraint is indeed complete.

Example 6.2 (*Incompleteness with disjunctive linear constraints*). Consider a non-empty, finite multiset S of reals. Let $\mathscr{C}(E) \equiv E = 0 \vee E = 2$ be the constraint known to be satisfied by each element E of the multiset S. It is obvious that $sum(S)$ is non-negative and even. (Evenness can be expressed using aggregation constraints by asserting that $sum(S) = 2 * count(S1)$,[7] where $S1$ is a new multiset variable.) However,

[7] Note that $\mathscr{C}(E) \equiv E = 2 * count(S1)$, where $S1$ is a new multiset variable, forces each element of the multiset S to be the same non-negative even integer, rather than S being any multiset of non-negative even integers.

this cannot be inferred using the constraint in Theorem 6.1. Intuitively, this is because the constraint $\mathscr{C}(min(S)) \wedge \mathscr{C}(max(S))$ does not imply that each element of the multiset is either 0 or 2, which is the case in this example.

Example 6.3 (*Incompleteness with non-linear constraints*). Consider a non-empty, finite multiset S of reals. Let $\mathscr{C}(E) \equiv E*E=2*E$ be the constraint known to be satisfied by each element E of the multiset S. Since $E*E=2*E$ is equivalent to $E=0 \vee E=2$, incompleteness follows from the previous example.

Theorem 6.2. *Let $\mathscr{C}(E)$ be a range constraint on E. Consider a non-empty, finite multiset S of reals. Let $\mathscr{A}(S)$ be the axiomatization relating the results of aggregate functions min, max, sum, count and average for multiset S. Suppose $\forall E \in S, \mathscr{C}(E)$. Then,*

$$\mathscr{C}(min(S)) \wedge \mathscr{C}(max(S)) \wedge count(S) > 0 \wedge \mathscr{A}(S)$$

is a complete *aggregation constraint satisfied by the results of the aggregate functions min, max, sum, count and average on multiset S.*

Proof. Consider the aggregation constraint

$$\mathscr{C}(min(S)) \wedge \mathscr{C}(max(S)) \wedge count(S) > 0 \wedge \mathscr{A}(S).$$

Since $\mathscr{C}$ is a range constraint, the constraint $\mathscr{C}(min(S)) \wedge \mathscr{C}(max(S))$ implies that each element of the multiset lies in the range given by $\mathscr{C}$. Further, the constraint $count(S) > 0$ implies that the multiset is non-empty. □

Note that the constraint $\mathscr{C}(E)$ allowed on the multiset elements is quite restricted. For example, constraints of the form $\forall E1, E2 \in S,\ E1 \leqslant 2 + E2$, i.e., constraints that relate different elements of the multiset, are not allowed. Constraints of the form, $\forall E \in S,\ E = count(S)$ are not allowed either since the constraint involves an aggregate function. Existential quantification on the set elements, such as $\exists E \in S,\ E=2$ is not allowed either.

Although the class of constraints allowed on multiset elements is small, it is of significant practical value in applications such as database query optimization. Database queries typically specify only simple range constraints, as is the case in Example 6.1.

7. Inferring constraints on multiset elements

Given constraints on an SQL view defined using aggregation, it is useful to be able to infer constraints on the tuples of the database relations used to define the view.

Example 7.1 (*Inferring constraints on multiset elements*). Let P be a database relation with attributes X and Y. Consider the following view:

```
CREATE   VIEW V (X, Min) AS
SELECT   X, MIN(Y)
FROM     P
GROUP BY X
```

Suppose we are given the following (integrity) constraint on the view V:

$$\forall X \, \forall M, \ V(X,M) \rightarrow (M > 5).$$

Then we can infer the following integrity constraint on the relation P:

$$\forall X \, \forall Y, \ P(X,Y) \rightarrow (Y > 5).$$

The following result is straightforward.

Theorem 7.1. *Consider a conjunction of aggregation constraints $\mathscr{C}(S)$ on a single multiset variable S. Let $\mathscr{A}(S)$ be the axioms on a multiset, as in Theorem 4.2. Then for any multiset assignment θ satisfying $\mathscr{C}(S)$, for element $e \in \theta(S)$ the constraint*

$$\mathscr{C}(S) \wedge \mathscr{A}(S) \wedge e \geqslant min(S) \wedge e \leqslant max(S)$$

is aggregate satisfiable.

Let $\mathscr{E}(E)$ denote the following constraint on E

$$\exists S \ \mathscr{C}(S) \wedge \mathscr{A}(S) \wedge E \geqslant min(S) \wedge E \leqslant max(S)$$

$\mathscr{E}(E)$ is always a constraint that defines a (possibly unbounded) range of values for E.

Example 7.2. Consider the aggregation constraint $\mathscr{C}_4$

$$sum(S) = 5 \wedge min(S) \geqslant 1,$$

where S is of type $\mathscr{M}^+(\mathscr{R})$. Then $\mathscr{E}(E)$ is

$$\exists S \ sum(S) = 5 \wedge min(S) \geqslant 1 \wedge \mathscr{A}(S) \wedge E \geqslant min(S) \wedge E \leqslant max(S)$$

or equivalently $E \geqslant 1 \wedge E \leqslant 4$. Thus, any multiset assignment satisfying $\mathscr{C}_4$, for example $\{S \mapsto \{1,4\}\}$ or $\{S \mapsto \{1,1,1,1,1\}\}$, must assign elements to S in the range 1 to 4. Similarly given the aggregation constraint $\mathscr{C}_5$

$$sum(S) \geqslant 10 \wedge count(S) = 1,$$

where S is of type $\mathscr{M}^+(\mathscr{R})$, then $\mathscr{E}(E)$ is equivalent to the (unbounded in one direction) constraint $E \geqslant 10$.

We conjecture that $\mathscr{E}(E)$ is the tightest constraint in the class of conjunctive linear arithmetic constraints in E that hold on elements of the multiset satisfying $\mathscr{C}(S)$.

The conjecture does not hold if either disjunction or non-linearity is allowed, as the following example demonstrates.

Example 7.3 (*Incompleteness with disjunctions or non-linearity*). Consider the following conjunction $\mathscr{C}$ of aggregation constraints:

$$sum(S) = 13 \wedge count(S) = 4 \wedge min(S) = 1 \wedge max(S) = 10,$$

where S is of type $\mathscr{M}^+(\mathscr{R})$. In this case $\mathscr{E}(E)$ is $E \geqslant 1 \wedge E \leqslant 10$. According to the above conjecture, this should be the tightest conjunction of constraints linear in E that holds for all elements of the multiset S. However, the only multiset S that satisfies $\mathscr{C}$ is $\{1, 1, 1, 10\}$, for which the stronger disjunctive constraint $E = 1 \vee E = 10$ holds for all $E \in S$. Note that this disjunctive constraint is equivalent to the non-linear conjunctive constraint $E * E + 10 = 11 * E$.

8. Query constraints and relevance

Queries can have constraints associated with them. Intuitively, only answers that satisfy these constraints are "relevant" to the query. Such constraints are referred to as *query constraints*, and are used extensively in query optimization (e.g., [7–9, 2]).

Query constraints in the presence of aggregate functions have been considered in [9, 2]. However, they consider special cases. Sudarshan and Ramakrishnan [9] essentially consider dynamic order constraints of the form $X \leqslant f_1$ and $X \geqslant f_2$, where f_1 is the "current" value of $min(S)$ and f_2 is the "current" value of $max(S)$, and S is a multiset that is incrementally computed during program evaluation. Levy et al. [2] only consider constraints of the form $max(S) \geqslant c$ and $min(S) \leqslant c$, where c is a constant.

The following examples illustrate the benefits of inferring query constraints on multiset elements, given query constraints on the results of aggregate functions on the multiset, in cases that are not handled by earlier techniques.

Example 8.1 (*Inferring query constraints*). Let `P` be a relation with attributes `X` and `Y`. Consider the following view:

```
CREATE   VIEW V (X, Max) AS
SELECT   X, MAX(Y)
FROM     P
GROUP BY X
```

and the following query:

```
SELECT   X, Max
FROM     V
WHERE    Max≥X
```

Consider a tuple (x, y) of P satisfying $y<x$. Two cases need to be considered. First, when y is not the maximum value in the group for x. In this case, the tuple (x, y) is irrelevant for computing V. (Note that an (x, y) tuple of P, where y is not the maximum value in the group for x, is irrelevant whether or not $y<x$.) Next, consider the case when y is the maximum value in the group for x. Then, the tuple (x, y) is in the extension of V; however, this tuple does not satisfy the given query constraint. In either case, if $y<x$, the tuple (x, y) of P is irrelevant to the given query. Hence, the query constraint $\mathrm{P}(X, Y): Y \geqslant X$ can be inferred on the relation P; this can be used to optimize query evaluation.

A similar observation holds for the query

```
SELECT   X, Max
FROM     V
WHERE    Max = X
```

Since $\mathrm{Max} = \mathrm{X} \rightarrow \mathrm{Max} \geqslant \mathrm{X}$, the previous arguments can be used to infer the query constraint $\mathrm{P}(X, Y): Y \geqslant X$ on the relation P.

The following theorem indicates how aggregation constraints can be used in query optimization.

Theorem 8.1. *Let view* V *be defined as follows*:

```
CREATE   VIEW V (X1, ..., Xn, Max) AS
SELECT   X1, ..., Xn, MAX(Y)
FROM     P
GROUP BY X1, ..., Xn
```

where $\mathrm{X}_1, \ldots, \mathrm{X}_n$ *and* Y *are distinct attributes of P. Let* $\bar{X}$ *denote the attributes* $\mathrm{X}_1, \ldots, \mathrm{X}_n$, *and let* $\bar{Z}$ *denote the attributes of* P *other than* $\bar{X}$ *and* Y. *Suppose we are given a query on view* V *with query constraint* $\mathscr{C}(\bar{X}, \mathrm{Max})$ *on the tuples in* V. *Let* $f(\bar{X}) \leqslant \mathrm{Max}$ *be a constraint that is implied by the constraint* $\mathscr{C}(\bar{X}, \mathrm{Max})$. *Then the answer to the query does not change if the definition of* V *is replaced with*

```
CREATE   VIEW V (X1, ..., Xn, Max) AS
SELECT   X1, ..., Xn, MAX(Y)
FROM     P
WHERE    f(X̄) ≤ Y
GROUP BY X1, ..., Xn
```

Proof. Consider any tuple $(\bar{x}, \bar{z}, y)$ of P that does not satisfy $f(\bar{x}) \leqslant y$. Two cases need to be considered. First, when y is not the maximum value in the group for $\bar{x}$. In this case, the tuple $(\bar{x}, \bar{z}, y)$ does not contribute to any tuple of V. Next, consider the case when y is the maximum value in the group for $\bar{x}$. Then, the tuple $(\bar{x}, y)$ is in the extension of V; however, this tuple does not satisfy the given query constraint on V. In either case, if $f(\bar{x}) \leqslant y$ is not satisfied, the tuple $(\bar{x}, \bar{z}, y)$ of P is irrelevant to the given query. □

A consequence of this theorem is that the constraint $f(\bar{X}) \leqslant \texttt{Y}$ can be pushed into the evaluation of P. If P is itself a view, or if $f(\bar{X}) \leqslant \texttt{Y}$ allows a more efficient indexed lookup of P, then we can potentially improve the performance of the query. A result similar to Theorem 8.1, but with the aggregate function *min* used in the rule instead of *max*, and a constraint of the form $f(\bar{X}) \geqslant \texttt{Min}$ instead of $f(\bar{X}) \leqslant \texttt{Max}$, also holds.

We conjecture that the query constraint derived by the above theorem is the strongest conjunctive query constraint that is linear in Y that can be derived on relation P.

9. Conclusions and future work

We have presented a new and extremely useful class of constraints, *aggregation constraints*, and studied the problem of checking for satisfiability of conjunctions of primitive aggregation constraints. There are many interesting directions to pursue. An important direction of active research is to significantly extend the class of aggregation constraints for which satisfiability can be efficiently checked. We believe that our algorithm works on a larger class of aggregation constraints than presented here – for instance, we believe that our algorithm will work correctly even if we relax the conditions to not require *min* and *max* to be separated; characterizing this class will be very useful.

Combining aggregation constraints with multiset constraints that give additional information about the multisets (using functions and predicates such as $\cup, \in, \subseteq$, etc.) will be very important practically.

Another important direction is to examine how this research can be used to improve query optimization and integrity constraint verification in database query languages such as SQL. Sudarshan and Ramakrishnan [9] and Levy et al. [2] consider how to use simple aggregation conditions for query optimization; it would be interesting to see how their work can be generalized. It would also be interesting to see how to use aggregation constraints in conjunction with Stuckey and Sudarshan's technique [8] for compilation of query constraints.

We believe that we have identified an important area of research, namely aggregation constraints, and have laid the foundations for further research.

Appendix. Multiset ranges: *min*, *max*, *sum*, *average* and *count*

The function Gen_Multiset_Ranges, below, is a generalization of the function in Section 5.2.1. It takes five finite and closed ranges, $[m_l, m_h]$, $[M_l, M_h]$, $[s_l, s_h]$, $[a_l, a_h]$ and an integer range $[k_l, k_h]$, and answers the following question:

Do there exist $k > 0$ numbers, k between k_l and k_h, such that the minimum of the k numbers is between m_l and m_h, the maximum of the k numbers is between M_l and M_h, the sum of the k numbers is between s_l and s_h, and the average of the k numbers is between a_l and a_h?

```
function Gen_Multiset_Ranges $(m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h, k_l, k_h)$ {
/* we assume finite and closed ranges */
(1) /* Tighten min, max, average and count bounds. */
    (a) Tighten_MMA_Bounds $(m_l, m_h, M_l, M_h, a_l, a_h)$.
    (b) Tighten_Count_Bounds $(m_l, m_h, M_l, M_h, a_l, a_h, k_l, k_h)$.
(2) if (Obviously_Unsatisfiable $(m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h, k_l, k_h)$) then
    return 0.
/* For each $k$ in $[k_l, k_h]$, we now have that $[k * a_l, k * a_h]$ overlaps
        $[(k-1) * m_l + M_l, m_h + (k-1) * M_h]$. */
/* Case A: Based on min and max elements can be $< 0$, $= 0$
            or $> 0$. */
(3) if ($[m_l, M_h]$ overlaps $[0,0]$) then
    (a) if ($[s_l, s_h]$ does not overlap $[(k_h - 1) * m_l + M_l, m_h + (k_h - 1)$
        $* M_h]$) then return 0.
    (b) if ($[a_l, a_h]$ overlaps $[0,0]$) then
        (i) if ($[s_l, s_h]$ does not overlap $[k_h * a_l, k_h * a_h]$) then
            return 0.
        (ii) else return 1.
    (c) if $(a_h < 0)$ then
        (i) Switch_Signs $(m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h)$.
        /* Falls through to the next case. */
    (d) /* else $a_l > 0$ */
        (i) if ($[s_l, s_h]$ does not overlap $[k_l * a_l, k_h * a_h]$) then
            return 0.
        (ii) else if (In_Sum_Gap_NP $(m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h, k_l, k_h)$)
            then return 0.
        (iii) else return 1.
/* Case B: All elements are negative. Switch everything. */
(4) if $(M_h < 0)$ then
    (a) Switch_Signs $(m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h)$.
    /* Falls through to the next case. */
/* Case C: All elements are positive. */
(5) /* else $m_l > 0$ */
    /* Range for sum outside bounds dictated by min and
       max. */
    (a) if ($[s_l, s_h]$ does not overlap $[(k_l - 1) * m_l + M_l, m_h + (k_h - 1)$
        $* M_h]$) then return 0.
    /* Range for sum outside bounds dictated by average. */
    (b) else if ($[s_l, s_h]$ does not overlap $[k_l * a_l, k_h * a_h]$) then
          return 0.
    (c) else if (In_Sum_Gap_PP ($m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h, k_l, k_h$)) then
          return 0.
    (d) else return 1.
}

Tighten_MMA_Bounds ($m_l, m_h, M_l, M_h, a_l, a_h$) {
/* Tighten bounds for max based on $min(S) \leqslant max(S)$. */
/* Tighten bounds for min based on $min(S) \leqslant max(S)$. */
(2) if ($m_h > M_h$) then $m_h = M_h$.
/* Tighten bounds for average based on $min(S) \leqslant average(S)$. */
(3) if ($a_l < m_l$) then $a_l = m_l$.
/* Tighten bounds for average based on $average(S) \leqslant max(S)$. */
(4) if ($a_h > M_h$) then $a_h = M_h$.
}

Tighten_Count_Bounds ($m_l, m_h, M_l, M_h, a_l, a_h, k_l, k_h$) {
/* Tighten lower bound for count using min, max and average
ranges. */
(1) if ($k_l < 1$) then $k_l = 1$.
(2) if ($a_h < ((k_l - 1) * m_l + M_l)/k_l$ and $M_l \neq m_l$) then
    /* Known range for average to the left of smallest inferred
       range. */
    (a) $k_l = \lceil (M_l - m_l)/(a_h - m_l) \rceil$.
(3) if ($a_l > (m_h + (k_l - 1) * M_h)/k_l$ and $M_h \neq m_h$) then
    /* Known range for average to the right of smallest
         inferred range. */
    (a) $k_l = \lceil (M_h - m_h)/(M_h - a_l) \rceil$.
}

function Obviously_Unsatisfiable ($m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h, k_l, k_h$) {
/* Infeasible ranges. */
(1) if ($k_l > k_h$ or $m_l > m_h$ or $M_l > M_h$ or $s_l > s_h$ or $a_l > a_h$) then
      return 1.
(2) else return 0.
}

Switch_Signs ($m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h$) {
(1) $[t1, t2] = [-M_h, -M_l]$; $[M_l, M_h] = [-m_h, -m_l]$; $[m_l, m_h] = [t1, t2]$.
(2) $t = -a_l$; $a_l = -a_h$; $a_h = t$.
```

```
(3) $t = -s_l;\ s_l = -s_h;\ s_h = t$.
}

In_Sum_Gap_NP $(m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h, k_l, k_h)$ {
/* Check if there is some $k$ in $[k_l, k_h]$ such that $[s_l, s_h]$
   overlaps the intersection of $[k * a_l, k * a_h]$ and
   $[(k-1) * m_l + M_l, m_h + (k-1) * M_h]$. */
/* Case A: Determine a lower count bound based on
   sum, min, max. */
(1) if $(s_h < (k_l - 1) * m_l + M_l)$ then
      /* sum to the left of smallest inferred range from
         min, max. */
      (a) $[k_1, k_3] = [\lceil (s_h + m_l - M_l)/m_l \rceil, k_h]$.
(2) else if $(s_l > m_h + (k_l - 1) * M_h)$ then
      /* sum to the right of smallest inferred range from
         min, max. */
      (a) $[k_1, k_3] = [\lceil (s_l + M_h - m_h)/M_h \rceil, k_h]$.
(3) else $[k_1, k_3] = [k_l, k_h]$.
/* Case B: check if $[s_l, s_h]$ overlaps $[k * a_l, k * a_h]$ for any
            $k \in [k_l, k_h]$. */
(4) define $k_1'$ and $k_2'$ by $s_l = k_1' * a_h - k_2', 0 \leqslant k_2' < a_h$, and integer $k_1'$.
      /* multiset cardinality must be $\geqslant k_1'$, for sum $\geqslant s_l$. */
(5) define $k_3'$ and $k_4'$ by $s_h = k_3' * a_l + k_4', 0 \leqslant k_4' < a_l$, and integer $k_3'$.
      /* multiset cardinality must be $\leqslant k_3'$, for sum $\leqslant s_h$. */
(6) if ($[k_1', k_3']$ is not feasible) then    /* in a gap, based on
    average alone */
    return 1.
(7) if ($[k_1, k_3], [k_1', k_3']$ and $[k_l, k_h]$ all overlap) then
      /* any $k$ in the intersection of the three ranges is a
         witness. */
      return 0.
(8) else return 1.
}

In_Sum_Gap_PP $(m_l, m_h, M_l, M_h, s_l, s_h, a_l, a_h, k_l, k_h)$ {
/* Check if there is some $k$ in $[k_l, k_h]$ such that $[s_l, s_h]$ overlaps
   the intersection of $[k * a_l, k * a_h]$ and
   $[(k-1) * m_l + M_l, m_h + (k-1) * M_h]$. */
/* Case A: check if $[s_l, s_h]$ overlaps $[(k-1) * m_l + M_l, m_h + (k-1) * M_h]$
   for any $k \in [k_l, k_h]$. */
(1) define $k_1$ and $k_2$ by $s_l = m_h + (k_1 - 1) * M_h - k_2, 0 \leqslant k_2 < M_h$, and
    integer $k_1$.
      /* multiset cardinality must be $\geqslant k_1$, for sum $\geqslant s_l$. */
```

```
(2) define $k_3$ and $k_4$ by $s_h = (k_3 - 1) * m_l + M_l + k_4, 0 \leqslant k_4 < m_l$,
    and integer $k_3$.
     /* multiset cardinality must be $\leqslant k_3$, for $sum \leqslant s_h$. */
(3) if ($[k_1, k_3]$ is not feasible) then     /* in a gap, based on
    min and max alone */
    return 1.
/* Case B: check if $[s_l, s_h]$ overlaps $[k * a_l, k * a_h]$ for any
            $k \in [k_l, k_h]$. */
(4) define $k_1'$ and $k_2'$ by $s_l = k_1' * a_h - k_2', 0 \leqslant k_2' < a_h$, and integer $k_1'$.
     /* multiset cardinality must be $\geqslant k_1'$, for $sum \geqslant s_l$. */
(5) define $k_3'$ and $k_4'$ by $s_h = k_3' * a_l + k_4', 0 \leqslant k_4' < a_l$, and integer $k_3'$.
     /* multiset cardinality must be $\leqslant k_3'$, for $sum \leqslant s_h$. */
(6) if ($[k_1', k_3']$ is not feasible) then     /* in a gap, based on
    average alone */
    return 1.
(7) if ($[k_1, k_3], [k_1', k_3']$ and $[k_l, k_h]$ all overlap) then
     /* any $k$ in the intersection of the three ranges is a
     witness. */ return 0.
(8) else return 1.
}
```

Theorem A.1. *Function* Gen_Multiset_Ranges *returns* 1 *iff there exist* $k > 0$ *real numbers,* $k_l \leqslant k \leqslant k_h$*, such that the minimum of the k numbers is in* $[m_l, m_h]$*, the maximum of the k numbers is in* $[M_l, M_h]$*, the sum of the k numbers is in* $[s_l, s_h]$*, and the average of the k numbers is in* $[a_l, a_h]$.

Further, Gen_Multiset_Ranges *has polynomial time complexity in the size of the representation of the input.*

Proof. We prove the first part of the theorem by showing that the algorithm returns 1 if and only if the given constraints along with the four axioms of Theorem 4.2 are satisfiable.

Consider Steps (1) and (2) of Gen_Multiset_Ranges. Step (1a) generates all constraints on *min, max* and *average* that can be inferred from the given range constraints on *min, max* and *average* and the axioms. Step (1b) extends these by generating all constraints on *count* that can be inferred from the given range constraints on *min, max* and *average* and the axioms. Note that all the constraints inferred above are range constraints on *min, max, average* and *count*.

If function Obviously_Unsatisfiable returns 1, the resultant set of constraints is clearly unsatisfiable. If it returns 0, the conjunction of the given range constraints on *min, max, count* and *average* and all the axioms is satisfiable.

All elements in the multiset have to lie in the range $[m_l, M_h]$; the minimum and maximum elements are additionally constrained to lie in the ranges $[m_l, m_h]$ and $[M_l, M_h]$ respectively. If the multiset has i elements, axioms (2) and (3) are satisfied if and only if the multiset has a sum in the range

$$[(i-1)*m_l + M_l, m_h + (i-1)*M_h].$$

Also, the average value of the multiset elements has to lie in the range $[a_l, a_h]$. If the multiset has i elements, axiom (4) is satisfied if and only if the multiset has a sum in the range:

$$[i*a_l, i*a_h].$$

Consequently, if the count of the multiset is constrained to lie in the range $[k_l, k_h]$, the sum can take values only from the union of the ranges:

$$\bigcup_{i=k_l}^{k_h} ([(i-1)*m_l + M_l, m_h + (i-1)*M_h] \cap [i*a_l, i*a_h]).$$

In general, this union of ranges may not be convex; there may be gaps.

Thus, the conjunction of the given constraints and axioms (1)–(4) is satisfiable if and only if there is an i such that the given range on sum, $[s_l, s_h]$ overlaps with the range: $[(i-1)*m_l + M_l, m_h + (i-1)*M_h] \cap [i*a_l, i*a_h]$. The algorithm for testing the above has three cases, based on the location of the $[m_l, M_h]$ range with respect to zero.

- The first case is when the $[m_l, M_h]$ range includes zero. Three subcases arise based on the location of the $[a_l, a_h]$ range with respect to zero.

  The first subcase is when the $[a_l, a_h]$ range includes zero; in this case the union of the ranges is convex, and is given by

  $$[(k_h-1)*m_l + M_l, m_h + (k_h-1)*M_h] \cap [k_h*a_l, k_h*a_h].$$

  To check that the given range for sum, $[s_l, s_h]$, overlaps with this intersection of ranges, it suffices to check that $[s_l, s_h]$ intersects with each of the ranges separately, since $[(k_h-1)*m_l + M_l, m_h + (k_h-1)*M_h]$ and $[k_h*a_l, k_h*a_h]$ are known to intersect at 0. Steps (3a) and (3b) of Gen_Multiset_Ranges check for this subcase.

  The second subcase is when the $[a_l, a_h]$ range includes only negative numbers, and the third subcase is when the $[a_l, a_h]$ range includes only positive numbers. These two subcases are symmetric, and we transform the second subcase into the third subcase in Step (3c) of Gen_Multiset_Ranges, and consider only the third subcase in detail in Step (3d).

  In the third subcase, the sum lies within the range

  $$[(k_h-1)*m_l + M_l, m_h + (k_h-1)*M_h] \cap [k_l*a_l, k_h*a_h]$$

  but not all values in this range are feasible – there may be gaps. The conjunction of constraints is unsatisfiable if and only if the $[s_l, s_h]$ range lies outside $[(k_h-1)*$

$m_l+M_l, m_h+(k_h-1)*M_h] \cap [k_l*a_l, k_h*a_h]$, or entirely within one of the gaps. Since Function Tighten_Count_Bounds was invoked in Step (1b) of Gen_Multiset_Ranges, the two ranges $[(k_h-1)*m_l+M_l, m_h+(k_h-1)*M_h]$ and $[k_l*a_l, k_h*a_h]$ overlap. Consequently, from the property of ranges, it follows that to check that the $[s_l, s_h]$ range lies outside the intersection of these two ranges, it suffices to check that $[s_l, s_h]$ lies outside at least one of the two ranges; steps (3a) and (3d)(i) check for this. Steps (3d)(ii) and (iii) check for the second possibility, viz., $[s_l, s_h]$ lies entirely within one of the gaps of:

$$\bigcup_{i=k_l}^{k_h} ([(i-1)*m_l+M_l, m_h+(i-1)*M_h] \cap [i*a_l, i*a_h]).$$

Tighten_Count_Bounds has adjusted k_l to ensure that for k_l is the *smallest* i for which the ranges $[(i-1)*m_l+M_l, m_h+(i-1)*M_h]$ and $[i*a_l, i*a_h]$ overlap. Further, Tighten_MMA_Bounds (invoked in Step (1a) of Gen_Multiset_Ranges has tightened M_l, m_h, a_l and a_h to ensure each of $m_l \leqslant M_l, m_l \leqslant a_l$, $m_h \leqslant M_h$ and $a_h \leqslant M_h$ hold. The above two points guarantee that for all $i \geqslant k_l$ it is the case $[(i-1)*m_l+M_l, m_h+(i-1)*M_h]$ and $[i*a_l, i*a_h]$ overlap. Hence, from the property of ranges, it follows that to check that $[s_l, s_h]$ does not fall entirely within a gap of

$$\bigcup_{i=k_l}^{k_h} ([(i-1)*m_l+M_l, m_h+(i-1)*M_h] \cap [i*a_l, i*a_h])$$

it suffices to check that there is at least one i in $[k_l, k_h]$, such that $[s_l, s_h]$ overlaps with each of $[(i-1)*m_l+M_l, m_h+(i-1)*M_h]$ and with $[i*a_l, i*a_h]$. Function In_Sum_Gap_NP checks for this possibility as follows: (a) it computes the range $[k_1, k_3]$ such that for each i in $[k_1, k_3]$, the range $[s_l, s_h]$ overlaps with $[(i-1)*m_l+M_l, m_h+(i-1)*M_h]$; (b) it computes the range $[k_1', k_3']$ (using the same technique as in Multiset_Ranges) such that for each i in $[k_1', k_3']$, the range $[s_l, s_h]$ overlaps with $[i*a_l, i*a_h]$; (c) finally, it checks that there is some i which lies in each of the three ranges $[k_l, k_h], [k_1, k_3]$ and $[k_1', k_3']$, which provides the required witness.

- The second case is when the $[m_l, M_h]$ range includes only negative numbers, and hence the average must also be negative. Function Tighten_MMA_Bounds has tightened the $[a_l, a_h]$ range to include only negative numbers. This is symmetric to the third case (discussed in detail below), and Switch_Signs (invoked in Step (4a)) transforms the second case into the third case.
- The third case is when the $[m_l, M_h]$ range includes only positive numbers, and hence the average must also be positive. Function Tighten_MMA_Bounds has tightened the $[a_l, a_h]$ range to include only positive numbers. In this case, the sum lies within the range

$$[(k_l-1)*m_l+M_l, m_h+(k_h-1)*M_h] \cap [k_l*a_l, k_h*a_h]$$

but not all values in this range are feasible – as before, there may be gaps. The conjunction of constraints is unsatisfiable if and only if the $[s_l, s_h]$ range lies outside

$[(k_l - 1)*m_l + M_l, m_h + (k_h - 1)*M_h] \cap [k_l * a_l, k_h * a_h]$, or entirely within one of the gaps. Since Function Tighten_Count_Bounds was invoked in Step (1b) of Gen_Multiset_Ranges, the two ranges $[(k_l - 1)*m_l + M_l, m_h + (k_h - 1)*M_h]$ and $[k_l * a_l, k_h * a_h]$ overlap. Consequently, from the property of ranges, it follows that to check that the $[s_l, s_h]$ range lies outside the intersection of these two ranges, it suffices to check that $[s_l, s_h]$ lies outside at least one of the two ranges; steps (5a) and (5b) of Gen_Multiset_Ranges check for this. Steps (5c) and (5d) check for the second possibility, viz., $[s_l, s_h]$ lies entirely within one of the gaps of

$$\bigcup_{i=k_l}^{k_h} ([(i-1)*m_l + M_l, m_h + (i-1)*M_h] \cap [i*a_l, i*a_h])$$

As in the third subcase of the first case above, it suffices to check that there is at least one i in $[k_l, k_h]$, such that $[s_l, s_h]$ overlaps with each of $[(i-1)*m_l + M_l, m_h + (i-1)*M_h]$ and with $[i*a_l, i*a_h]$. Function In_Sum_Gap_PP checks for this possibility as follows: (a) it computes the range $[k_1, k_3]$ (using the same technique as in Multiset_Ranges) such that for each i in $[k_1, k_3]$, the range $[s_l, s_h]$ overlaps with $[(i-1)*m_l + M_l, m_h + (i-1)*M_h]$; (b) it computes the range $[k'_1, k'_3]$ (using the same technique as in Multiset_Ranges) such that for each i in $[k'_1, k'_3]$, the range $[s_l, s_h]$ overlaps with $[i*a_l, i*a_h]$; (c) finally, it checks that there is some i which lies in each of the three ranges $[k_l, k_h], [k_1, k_3]$ and $[k'_1, k'_3]$, which provides the required witness.

This concludes the proof of the first part of the theorem.

The proof of the second part of the theorem is straightforward because the number of steps in Gen_Multiset_Ranges is bounded above by a constant, and each step is polynomial in the size of representation of the input. □

References

[1] J. Jaffar, S. Michaylov, P.J. Stuckey and R.H.C. Yap, The CLP($\mathscr{R}$) language and system, *ACM Trans. Programming Languages Systems* **14** (1992) 339–395.

[2] A.Y. Levy, I.S. Mumick and Y. Sagiv, Query optimization by predicate move-around, in: *Proc. Internat. Conf. on Very Large Databases*, Santiago, Chile, September (1994) 96–107.

[3] K. Marriott and P.J. Stuckey, Semantics of constraint logic programs with optimization, *Lett. Programming Languages Systems* **2** (1993) 197–212.

[4] J. Melton and A.R. Simon, *Understanding the New SQL: A Complete Guide* (Morgan Kaufmann, San Francisco, CA, 1993).

[5] K.A. Ross, D. Srivastava, P.J. Stuckey and S. Sudarshan, Foundations of aggregation constraints, in: *Proc. 2nd Internat. Workshop on Principles and Practice of Constraint Programming*, Orcas Island, WA, Lecture Notes in Computer Science, Vol. 874 (Springer, Berlin, 1994) 193–204.

[6] A. Schrijver, *Theory of Linear and Integer Programming*, Discrete Mathematics and Optimization (Wiley-Interscience, New York, 1986).

[7] D. Srivastava and R. Ramakrishnan, Pushing constraint selections, *J. Logic Programming* **16** (1993) 361–414.

[8] P.J. Stuckey and S. Sudarshan, Compiling query constraints, in: *Proc. ACM Symp. on Principles of Database Systems*, Minneapolis, MN, May (1994) 56–67.

[9] S. Sudarshan and R. Ramakrishnan, Aggregation and relevance in deductive databases, in: *Proc. Internat. Conf. on Very Large Databases*, Barcelona, Spain, September (1991) 501–512.