Towards a Theory of DATA-DIFF: Optimal Synthesis of Succinct Data Modification Scripts

Tana Wattanawaroon
University of Illinois (UIUC)
wattana2@illinois.edu

Stephen Macke
University of Illinois (UIUC)
smacke@illinois.edu

Aditya Parameswaran
University of Illinois (UIUC)
adityagp@illinois.edu

ABSTRACT

This paper addresses the DATA-DIFF problem: given a dataset and a subsequent version of the dataset, find the shortest sequence of operations that transforms the dataset to the subsequent version, under a restricted family of operations. We consider operations similar to SQL UPDATE, each with a *condition* (WHERE) that matches a subset of tuples and a *modifier* (SET) that makes changes to those matched tuples. We characterize the problem based on different constraints on the attributes and the allowed conditions and modifiers, providing complexity classification and algorithms in each case.

1. INTRODUCTION

Over the course of data analysis, data scientists routinely generate versions of datasets by performing various data curation and cleaning operations, including updating, normalizing, fixing, adding, or deleting attribute values or rows, or adding or deleting new features or columns. They may use various ad-hoc tools for performing these edit operations, including scripting tools like sed, awk, or perl, or programming languages, like R or Python. Each such new dataset version is stored in a networked file system and shared with other data scientists [5, 22, 6]. Usually, however, the sequence of edit operations or the script that was used to generate the new version is not recorded along with the new version—since it may have been the result of a quick-and-dirty update; and even if the script is recorded, since the script may be in various programming or scripting languages, it may be hard to decipher or reverse-engineer the sequence of edit operations performed within this script.

To tackle this issue, in this paper, we introduce the DATA-DIFF problem: *given a dataset D_S and a subsequent dataset D_T that was derived from D_S, can we synthesize the most succinct sequence of edit operations, Δ, that transforms D_S to D_T?* Our target is SQL edit operations that can be efficiently executed in relational databases. We call this problem the DATA-DIFF problem as the *data*-analog of the traditional text *diff*, or differentiation problem, often used in source code versioning systems to synthesize the sequence of edit operations that resulted in a new version.

There are three reasons why solving DATA-DIFF, i.e., synthesizing a succinct sequence of edit operations, is valuable: *understanding, generalization, and compactness*. First, the data-diff helps users compactly understand the edit operations that have been made to generate a new version D_T from D_S, without having to read through a long programming script; second, it allows us to potentially record and recreate the edit operations so that they can be similarly applied to other datasets; and third, instead of storing D_T, we can simply store the sequence of edit operations, which, since it is written in SQL, is often smaller.

Our Focus. In this paper, *our key contribution is to introduce the* DATA-DIFF *problem and study it from a theoretical perspective, aiming to characterize the complexity of the problem and understand when the problem becomes intractable*. We focus on recovering edits to a single relation R, with edit operations that follow the following template:

$$\texttt{UPDATE}\ R\ \texttt{SET}\ \langle U \rangle\ \texttt{WHERE}\ \langle C \rangle;$$

We characterize the complexity of DATA-DIFF across three dimensions:

1. *[characteristics]* the attributes that may be used within U and C: we call an attribute *read-only* (*write-only*) if it can be used within C (U) but not U (C), *read-write* if it can be used within U and C, and *inaccessible* if should not be used within either U or C;
2. *[modifiers]* the space of transformations that can be used within U: we span basic assignment operations, as well as arithmetic operations; and
3. *[conditions]* the space of conditions that can be used within C: we span both equality conditions, $\leq$ and $\geq$, and range-based conditions.

In any of these cases, the user will specify the space of attribute characteristics, modifiers, and conditions, and the system will then automatically synthesize the smallest sequence of edit operations. Next, we illustrate the challenges in solving DATA-DIFF using a simple example.

EXAMPLE 1.1 (MOTIVATING EXAMPLE). *Consider the scenario in Figure 1, where we depict three versions of a given relation R, namely R_1, R_2, and R_3, with the primary key* K. *Using* K*, we can identify how individual tuples have evolved across the versions. For this simple example, we do not have any tuples being added or deleted, nor do we have any attributes being added or deleted. Our goal is to solve* DATA-DIFF *under the specification that we have one read-write attribute,* A*, and one write-only attribute,* B. *(since* K *is the primary key, in this case, it has been denoted an inaccessible attribute, which means that it cannot be used in the modifier or in the condition.)*

	$\leftarrow$	$+$	$\leftarrow +$	aff
$=$	[Thm 4.1] $O(N \log N)$	[Thm 4.1] $O(N \log N)$	[Thm 4.1] $O(N \log N)$	[Thm 4.1] $O(N \log N)$
$\leq$	[Thm 4.2] $O(N \log N)$	[Thm 4.3] $O(N \log N)$	[Thm 4.4] $O(N \log N)$	[Thm 4.5] $O(N \log N)$
$\leq \geq$	[Thm 4.6] $O(N^2)$	[Thm 4.7] NP-hard [Thm B.1] $O(N \log N)$ to $+1$-approx	[Thm 4.17] NP-hard	[Thm 4.19] NP-hard
R	[Thm 4.20] $O(N^4)$	[Thm 4.21] NP-hard [Thm B.2] $O(N \log N)$ to $\times 2$-approx	[Thm 4.28] NP-hard	[Thm 4.30] NP-hard
U	[Thm 4.31] NP-hard	[Thm 4.38] NP-hard	[Thm 4.38] NP-hard	[Thm 4.38] NP-hard

Table 1: Results summary for the one read-only attribute, one write-only attribute case

K	A	B
c17	1	0
3bd	5	0
97a	3	0
1b8	0	0
94f	4	0
842	2	0

R_1

K	A	B
c17	1	1
3bd	5	3
97a	3	2
1b8	0	1
94f	4	2
842	2	1

R_2

K	A	B
c17	7	1
3bd	8	3
97a	8	2
1b8	7	1
94f	8	2
842	7	1

R_3

Figure 1: Example of a setting with three versions of a relation R where we want to solve DATA-DIFF with one read-write column A, and one write-only column B.

One approach to solving DATA-DIFF *between R_1 and R_2, which only differ in the value of* B*, is to use six edit operations of the following form:*

$$\texttt{UPDATE } R \texttt{ SET B} = b_k \texttt{ WHERE A} = a_k;$$

one for each tuple. Recall that A *being a read-write attribute, can be used for the (equality) condition, while* B *being a write-only attribute can be used for the (assignment) modifier. If we relax the space of conditions to admit $\leq$ and $\geq$ in addition to equality, then there is a shorter sequence of three edit operations:*

$$U_1 : \texttt{UPDATE } R \texttt{ SET B} = 1 \texttt{ WHERE A} \leq 2;$$
$$U_2 : \texttt{UPDATE } R \texttt{ SET B} = 2 \texttt{ WHERE A} \geq 3;$$
$$U_3 : \texttt{UPDATE } R \texttt{ SET B} = 3 \texttt{ WHERE A} = 5;$$

Notice that the order of operations is important: $U_1 \rightarrow U_2 \rightarrow U_3$ does not give the same result as $U_1 \rightarrow U_3 \rightarrow U_2$. Similarly, to solve DATA-DIFF *between R_2 and R_3 (wherein the read-write attribute* A *is transformed), we could use as many as six operations, but in fact two operations suffice:*

$$U_4 : \texttt{UPDATE } R \texttt{ SET A} = 7 \texttt{ WHERE A} \leq 2;$$
$$U_5 : \texttt{UPDATE } R \texttt{ SET A} = 8 \texttt{ WHERE A} \leq 5;$$

Once again, $U_5 \rightarrow U_4$ does not provide the same result as $U_4 \rightarrow U_5$. As it turns out, this sequence of three edit operations for the first case, and two edit operations for the second case are the smallest possible sequences, based on modifiers that are assignment-based and on conditions that are based on $\leq$, $\geq$ or equality. Indeed, when we expand the space of modifiers to not just assignment, but also addition or subtraction, the DATA-DIFF *problem becomes even more challenging. Overall, depending on the instance, the smallest sequence of operations may be as small as one operation, or as many as $O(N)$ (typically non-commutative) operations, where N is the number of tuples, making it challenging to navigate.*

Related Work. The DATA-DIFF problem is related to the *view synthesis* problem, a complementary problem that targets the following setting: given D, D', find the most succinct single view definition Q using selection operations such that $D' \approx Q(D)$ [13, 28]. For example,

$$Q(R) : \texttt{SELECT} * \texttt{ FROM } R \texttt{ WHERE A} = k;$$

is a view definition Q that selects all of the tuples that match a certain criteria from R. This work has been extended in multiple directions that we will discuss in Section 2. DATA-DIFF is much harder than view synthesis, due to non-commutativity of edit operations, leading to intractability even for relations with a finite number of attributes, while view synthesis is only intractable when the number of attributes is allowed to vary. DATA-DIFF is also related to the problem of synthesizing string transformations—the difference between that line of work and ours is the difference between learning regular expressions and learning SQL modification statements: the space of operations and therefore the techniques and contributions are very different. We will cover related work in more detail in Section 2.

Contributions. We introduce the family of DATA-DIFF problems under different attribute characteristics and the space of modifiers and conditions of interest. We identify a "base case", fully characterize it, and then identify a generalization and proceed to show hardness results in the generalization. The characterization summary can be seen in Tables 1 and 2.

2. RELATED WORK

DATA-DIFF is related to the topics of view synthesis and learning string transformations from examples.

View Synthesis. The view synthesis problem originally defined the question of synthesizing a view definition given two database instances, which was originally laid out in Das Sarma et al. [13] and Tran et al. [28] and extended in various ways since then [21, 31, 29, 30, 23]. For example, recent work has extended the original work on the view synthesis problem to the problem of synthesizing join queries [30] and top-k queries [23]. Other work has extended the view synthesis problem to an iterative one, with the user being asked to confirm the presence or absence of tuples one at a time in order to *learn* an appropriate user query for various settings [8, 10, 7, 9, 1]. Earlier work studied the problem of checking if there exists a view definition without synthesizing it [14].

	$\leftarrow$	$+$	$\leftarrow +$	aff
$=$	[Thm 5.1] NP-hard	[Thm 5.2] NP-hard	[Thm 5.2] NP-hard	[Thm 5.2] NP-hard
$\leq$	?	[Thm 5.4] NP-hard	[Thm 5.4] NP-hard	[Thm 5.4] NP-hard
$\leq\geq$	?	[Thm 5.6] NP-hard	[Thm 5.6] NP-hard	[Thm 5.6] NP-hard
R	[Thm 5.7] NP-hard	[Thm 5.6] NP-hard	[Thm 5.6] NP-hard	[Thm 5.6] NP-hard
U	[Thm 5.9] NP-hard	[Thm 5.6] NP-hard	[Thm 5.6] NP-hard	[Thm 5.6] NP-hard

Table 2: Results summary for the multiple read-only attributes, one write-only attribute case

Another related direction is that of synthesizing a view given multiple pairs of database instances, introduced in the context of data integration as a problem of learning schema mappings from data examples [11, 16, 2].

While all of these directions are interesting and relevant to the DATA-DIFF problem, note that the DATA-DIFF problem is substantially harder than the view synthesis problem, even when applied on a single relation R. First, edit operations, unlike selection operations, are non-commutative and therefore cannot be applied in any order. Thus, the order of operations, while unimportant in view synthesis, is crucial in DATA-DIFF. Second, the ability to use multiple operations is not very important in the view synthesis problem, since we can simply overload the `WHERE` clause to be more complex; in the DATA-DIFF problem on the other hand, multiple edit operations offer substantial additional power, e.g., transforming R_1 to R_3 as given in Figure 1 would be difficult using one operation.

For these reasons, we find that the problem of DATA-DIFF becomes intractable much sooner—even on edit operations on a single relation with two or three attributes, while the view synthesis problem is only intractable when the number of attributes is allowed to vary. In fact, notice that DATA-DIFF problem has a view synthesis problem as a sub-problem: for the case where a number of tuples have been deleted from D to D', we could use the results from the view synthesis problem to identify the condition that selects all of the tuples to be deleted, and therefore we can inherit all of the same hardness results for those cases. To understand the complexity of DATA-DIFF independent of view synthesis, we focus on the case when no tuples have been deleted.

String Transformations. A related direction from the program analysis community focuses on the learning of string transformations given input-output examples [17, 18], extending it to various settings in cleaning data in spreadsheets, such as transforming times and dates [24], numbers [25], text [20], and miscellaneous data types [26], changing the structure of spreadsheet tables [19], as well as extracting structured data from semi-structured spreadsheet data [4]. Like us, this body of work targets *edit operations*—however, these operations are regular-expression like operations that are applied to transform each value in a set of values (e.g., extracting the first three digits of a phone number). Each such value can be then treated as a training example for learning the edit operation. Instead, we focus SQL operations: not as fine-grained at the value level, but are more fine-grained at a global level, admitting conditional clauses, e.g.: if $\mathsf{A} \in [a, b]$, add c to B. Thus, the difference in the space of operations under consideration can be seen as the difference between regular expressions being applied to a set of values (in the string transformation case), versus a sequence of SQL modification statements (in our case). In addition, we do not attempt to precisely characterize the complexity of learning transformations as a function of the space of operations, preferring instead to prove soundness and completeness.

3. PROBLEM DEFINITIONS

In this section, we formulate the problem of finding a succinct description of changes between two datasets. We define the *diff*, which captures the notion of the description of changes, along with some relevant terms. Then we formally define the problem and scope of operations that are of interest in this paper.

To understand and characterize the complexity frontier of the DATA-DIFF problem, where the goal is to find the most succinct sequence of operations that transform D_S to D_T, we assume that D_S and D_T are both single relations R_S and R_T with the same schema, along with an unmodified primary key attribute (e.g., `employeeID`, `transactionID`) that allows us to track how tuples have evolved—thus, there is a one-to-one correspondence between the tuples in R_S and R_T. We further assume that the primary key values in R_S and R_T are the same, essentially guaranteeing that there are no insertions or deletions. Thus, overall, our setting is one where there is a single relation (with a primary key) being modified by *data modification operations*, but there are no insertions or deletions (of tuples or attributes), or modification of schema. We will formalize these assumptions later in this section.

Rationale for Assumptions. We now briefly describe why we make these simplifying assumptions to focus on DATA-DIFF for *data modification operations*. When there is an unmodified primary key, insertions of new tuples are easy to identify, and trivial to represent as either a single batch `INSERT` statement, or insertion of one tuple at a time, with no further compression possible or necessary. Deletions of tuples, on the other hand, ends up being equivalent to the view synthesis problem (as described in Section 2), since we need to identify a query Q that selects precisely the tuples that were deleted, and thus we can reuse existing results from related work previously discussed. Since there is an unmodified primary key, if we know which attributes are deleted, they can all be dropped in one single `ALTER` statement, along with any attributes that are renamed. Naturally, attributes that are inserted are a lot more complicated, since, in general, a succinct description for new attributes would fall under the realm of pattern recognition—this is outside the scope of our work, which focuses on data modification.

3.1 Similar Relations, Diff, and Best Diff

First, we introduce the notion of attribute characteristics.

Different settings of characteristics play a major role in determining the hardness of the problem. Here, $\mathcal{A}$ is the set of *read* attributes on which conditions are based, and $\mathcal{B}$ is the set of *write* attributes on which modifiers make changes.

In general, we can detect which attributes $\mathcal{B}$ have been modified automatically, but we allow the user to specify the set of attributes $\mathcal{A}$ explicitly, since they may not want the system to use all attributes to infer SQL data modification scripts. For example, if the user knows that `gender` is never an attribute that is read when modifying the `GPA`, they can exclude `gender` from the set of attributes in $\mathcal{A}$.

DEFINITION 3.1 (ATTRIBUTE CHARACTERISTICS). *An attribute* $\mathsf{A} \in \mathcal{A} \cup \mathcal{B}$ *is called* read-only *if* $\mathsf{A} \in \mathcal{A}$ *and* $\mathsf{A} \notin \mathcal{B}$*, or* write-only *if* $\mathsf{A} \notin \mathcal{A}$ *and* $\mathsf{A} \in \mathcal{B}$*, or* read-write *otherwise.*

Second, we define "similar" relations. The DATA-DIFF problem concerns two relations, one representing the "before" snapshot and the other representing the "after" snapshot. As previously discussed, we will not consider adding or removing attributes, and we want to exclude insertions and deletions of tuples from our family of possible operations; we only consider "update" operations. Thus, the two relations should have the same schema and the same number of rows.

In addition, we want to be able to tell which tuples map to which in the two relations, hence the requirement that the two relations share a primary key K, and that the sets of primary keys are identical and cannot be modified. The primary key serves as an identifier of the tuples in the two relations.

DEFINITION 3.2 (SIMILAR RELATION). *For an attribute* K *and sets of attributes* $\mathcal{A}$ *and* $\mathcal{B}$*, neither of which contains* K*, two relations* R_S *and* R_T *are* $(\mathsf{K}, \mathcal{A}, \mathcal{B})$-similar *iff*

- R_S *and* R_T *both have schema* $\{\mathsf{K}\} \cup \mathcal{A} \cup \mathcal{B}$*, and* K *is their primary key, and*
- $\pi_\mathsf{K}(R_\mathsf{S}) = \pi_\mathsf{K}(R_\mathsf{T})$ *(here* π *is the projection operator in relational algebra).*

In other words, R_S and R_T have the same schema with one primary key attribute containing the same set of values. This implies that the number of tuples in R_S and R_T are equal, and that we can match the tuples in R_S and R_T one-to-one based on the primary key. Note that we could simply define two relations as similar if they have the same schema, but our definition explicitly references the sets $\mathcal{A}$ and $\mathcal{B}$ as a notational convenience that will help with later exposition.

Third, we define what an operation is, and what it does. It must obey the read-write characteristics of the attributes.

DEFINITION 3.3 (OPERATION). *For sets of attributes* $\mathcal{A}$ *and* $\mathcal{B}$*, an* $(\mathcal{A}, \mathcal{B})$-operation $f = (p, u)$ *has a* condition p *on attributes in* $\mathcal{A}$ *and a* modifier u *on attributes in* $\mathcal{B}$*. Let* $f(R_\mathsf{S}) = R_\mathsf{T}$ *if and only if* R_T *is the resulting relation after calling the SQL command:*

`UPDATE` R_S `SET` u `WHERE` p.

Note that the result of an $(\mathcal{A}, \mathcal{B})$-operation is $(\mathsf{K}, \mathcal{A}, \mathcal{B})$-similar to the operand; i.e., if R_S and R_T are relations such that $f(R_\mathsf{S}) = R_\mathsf{T}$ and R_S has schema $\{\mathsf{K}\} \cup \mathcal{A} \cup \mathcal{B}$, then R_S and R_T are $(\mathsf{K}, \mathcal{A}, \mathcal{B})$-similar.

We now define the *diff*, the sequence of operations transforming the "before" relation to the "after" relation, along with its associated cost. In the following definitions, we consider an attribute K, sets of attributes $\mathcal{A}$ and $\mathcal{B}$ neither of which contain K, a set of $(\mathcal{A}, \mathcal{B})$-operations $\mathcal{F}$, and $(\mathsf{K}, \mathcal{A}, \mathcal{B})$-similar relations R_S and R_T.

DEFINITION 3.4 (DIFF). *A sequence of operations* $F = (f_1, \ldots, f_m)$ *where* $f_i \in \mathcal{F}$ *for each* $i \in [m]$ *is called a* diff *between* R_S *and* R_T *under* $\mathcal{F}$*, also written* $F(R_\mathsf{S}) = R_\mathsf{T}$*, if there are relations* $R_0, \ldots, R_m$ *such that*

- $R_0 = R_\mathsf{S}$*,*
- $R_m = R_\mathsf{T}$*, and*
- $f_i(R_{i-1}) = R_i$ *for all* $i \in [m]$.

Let $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F})$ *denote the set of all diffs between* R_S *and* R_T *under* $\mathcal{F}$*.*

DEFINITION 3.5 (COST). *Each operation* f *has an associated integer* cost*, denoted* $cost(f)$*. The cost of a diff* $F = (f_1, \ldots, f_m)$ *is defined as* $cost(F) = \sum_{i=1}^{m} cost(f_i)$*.*

DEFINITION 3.6 (BEST DIFF). *A diff* $F \in \Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F})$ *is called a* best diff *between* R_S *and* R_T *under* $\mathcal{F}$ *if it has the smallest cost in* $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F})$*; i.e., for any diff* $F' \in \Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F})$*, we have* $cost(F) \leq cost(F')$*. We also write that* F *is a best diff in* $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F})$*.*

Note that if $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F})$ is nonempty, then it must contain a best diff, by the well-ordering principle of integers.

3.2 Diff Problems

Next, we define the best diff problem that is the focal point of this paper.

DEFINITION 3.7 (BEST DIFF PROBLEM). *Fix a family of* $(\mathcal{A}, \mathcal{B})$*-operations* $\mathcal{F}$*. The best diff problem* $\mathrm{BD}(\mathcal{F})$ *is, given as input:*

- *an attribute* K*,*
- *attribute sets* $\mathcal{A}$ *and* $\mathcal{B}$*, neither of which contain* K*, and*
- *two* $(\mathsf{K}, \mathcal{A}, \mathcal{B})$*-similar* N*-tuple relations* R_S *and* R_T*, where all values are integers,*

find and return a best diff between R_S *and* R_T *under* $\mathcal{F}$ *if one exists, or correctly report that no diffs exist.*

Note that while we restrict relations R_S and R_T to integer values (for simple arguments of representation sizes), conditions and modifiers are not restricted to integers; real values can be used.

The following auxiliary definitions are used in proofs.

DEFINITION 3.8 (ATTRIBUTE VALUES). *For an attribute* A*,* $V_\mathsf{A}(R_\mathsf{S}, R_\mathsf{T})$ *is the set of all* A *values in relations* R_S *and* R_T*; in other words,*

$$V_\mathsf{A}(R_\mathsf{S}, R_\mathsf{T}) = \pi_\mathsf{A}(R_\mathsf{S}) \cup \pi_\mathsf{A}(R_\mathsf{T})$$

DEFINITION 3.9 (BOUNDARY AND LENGTH). *Let*

$$v_\mathsf{A}^{\max} = \max V_\mathsf{A}(R_\mathsf{S}, R_\mathsf{T}) + 1$$
$$v_\mathsf{A}^{\min} = \min V_\mathsf{A}(R_\mathsf{S}, R_\mathsf{T}) - 1$$

For an operation f, define the length $\ell(f)$ *as*

$$\ell(f) = \begin{cases} a - v_{\mathsf{A}}^{\min} & \text{if } f \text{ has the condition } \mathsf{A} \leq a \\ v_{\mathsf{A}}^{\max} - a & \text{if } f \text{ has the condition } \mathsf{A} \geq a \\ z - a + 1 & \text{if } f \text{ has the condition } \mathsf{A} \in [a, z] \end{cases}$$

For a sequence of operations $F = (f_1, \ldots, f_m)$, define the total length *of F as $\ell(F) = \sum_{i \in [m]} \ell(f_i)$.*

3.3 Families of Operations

Generally, $(\mathcal{A}, \mathcal{B})$-operations can be simple or complicated. Given two relations R_{S} and R_{T}, one might claim that there is a diff between them containing the following $(\mathcal{A}, \mathcal{B})$-operation as its only operation:

$$\left(\mathsf{A} \in \{2, 9, 11, 23\}, \mathsf{B} \leftarrow \left\lceil |\mathsf{B}|^{\sqrt{11}} \right\rceil - 7\mathsf{B}^2\right)$$

The given $(\mathcal{A}, \mathcal{B})$-operation has an overfitting condition and a complicated modifier, which makes it unlikely to be an operation actually used to transform R_{S} into R_{T} by, say, an accountant working on this database. Therefore, we would like to limit ourselves to operations that are relatively simple and are more likely to correspond to actual scenarios.

We describe families of $(\mathcal{A}, \mathcal{B})$-operations that are of interest in this paper. Here, A is an attribute from $\mathcal{A}$, and B is an attribute from $\mathcal{B}$.

Condition Types. We consider conditions p that are conjunctions of single-attribute clauses, i.e., statements in the form $p = p_1 \wedge \ldots \wedge p_h$, where the clauses p_i have the same type but are on different attributes. The condition p on $\mathcal{A}$ does not necessarily use all attributes in $\mathcal{A}$, but must use at least one (cannot be empty).

We consider the following single-attribute clause types.

symbol	name	condition	cost
$=$	equality	$\mathsf{A} = a$	1
$\leq$	at-most	$\mathsf{A} \leq a$	1
$\leq\geq$	at-most/at-least	$\mathsf{A} \leq a$ **or** $\mathsf{A} \geq a$	1
R	range	$\mathsf{A} \in [a, z]$	1
U	union-of-ranges	$\mathsf{A} \in \bigcup_{j=1}^{r} [a_j, z_j]$	varies

The cost is 1 *per operation* (not per clause), except in the union-of-ranges case, where the cost is $\kappa_0 + \kappa_1 \sum r$, where $\sum r$ is the sum of number of ranges over all clauses. Here, κ_0 and κ_1 are non-negative integers to be supplied as input.

For the at-most/at-least clause type, each clause can assume either of the two subtypes, and it is not required that all clauses use the same subtype. The clause type with only *at-least* condition is not explicitly discussed, because it is symmetric to using the at-most clause type.

Modifier Types. We only consider single-attribute modifiers in this paper. We consider the following modifier types.

symbol	name	modifier
$\leftarrow$	assignment	$\mathsf{B} \leftarrow b$
$+$	increment	$\mathsf{B} \leftarrow \mathsf{B} + b$
$\leftarrow +$	assignment/increment	$\mathsf{B} \leftarrow b$ **or** $\mathsf{B} \leftarrow \mathsf{B} + b$
aff	affine	$\mathsf{B} \leftarrow b\mathsf{B} + c$

The modifier type does not affect the cost of an operation.

Operations. The family of $(\mathcal{A}, \mathcal{B})$-operations using condition type ϕ and modifier type ω is denoted by $\mathcal{F}_{\omega}^{\phi}$. For example, $\mathcal{F}_{+}^{\leq}$ is the family of $(\mathcal{A}, \mathcal{B})$-operations where each operation uses an *at-most* condition and an *increment* modifier.

EXAMPLE 3.10. *Once again, consider the three versions of the relation R given in Figure 1, namely R_1, R_2, and R_3. Let $\mathcal{A} = \{\mathsf{A}\}$, $\mathcal{B} = \{\mathsf{A}, \mathsf{B}\}$, so that A is a read-write attribute and B is a write-only attribute.*

Let f_1 and f_2 be the following operations:

$$f_1 = (\mathsf{A} \leq 2, \mathsf{A} \leftarrow 7)$$
$$f_2 = (\mathsf{A} \leq 5, \mathsf{A} \leftarrow 8)$$

Here, f_1 is in $\mathcal{F}_{\leftarrow}^{\leq\geq}$ (and also $\mathcal{F}_{\leftarrow}^{\leq}$ and $\mathcal{F}_{\leftarrow+}^{\leq\geq}$), and so is f_2.

If $F = (f_1, f_2)$, then F is a diff between R_2 and R_3 under $\mathcal{F}_{\leftarrow}^{\leq\geq}$, and $cost(F) = 2$. Note that $F' = (f_2, f_1)$ is, however, not a diff between R_2 and R_3 under $\mathcal{F}_{\leftarrow}^{\leq\geq}$.

4. BASE CASE: BD1 PROBLEMS

In this section, we consider a "base case" of the best diff problems in terms of number of attributes and attribute characteristics, and present its characterization under different families of operations.

The BD1($\mathcal{F}$) problem is similar to the best diff BD($\mathcal{F}$) problem, but constrained to one read-only attribute, one write-only attribute, and no read-write attributes. Let $\mathcal{A} = \{\mathsf{A}\}$ and $\mathcal{B} = \{\mathsf{B}\}$, where A and B are different attributes.

We also assume that for any tuple $T_1 \in R_{\mathsf{S}}$ and $T_2 \in R_{\mathsf{T}}$, if $T_1.\mathsf{K} = T_2.\mathsf{K}$ then $T_1.\mathsf{A} = T_2.\mathsf{A}$, because an $(\mathcal{A}, \mathcal{B})$-operation cannot modify A values. If the assumption does not hold, we can immediately claim that a diff between R_{S} and R_{T} does not exist.

Table 1 summarizes the characterization. The table is roughly ordered according to how "powerful" each condition/modifier type is, although it is not necessarily true that a condition/modifier is a generalization of what precedes it. We encounter the hardness boundary at the families of operations $\mathcal{F}_{+}^{\leq\geq}$ and $\mathcal{F}_{\leftarrow}^{\mathsf{U}}$, where we present two main NP-hardness results via reductions from different problems. While the remaining NP-hardness results do not trivially follow from the two main results, they use similar reductions. Polynomial-time results are discussed more thoroughly in Appendix A.

4.1 With Equality Conditions

With *equality* ($\mathsf{A} = a$) conditions, tuples with different A values are independent of each other, in terms of how the $(\mathcal{A}, \mathcal{B})$-operations affect them. Therefore, the best diff problem under these families of operations is rather straightforward.

THEOREM 4.1. *The* BD1($\mathcal{F}_{\leftarrow}^{=}$), BD1($\mathcal{F}_{+}^{=}$), BD1($\mathcal{F}_{\leftarrow+}^{=}$), *and* BD1($\mathcal{F}_{\mathsf{aff}}^{=}$) *problems can be solved in $O(N \log N)$ time.*

4.2 With At-most Conditions

With *at-most* ($\mathsf{A} \leq a$) conditions, we can always reorder the operations within a diff (with some modifications) so that they affect the tuples in a certain order. Such reordering allows for polynomial time algorithms under all families of operations of interest.

More precisely, the at-most condition and the modifiers permit the theorems to utilize this property: if there is a best diff, there must be a best diff $F = (f_1, \ldots, f_m)$ in which for all $i, j \in [m]$, if $i < j$ and f_i has condition $\mathsf{A} \leq a_i$ and f_j has condition $\mathsf{A} \leq a_j$, then $a_i > a_j$.

THEOREM 4.2. *The* $\mathrm{BD1}(\mathcal{F}^{\leq}_{\leftarrow})$ *problem can be solved in* $O(N \log N)$ *time.*

THEOREM 4.3. *The* $\mathrm{BD1}(\mathcal{F}^{\leq}_{+})$ *problem can be solved in* $O(N \log N)$ *time.*

THEOREM 4.4. *The* $\mathrm{BD1}(\mathcal{F}^{\leq}_{\leftarrow+})$ *problem can be solved in* $O(N \log N)$ *time.*

THEOREM 4.5. *The* $\mathrm{BD1}(\mathcal{F}^{\leq}_{\mathsf{aff}})$ *problem can be solved in* $O(N \log N)$ *time.*

4.3 With At-most/At-least Conditions

With *at-most/at-least* ($\mathsf{A} \leq a$ **or** $\mathsf{A} \geq a$) conditions, the arguments from the previous section cannot be directly reused. In fact, the introduction of this new condition type is where we first encounter the hardness boundary for most families of operations.

4.3.1 With Assignment Modifiers

There is still a polynomial time algorithm for the family of operations with the *assignment* modifier, following the reasoning that it is possible to avoid having a tuple selected by both an *at-most* condition and an *at-least* condition.

THEOREM 4.6. *The* $\mathrm{BD1}(\mathcal{F}^{\leq\geq}_{\leftarrow})$ *problem can be solved in* $O(N^2)$ *time.*

4.3.2 With Increment Modifiers

This is the first time we encounter the hardness boundary. Despite the fact that the operations are commutative, we cannot utilize the same techniques as we did for other families of operations.

THEOREM 4.7. *The* $\mathrm{BD1}(\mathcal{F}^{\leq\geq}_{+})$ *problem is* NP*-hard.*

In order to prove Theorem 4.7, we provide a polynomial-time reduction from SUBSETSUM, which is a known NP-hard problem, defined as follows [15].

DEFINITION 4.8 (SUBSETSUM). *The* SUBSETSUM *decision problem is, given a set* $S = \{s_1, \ldots, s_n\}$ *of positive integers, and a positive integer* t*, determine whether there exists a subset* $T \subseteq S$ *such that the sum of all elements in* T *equals* t*.*

Consider an instance of the SUBSETSUM problem with a set $S = \{s_1, \ldots, s_n\}$ of positive integers and a positive integer t. The reduction is as follows: let $s_0 = -t$ and

$$R_{\mathsf{S}} = \{(\mathsf{K} = k, \mathsf{A} = k, \mathsf{B} = 0) \mid k \in \{0, \ldots, n\}\}$$
$$R_{\mathsf{T}} = \{(\mathsf{K} = k, \mathsf{A} = k, \mathsf{B} = b_k) \mid k \in \{0, \ldots, n\}\}$$

where $b_k = \sum_{\ell=0}^{k} s_\ell$. This reduction takes polynomial time. The claim is that it is a positive instance of SUBSETSUM if and only if the best diff between R_{S} and R_{T} under $\mathcal{F}^{\leq\geq}_{+}$ has cost n. We show the correctness of this reduction via a series of lemmas. Throughout this subsection, R_{S} and R_{T} refer to the sets of tuples from the reduction as described here.

LEMMA 4.9. *Operations in* $\mathcal{F}^{\leq\geq}_{+}$ *are commutative.*

PROOF. This follows immediately from commutativity of addition and the fact that attributes in $\mathcal{A}$ never change as a result of an $(\mathcal{A}, \mathcal{B})$-operation. □

LEMMA 4.10. $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{+})$ *is nonempty, and if* F *is its best diff, then* $cost(F) \leq n + 1$.

PROOF. A sequence of operations $F = (f_0, \ldots, f_n)$ where $f_k = (\mathsf{A} \geq k, \mathsf{B} \leftarrow \mathsf{B} + s_k)$ for $k \in \{0, \ldots, n\}$ is a diff between R_{S} and R_{T}, and $cost(F) = n + 1$. □

Next, we establish a few lemmas claiming that there must be best diffs between R_{S} and R_{T} satisfying certain properties.

LEMMA 4.11. $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{+})$ *contains a* bounded *best diff* $F' = (f'_1, \ldots, f'_m)$*, in which for all* $i \in [m]$*,*

$$f'_i = (\mathsf{A} \leq a'_i, \mathsf{B} \leftarrow \mathsf{B} + b'_i) \text{ or}$$
$$f'_i = (\mathsf{A} \geq a'_i, \mathsf{B} \leftarrow \mathsf{B} + b'_i)$$

where a'_i *is an integer in* $\{0, \ldots, n\}$.

PROOF. By Lemma 4.10, $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{+})$ contains a best diff $F = (f_1, \ldots, f_m)$. Entries in the A attribute in R_{S} and R_{T}, by construction, are integers in $\{0, \ldots, n\}$. Define

$$\mathrm{bnd}(a) = \max\{0, \min\{n, a\}\}$$

We construct $F' = (f'_1, \ldots, f'_m)$ from F: for each $i \in [m]$,

- if $f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, then we construct $f'_i = (\mathsf{A} \leq \lfloor \mathrm{bnd}(a_i) \rfloor, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, since $\mathsf{A} \leq a_i$ if and only if $\mathsf{A} \leq \lfloor \mathrm{bnd}(a_i) \rfloor$.
- if $f_i = (\mathsf{A} \geq a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, then we construct $f'_i = (\mathsf{A} \geq \lceil \mathrm{bnd}(a_i) \rceil, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, since $\mathsf{A} \geq a_i$ if and only if $\mathsf{A} \geq \lceil \mathrm{bnd}(a_i) \rceil$.

Thus, F' is a bounded best diff in $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{+})$. □

DEFINITION 4.12 (GAP OPERATION). *A* gap operation *at* k*, where* $k \in [n]$*, is an operation with the condition* $\mathsf{A} \leq k - 1$ *or the condition* $\mathsf{A} \geq k$*.*

LEMMA 4.13. $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{+})$ *contains a* canonical *best diff* F' *where* F' *contains exactly one gap operation at* k*, which must be either*

$$f = (\mathsf{A} \leq k - 1, \mathsf{B} \leftarrow \mathsf{B} - s_k) \text{ or}$$
$$f = (\mathsf{A} \geq k, \mathsf{B} \leftarrow \mathsf{B} + s_k)$$

for every $k \in [n]$*,*

PROOF. Let $F = (f_1, \ldots, f_m)$ be the bounded best diff with the fewest gap operations.

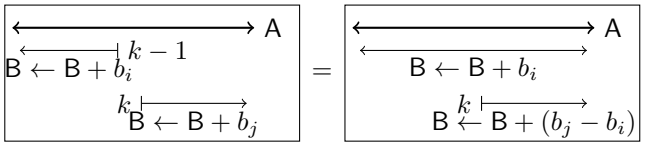
First, we prove that F has at most n gap operations, one at every $k \in [n]$. The proof follows. If F contains two gap operations with the same condition, by Lemma 4.9, they can be reordered and combined, reducing the number of gap operations, a contradiction. If F contains both

$$f_i = (\mathsf{A} \leq k - 1, \mathsf{B} \leftarrow \mathsf{B} + b_i) \text{ and}$$
$$f_j = (\mathsf{A} \geq k, \mathsf{B} \leftarrow \mathsf{B} + b_j)$$

for some $k \in [n]$, then we can replace them with

$$g_1 = (\mathsf{A} \leq n, \mathsf{B} \leftarrow \mathsf{B} + b_i) \text{ and}$$
$$g_2 = (\mathsf{A} \geq k, \mathsf{B} \leftarrow \mathsf{B} + (b_j - b_i))$$

to obtain a best diff with one fewer gap operation (g_1 is not a gap operation), a contradiction.



Second, we prove that F has at least n gap operations, one at every $k \in [n]$. The proof follows. Assume that there is a value $k \in [n]$ such that F has no gap operation at k; that is, all operations in F has neither the condition $\mathsf{A} \leq k-1$ nor the condition $\mathsf{A} \geq k$. One can show by induction on the number of operations performed on R_S that the tuple with $\mathsf{K} = \mathsf{A} = k-1$ and the tuple with $\mathsf{K} = \mathsf{A} = k$ will always have the same value in the B attribute. More precisely, for any $i \in \{0, \ldots, m\}$ and $F_i = (f_1, \ldots, f_i)$, in the relation $F_i(R_\mathsf{S})$, the tuple with $\mathsf{K} = \mathsf{A} = k-1$ and the tuple with $\mathsf{K} = \mathsf{A} = k$ have the same value in the B attribute. However, in R_T those values in the B attribute differ by $s_k \neq 0$ by construction, a contradiction.

Through a similar argument, for each $k \in [n]$, the gap operation at k in F must be either

$$f = (\mathsf{A} \leq k-1, \mathsf{B} \leftarrow \mathsf{B} - s_k) \text{ or}$$
$$f = (\mathsf{A} \geq k, \mathsf{B} \leftarrow \mathsf{B} + s_k)$$

for otherwise the difference between the B values of the tuple with $\mathsf{K} = \mathsf{A} = k-1$ and the tuple with $\mathsf{K} = \mathsf{A} = k$ in $F(R_\mathsf{S})$ will not be s_k, which implies that $F(R_\mathsf{S}) \neq R_\mathsf{T}$. □

LEMMA 4.14. *If F is a best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^{\leq\geq})$, then $cost(F) \geq n$.*

PROOF. This is a corollary of Lemma 4.13. □

LEMMA 4.15. *Best diffs in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^{\leq\geq})$ have cost n if and only if there is a subset $T \subseteq S$ such that the sum of all elements in T equals t.*

PROOF. ($\Leftarrow$) Let T be a subset of S such that the sum of elements in T equals t, then $F = (f_1, \ldots, f_n)$ where, for each $k \in [n]$,

$$f_k = \begin{cases} (\mathsf{A} \leq k-1, \mathsf{B} \leftarrow \mathsf{B} - s_k) & \text{if } s_k \in T \\ (\mathsf{A} \geq k, \mathsf{B} \leftarrow \mathsf{B} + s_k) & \text{otherwise} \end{cases}$$

is a best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^{\leq\geq})$ with cost n.

($\Rightarrow$) By Lemma 4.11 and Lemma 4.13, $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^{\leq\geq})$ contains a canonical best diff $F = (f_1, \ldots, f_n)$ with cost n.

Because the cost is n, F contains exactly one gap operation at k for each $k \in [n]$, as described in Lemma 4.13 and nothing else. By Lemma 4.9 let f_k be the gap operation at k for each $k \in [n]$. If f_k has the condition $\mathsf{A} \geq k$, it does not affect the tuple with $\mathsf{K} = \mathsf{A} = 0$. Let $\{f_{i_1}, \ldots, f_{i_m}\}$ be the subset of $\{f_1, \ldots, f_n\}$ of operations whose conditions are of the form $\mathsf{A} \leq i_\ell - 1$. Thus, in the relation $F(R_\mathsf{S}) = R_\mathsf{T}$, the tuple with $\mathsf{K} = \mathsf{A} = 0$ has the value in attribute B equal to $-\sum_{\ell=1}^{m} s_{i_\ell} = -t$. Thus, $T = \{s_{i_1}, \ldots, s_{i_m}\}$ is a subset of S whose sum of elements is equal to t. □

This proves the correctness of the polynomial-time reduction from SUBSETSUM, which concludes the NP-hardness proof for Theorem 4.7.

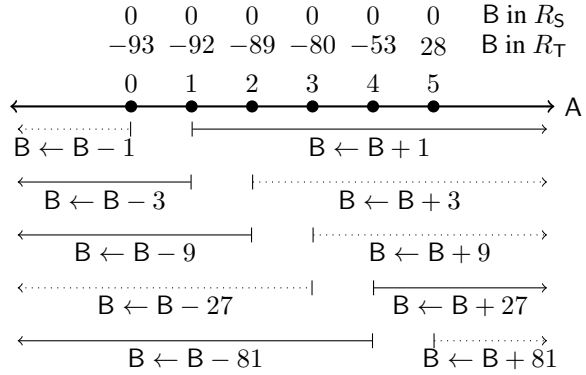


Figure 2: Illustration of Example 4.16

EXAMPLE 4.16. *Consider the* SUBSETSUM *instance with $S = \{1, 3, 9, 27, 81\}$ and $t = 93$. The subset $T = \{3, 9, 81\}$ of S has the sum of its elements equal to t. The reduction gives the following instance of the* BD1($\mathcal{F}_+^{\leq\geq}$) *problem.*

K	A	B
0	0	0
1	1	0
2	2	0
3	3	0
4	4	0
5	5	0

➡

K	A	B
0	0	−93
1	1	−92
2	2	−89
3	3	−80
4	4	−53
5	5	28

Figure 2 shows the two possible gap operations at k for each $k \in [n]$ in their own row. In the third row, for example, one of the two gap operations modifying B *by* 9 *must be used to ensure that the* B *values of the middle tuples differ by* 9 *(in the final relation, between* −89 *and* −80*). In this case, $F = (f_1, f_2, f_3, f_4, f_5)$ where*

$$f_1 = (\mathsf{A} \geq 1, \mathsf{B} \leftarrow \mathsf{B} + 1)$$
$$f_2 = (\mathsf{A} \leq 1, \mathsf{B} \leftarrow \mathsf{B} - 3)$$
$$f_3 = (\mathsf{A} \leq 2, \mathsf{B} \leftarrow \mathsf{B} - 9)$$
$$f_4 = (\mathsf{A} \geq 4, \mathsf{B} \leftarrow \mathsf{B} + 27)$$
$$f_5 = (\mathsf{A} \leq 4, \mathsf{B} \leftarrow \mathsf{B} - 81)$$

is a best diff with cost 5*. The corresponding chosen gap operations are shown in solid lines, while the ones not chosen are shown in dotted lines.*

4.3.3 With Assignment/Increment or Affine Modifiers

With *assignment/increment* or *affine* modifiers, the problem is still NP-hard.

For the *assignment/increment* modifiers, this can be shown via an extension of the proof above for the version with only *increment* modifiers. Essentially, the proof is to show that the

assignment modifier does not provide additional expressivity in the reduction given.

THEOREM 4.17. *The* $\mathrm{BD1}(\mathcal{F}^{\leq\geq}_{\leftarrow+})$ *problem is* NP*-hard.*

We prove the aforementioned theorem via the following lemma.

LEMMA 4.18. *Best diffs in* $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{\leftarrow+})$ *have cost* n *if and only if best diffs in* $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{+})$ *have cost* n*.*

PROOF. ($\Leftarrow$) Any diff in $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{+})$ is also a diff in $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{\leftarrow+})$.

($\Rightarrow$) Let $F = (f_1, \ldots, f_n)$ be a best diff in $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{\leftarrow+})$ of cost n that has the smallest number of assignment modifiers and, among the best diffs with the smallest number of assignment modifiers, has the smallest total length. We show that F has no assignment modifiers.

The proof follows. Assume to the contrary, and let i be the smallest index in $[n]$ such that f_i has an assignment modifier. Suppose $f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow b_i)$. (The proof for when f_i has condition $\mathsf{A} \geq a_i$ is similar.)

Case 1: There is an operation $f_j = (\mathsf{A} \geq a_j, \mathsf{B} \leftarrow \mathsf{B} + b_j)$ where $j < i$ and $a_j \leq a_i$. Then, let $f'_j = (\mathsf{A} \geq a_i + 1, \mathsf{B} \leftarrow \mathsf{B} + b_j)$. If F' is defined as F where f_j is replaced with f'_j, then F' would still yield $F'(R_{\mathsf{S}}) = R_{\mathsf{T}}$, but the total length of F' is smaller than that of F.

Case 2: There is an operation $f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow \mathsf{B} + b_j)$ where $j < i$ and $a_j \leq a_i$. If F' is defined as F where f_j is removed, then F' would still yield $F'(R_{\mathsf{S}}) = R_{\mathsf{T}}$, but the cost of F' is smaller than that of F.

Case 3: None of the above. Then, all tuples matching $\mathsf{A} \leq a_i$ still have the same value in the B attribute, say β, in $F''(R_{\mathsf{S}})$ where $F'' = (f_1, \ldots, f_{i-1})$. Then, let $f'_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow \mathsf{B} + (b_i - \beta))$. If F' is defined as F where f_i is replaced with f'_i, then F' would still yield $F'(R_{\mathsf{S}}) = R_{\mathsf{T}}$, but F' has fewer assignment modifiers than F.

Therefore, F has no assignment modifiers. Thus, F is also a best diff in $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\leq\geq}_{+})$. □

With the *affine* modifier, we again show NP-hardness via a polynomial-time reduction from SUBSETSUM, but the reduction is slightly different from the *increment* case.

THEOREM 4.19. *The* $\mathrm{BD1}(\mathcal{F}^{\leq\geq}_{\mathsf{aff}})$ *problem is* NP*-hard.*

Consider an instance of the SUBSETSUM problem with a set $S = \{s_1, \ldots, s_n\}$ of positive integers and a positive integer t. The reduction is as follows: let $s_0 = -t$ and

$$R_{\mathsf{S}} = \bigcup_{k \in \{0,\ldots,n\}} \{(\mathsf{K} = \mathsf{A} = 99k + i, \mathsf{B} = 0) \mid i \in [99]\}$$

$$R_{\mathsf{T}} = \bigcup_{k \in \{0,\ldots,n\}} \{(\mathsf{K} = \mathsf{A} = 99k + i, \mathsf{B} = b_k) \mid i \in [99]\}$$

where $b_k = \sum_{\ell=0}^{k} s_\ell$. This reduction takes polynomial time. The claim is that it is a positive instance of SUBSETSUM if and only if the best diff between R_{S} and R_{T} under $\mathcal{F}^{\leq\geq}_{\mathsf{aff}}$ has cost n.

The proof is similar to that given for *increment* and *assignment/increment*, and thus only the differences are sketched here. In the reduction, instead of one tuple for each integer in S, a *block* of 99 tuples with the same B value is created. Intuitively, if an operation has a modifier with nonzero slope ($\mathsf{B} \leftarrow b\mathsf{B} + c$ with $b \neq 0$) and it matches multiple tuples in the same block, then it can break the "same B value" requirement within that block. It can take a few operations or one operation with zero slope to fix the block. It can be shown that modifiers with nonzero slope are unnecessary in the best diff in this instance.

4.4 With Range Conditions

With *range* ($\mathsf{A} \in [a, z]$) conditions, the problem is NP-hard for all families of operations of interest, except the one with the *assignment* modifier, similar to the previous case with *at-most/at-least* conditions. The arguments utilize the same core ideas, but are somewhat more complicated.

4.4.1 With Assignment Modifiers

As in cases previously discussed, there is a polynomial time algorithm for the family of operations with the *assignment* modifier. The reasoning is slightly different although the main idea is similar: it is possible to avoid having a tuple selected by two ranges that partially overlap. That is, there is a diff for which any two ranges are either completely disjoint or are such that one is completely contained within the other.

THEOREM 4.20. *The* $\mathrm{BD1}(\mathcal{F}^{\mathsf{R}}_{\leftarrow})$ *problem can be solved in* $O(N^4)$ *time.*

4.4.2 With Increment Modifiers

With *increment* modifiers, like before, the problem is NP-hard. This follows from the same reduction from SUBSET-SUM given in the proof of Theorem 4.7. The proof of the reduction's correctness, however, is somewhat different.

THEOREM 4.21. *The* $\mathrm{BD1}(\mathcal{F}^{\mathsf{R}}_{+})$ *problem is* NP*-hard.*

We prove Theorem 4.21 via a series of lemmas. Throughout this subsection, R_{S} and R_{T} refer to the sets of tuples from the reduction.

LEMMA 4.22. *Operations in* $\mathcal{F}^{\mathsf{R}}_{+}$ *are commutative.*

PROOF. same as in 4.9 □

LEMMA 4.23. $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\mathsf{R}}_{+})$ *is nonempty, and if* F *is its best diff, then* $cost(F) \leq n + 1$*.*

PROOF. A sequence of operations $F = (f_0, \ldots, f_n)$ where $f_k = (\mathsf{A} \in [k, n], \mathsf{B} \leftarrow \mathsf{B} + s_k)$ for $k \in \{0, \ldots, n\}$ is a diff between R_{S} and R_{T}, and $cost(F) = n + 1$. □

LEMMA 4.24. $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\mathsf{R}}_{+})$ *contains a* bounded *best diff* $F' = (f'_1, \ldots, f'_m)$ *in which for all* $i \in [m]$, $f'_i = (\mathsf{A} \in [a'_i, z'_i], \mathsf{B} \leftarrow \mathsf{B} + b'_i)$ *where* a'_i *and* z'_i *are integers in* $\{0, \ldots, n\}$*.*

PROOF. By Lemma 4.23, $\Delta(R_{\mathsf{S}}, R_{\mathsf{T}}, \mathcal{F}^{\mathsf{R}}_{+})$ contains a best diff $F = (f_1, \ldots, f_m)$. Entries in the A attribute in R_{S} and R_{T}, by construction, are integers in $\{0, \ldots, n\}$. Define

$$\mathrm{bnd}(a) = \max\{0, \min\{n, a\}\}$$

We construct $F' = (f'_1, \ldots, f'_m)$ from F: for each $i \in [m]$,

- if $f_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + b_i)$, then we construct $f'_i = (\mathsf{A} \in [\lceil \mathrm{bnd}(a_i) \rceil, \lfloor \mathrm{bnd}(z_i) \rfloor], \mathsf{B} \leftarrow \mathsf{B} + b_i)$, since $\mathsf{A} \in [a_i, z_i]$ if and only if $\mathsf{A} \in [\lceil \mathrm{bnd}(a_i) \rceil, \lfloor \mathrm{bnd}(z_i) \rfloor]$.

Thus, F' is a bounded best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_+)$. □

LEMMA 4.25. *If F is a best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_+)$, then $cost(F) \geq n$.*

PROOF. Define the *jump* of a relation as the number of values $i \in [n]$ such that for tuples $(\mathsf{K} = i-1, \mathsf{A} = i-1, \mathsf{B} = b_{i-1})$ and $(\mathsf{K} = i, \mathsf{A} = i, \mathsf{B} = b_i)$, we have $b_{i-1} < b_i$. Note that the jumps in R_S and R_T are 0 and n, respectively. We prove the following statement by induction: after applying m operations from $\mathcal{F}^\mathsf{R}_+$ to R_S, the jump of the resulting relation is at most m. This implies that at least n operations are required to transform R_S into R_T.

The proof follows. The base case $m = 0$ is trivial. Assume, as an induction hypothesis, that for $m' < m$, applying m' operations to R_S resulting in jump that is at most m'. Let R' be the result of applying $m-1$ operations on R_S, and thus its jump is at most $m-1$. Consider applying $f = (\mathsf{A} \in [a, z], \mathsf{B} \leftarrow \mathsf{B}+b)$ to R' and let $f(R') = R''$. Consider tuples $(\mathsf{K} = i-1, \mathsf{A} = i-1, \mathsf{B} = b_{i-1})$ and $(\mathsf{K} = i, \mathsf{A} = i, \mathsf{B} = b_i)$ in R' where $b_{i-1} \geq b_i$.

- If $i-1$ and i are both not in $[a, z]$, then the B values remain b_{i-1} and b_i respectively, and $b_{i-1} \geq b_i$. This does not contribute to increase in jump.
- If $i-1$ and i are both in $[a, z]$, then the B values become $b_{i-1} + b$ and $b_i + b$ respectively, and $b_{i-1} + b \geq b_i + b$. This does not contribute to increase in jump.
- If $i - 1 < a \leq i$, then the B values become b_{i-1} and $b_i + b$ respectively, and if $b_{i-1} < b_i + b$, then $b > 0$.
- If $i - 1 \leq z < i$, then the B values become $b_{i-1} + b$ and b_i respectively, and if $b_{i-1} + b < b_i$, then $b < 0$.

Thus, jump can only increase by at most 1 depending on the value of b: if $b > 0$, then jump can only increase because of i where $i - 1 < a \leq i$, and if $b < 0$, then jump can only increase because of i where $i - 1 \leq z < i$. □

LEMMA 4.26. *$\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_+)$ contains a bounded best diff $F = (f_1, \ldots, f_m)$ in which there are no two operations*

$$f_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + b_i) \text{ and}$$
$$f_j = (\mathsf{A} \in [a_j, z_j], \mathsf{B} \leftarrow \mathsf{B} + b_j)$$

such that $a_i = a_j$ or $z_i = z_j$.

PROOF. Define a *collision* of F as a pair (i, j) where $i, j \in [m]$ and $i < j$ such that $a_i = a_j$ or $z_i = z_j$.

By Lemma 4.24, let $F = (f_1, \ldots, f_m)$, where $f_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + b_i)$ for all $i \in [m]$, be a bounded best diff with the smallest total length. We show that F contains no collisions.

The proof follows. Assume to the contrary that F has a collision (i, j). Suppose $a_i = a_j$. (The argument for when $z_i = z_j$ is symmetrical.) By commutativity,

$$F' = (f_1, \ldots, f_{i-1}, f_i, f_j, f_{i+1}, \ldots, f_{j-1}, f_{j+1}, \ldots, f_m)$$

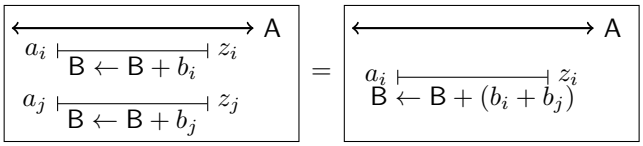
is also a bounded best diff.

Case 1: if $z_i = z_j$ then let

$$g = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + (b_i + b_j))$$

then F'' defined as follows is also a bounded best diff:

$$F'' = (f_1, \ldots, f_{i-1}, g, f_{i+1}, \ldots, f_{j-1}, f_{j+1}, \ldots, f_m)$$



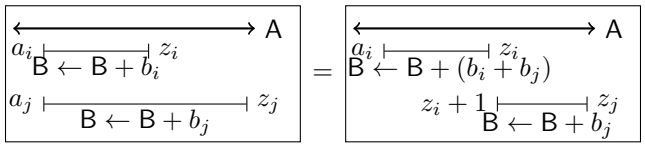
However, F'' has smaller cost than F, contradicting the fact that F is a best diff.

Case 2: if $z_i < z_j$ then let

$$g_1 = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + (b_i + b_j))$$
$$g_2 = (\mathsf{A} \in [z_i + 1, z_j], \mathsf{B} \leftarrow \mathsf{B} + b_j)$$

then F'' defined as follows is also a bounded best diff:

$$F'' = (f_1, \ldots, f_{i-1}, g_1, g_2, f_{i+1}, \ldots, f_{j-1}, f_{j+1}, \ldots, f_m)$$



However, $\ell(F'') = \ell(F') - (z_i - a_i + 1) < \ell(F') = \ell(F)$, contradicting the fact that F' has the smallest total length.

Case 3: if $z_i > z_j$, the proof is similar to Case 2.

Therefore, F has no collisions, and thus $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_+)$ contains a bounded best diff $F = (f_1, \ldots, f_m)$ in which there are no two operations

$$f_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + b_i) \text{ and}$$
$$f_j = (\mathsf{A} \in [a_j, z_j], \mathsf{B} \leftarrow \mathsf{B} + b_j)$$

such that $a_i = a_j$ or $z_i = z_j$. □

LEMMA 4.27. *Best diffs in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_+)$ have cost n if and only if best diffs in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\leq\geq}_+)$ have cost n.*

The idea of the proof is that a diff from one set can be translated into a diff from the other set with the same cost. The full proof is given in Appendix D.

By Lemma 4.15 and Lemma 4.27, the reduction is correct, implying Theorem 4.21.

4.4.3 With Assignment/Increment or Affine Modifiers

With *assignment/increment* or *affine* modifiers, once again, the problem is still NP-hard.

THEOREM 4.28. *The $\mathrm{BD1}(\mathcal{F}^\mathsf{R}_{\leftarrow+})$ problem is NP-hard.*

The proof of the theorem is still based on the same reduction from SUBSETSUM, and follows from the following lemma, the proof of which is given in Appendix D.

LEMMA 4.29. *Best diffs in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_{\leftarrow+})$ have cost n if and only if best diffs in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_+)$ have cost n.*

With the *affine* modifier, NP-hardness can be shown using the same polynomial-time reduction from SUBSETSUM as given for Theorem 4.19.

THEOREM 4.30. *The $\mathrm{BD1}(\mathcal{F}^\mathsf{R}_{\mathsf{aff}})$ problem is NP-hard.*

4.5 With Union-of-Ranges Conditions

With the *union-of-ranges* conditions, the problem becomes NP-hard even with the *assignment* modifier.

THEOREM 4.31. *The* $\mathrm{BD1}(\mathcal{F}^{\mathsf{U}}_{\leftarrow})$ *problem is* NP*-hard.*

In order to prove Theorem 4.31, we provide a polynomial-time reduction from 2SCS (shortest common supersequence of strings of length two), which is a known NP-hard problem, defined as follows [27].

DEFINITION 4.32 (2SCS). *The* 2SCS *decision problem is, given a set* $S = \{s_1, \ldots, s_n\}$ *of strings of length two, and a nonnegative integer* t*, determine whether* S *has a common supersequence of length at most* t*; that is, whether there exists a string* s *of length at most* t *such that for each string* $s_i \in S$*, it is possible to remove some symbols (possibly none) from* s *to obtain* s_i*.*

Note that the alphabet size is not necessarily constant: there can be as many as $2n$ different symbols in a given instance. Also, we assume that each symbol is given in the input represented as a positive integer.

In fact, we will provide a polynomial-time reduction from 2DISTINCTSCS, which is similar to 2SCS with an additional restriction that the two letters in each string in S are not the same. The proof that 2DISTINCTSCS is NP-hard, via a reduction from 2SCS, is given in Appendix C.

Consider an instance of the 2DISTINCTSCS problem with a set $S = \{s_1, \ldots, s_n\}$ of strings of length two and a nonnegative integer t. For each $k \in [n]$, let u_k and v_k be (positive integer representations of) the two symbols of s_k in order. The reduction is as follows: for $k \in [n]$, let

$$t_{5k-3} = (\mathsf{K} = 5k-3, \mathsf{A} = 4k-3, \mathsf{B} = 0)$$
$$t_{5k-2} = (\mathsf{K} = 5k-2, \mathsf{A} = 4k-2, \mathsf{B} = 0)$$
$$t_{5k-1} = (\mathsf{K} = 5k-1, \mathsf{A} = 4k-1, \mathsf{B} = 0)$$
$$t'_{5k-3} = (\mathsf{K} = 5k-3, \mathsf{A} = 4k-3, \mathsf{B} = u_k)$$
$$t'_{5k-2} = (\mathsf{K} = 5k-2, \mathsf{A} = 4k-2, \mathsf{B} = v_k)$$
$$t'_{5k-1} = (\mathsf{K} = 5k-1, \mathsf{A} = 4k-1, \mathsf{B} = u_k)$$

for $k \in \{0, \ldots, n\}$, let

$$t_{5k+0} = t'_{5k+0} = (\mathsf{K} = 5k+0, \mathsf{A} = 4k, \mathsf{B} = -1)$$
$$t_{5k+1} = t'_{5k+1} = (\mathsf{K} = 5k+1, \mathsf{A} = 4k, \mathsf{B} = -2)$$

and let $\kappa_0 = 1$, $\kappa_1 = t + 99$, and

$$R_{\mathsf{S}} = \{t_\ell \mid \ell \in \{0, \ldots, 5n+1\}\}$$
$$R_{\mathsf{T}} = \{t'_\ell \mid \ell \in \{0, \ldots, 5n+1\}\}$$

This reduction takes polynomial time. The claim is that it is a positive instance of 2DISTINCTSCS if and only if the best diff between R_{S} and R_{T} under $\mathcal{F}^{\mathsf{U}}_{\leftarrow}$ has cost at most $t + 2n(t+99)$. We show the correctness of this reduction via a series of lemmas. In this subsection, R_{S} and R_{T} refer to the sets of tuples from the reduction as described here.

First, we define *total range count*, which impacts the cost.

DEFINITION 4.33 (TOTAL RANGE COUNT). *Let* $F = (f_1, \ldots, f_m)$ *where, for* $i \in [m]$*,* $f_i \in \mathcal{F}^{\mathsf{U}}_{\leftarrow}$ *and*

$$f_i = (\mathsf{A} \in \bigcup_{j=1}^{r_i} [a_{ij}, z_{ij}], \mathsf{B} \leftarrow b_i)$$

The total range count *of* F *is defined as* $\sum_{i=1}^{m} r_i$*.*

Next, we establish the special purpose of the tuples of the form $\mathsf{A} = 4k$ in the construction, the proof of which is given in Appendix D.

LEMMA 4.34. *A diff between* R_{S} *and* R_{T} *contains no operation whose condition matches* $\mathsf{A} = 4k$ *for any* $k \in \{0, \ldots, n\}$*.*

These $\mathsf{A} = 4k$ tuples provide "barriers" over which no range condition can cross. Thus, they break the possible A values into partitions. Let partition k, denoted P_k, refers to tuples whose A value is between $4k-4$ and $4k$, exclusive, for $k \in [n]$. There are exactly three tuples in each partition. We say that an operation *affects* a partition if some tuple in that partition is matched by the condition of the operation.

LEMMA 4.35. *A diff between* R_{S} *and* R_{T} *has total range count at least* $2n$*, and each partition has at least two operations that affects it.*

PROOF. Each of the n partitions has two distinct B values in R_{T}, neither of which is 0 as in R_{S}. Thus, two assignment modifiers are required. By Lemma 4.34, a range cannot go across barriers, thus the total range count includes at least two ranges per partition. □

LEMMA 4.36. S *has a common supersequence of length at most* t *iff the best diff between* R_{S} *and* R_{T} *under* $\mathcal{F}^{\mathsf{U}}_{\leftarrow}$ *has cost at most* $t + 2n(t+99)$*.*

PROOF. ($\Leftarrow$) Let $F = (f_1, \ldots, f_m)$ be a best diff between R_{S} and R_{T} under $\mathcal{F}^{\mathsf{U}}_{\leftarrow}$, where

$$f_i = (\mathsf{A} \in \bigcup_{j=1}^{r_i} [a_{ij}, z_{ij}], \mathsf{B} \leftarrow b_i)$$

with cost at most $t + 2n(t+99)$. The total range count of F cannot exceed $2n$, otherwise its cost must be at least $(t+99)(2n+1) > t+2n(t+99)$. Together with Lemma 4.35, the total range count of F must be exactly $2n$. The cost implies that $m \leq t$.

Let $s = b_1 \ldots b_m$. Because each partition contains two distinct B values in R_{T}, and because the total range count must be $2n$, there must be exactly two operations that affects each partition. For $k \in [n]$, partition P_k has two operations f_i and f_j, where $i < j$, that affects it. The operations must be such that $b_i = u_k$ and $b_j = v_k$, where f_i sets the B value for all tuples in P_k to u_k, and f_j then sets the B value for one tuple to v_k. Hence, removing symbols from s except at indices i and j would yield $b_i b_j = u_k v_k = s_k$. Thus, s is a supersequence of S with length $m \leq t$.

($\Rightarrow$) Let $s = w_1 \ldots w_m$ be a common supersequence of S of length $m \leq t$. For each symbol c, let $\mathrm{first}(c)$ be the smallest i such that $w_i = c$; and $\mathrm{last}(c)$, largest.

Construct $F = (f_1, \ldots, f_m)$ where

$$R_i^{(1)} = \bigcup_{\substack{k \in [n] \\ \mathrm{first}(u_k) = i}} [4k-3, 4k-1]$$
$$R_i^{(2)} = \bigcup_{\substack{k \in [n] \\ \mathrm{last}(v_k) = i}} [4k-2, 4k-2]$$
$$f_i = (\mathsf{A} \in R_i^{(1)} \cup R_i^{(2)}, \mathsf{B} \leftarrow w_i)$$

For $k \in [n]$, partition P_k has two operations f_i and f_j that affects it, where $i = \text{first}(u_k)$ and $j = \text{last}(v_k)$. Because s is a supersequence of S, we have $i = \text{first}(u_k) < \text{last}(v_k) = j$. Hence, f_i sets the B value for all tuples in P_k to u_k, and f_j then sets the B value for one tuple to v_k. The total range count of F is $2n$. Thus, F is a diff between R_S and R_T under $\mathcal{F}^{\mathsf{U}}_{\leftarrow}$ whose cost is $m + 2n(t+99) \le t + 2n(t+99)$. □

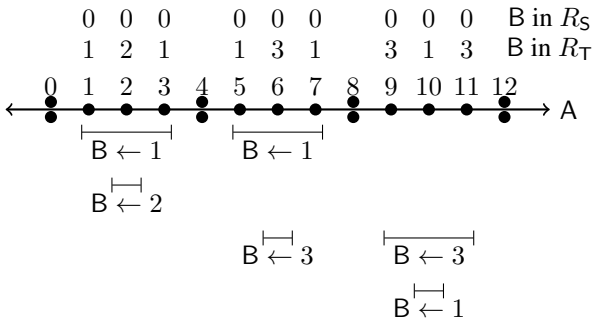


Figure 3: Illustration of Example 4.37

EXAMPLE 4.37. *Consider the* 2DISTINCTSCS *instance with* $S = \{12, 13, 31\}$ *and* $t = 4$. *The string* 1231 *is a supersequence of* S. *The reduction gives the instance of the* $\text{BD1}(\mathcal{F}^{\mathsf{U}}_{\leftarrow})$ *problem shown in Figure 3.*

In this case, $F = (f_1, f_2, f_3, f_4)$ *where*

$$f_1 = (\mathsf{A} \in [1,3] \cup [5,7], \mathsf{B} \leftarrow 1)$$
$$f_2 = (\mathsf{A} \in [2,2], \mathsf{B} \leftarrow 2)$$
$$f_3 = (\mathsf{A} \in [6,6] \cup [9,11], \mathsf{B} \leftarrow 3)$$
$$f_4 = (\mathsf{A} \in [10,10], \mathsf{B} \leftarrow 1)$$

is a best diff with cost $4 + 2 \cdot 13 \cdot 3 = 82$. *Dropping* f_4 *and changing the condition of* f_1 *to* $\mathsf{A} \in [1,3] \cup [5,7] \cup [10,10]$ *yield lower cost but is not permissible, because the tuple at* $\mathsf{A} = 10$ *would have an incorrect* B *value.*

THEOREM 4.38. *The* $\text{BD1}(\mathcal{F}^{\mathsf{U}}_{+})$, $\text{BD1}(\mathcal{F}^{\mathsf{U}}_{\leftarrow+})$, *and* $\text{BD1}(\mathcal{F}^{\mathsf{U}}_{\mathsf{aff}})$ *problems are* NP*-hard.*

PROOF. This follows from polynomial-time reductions from respective range versions, using the same instance and setting $\kappa_0 = 0$ and $\kappa_1 = 1$. □

5. RELAXATION: $\text{BD}\mu$ PROBLEMS

In this section, we discuss a relaxation to the constraints of the "base case" in the previous section. We allow the number of read-only attributes to be more than one. We see in the previous section that even when restricted to 1 read-only attribute, the problem becomes NP-hard even with relatively simple conditions and modifiers. With more attributes, the problem reaches the hardness boundary much more quickly.

The $\text{BD}\mu(\mathcal{F})$ problem is similar to the best diff $\text{BD}(\mathcal{F})$ problem, but constrained to one write-only attribute and no read-write attributes; the number of read-only attributes may vary. Let $\mathcal{A} = \{\mathsf{A}_1, \ldots, \mathsf{A}_\mu\}$ and $\mathcal{B} = \{\mathsf{B}\}$, where $\mathsf{B} \notin \mathcal{A}$.

5.1 With Equality Conditions

Unlike in the previous section with 1 read-only attribute, the problem becomes NP-hard even with *equality* conditions. For the *assignment* case, the problem is closely related to the view synthesis problem, and we derive the hardness result through it. For the remaining cases, we show hardness through reductions from the 1 read-only attribute version with range conditions.

THEOREM 5.1. *The* $\text{BD}\mu(\mathcal{F}^{=}_{\leftarrow})$ *problem is* NP*-hard.*

The proof of this theorem is given in Appendix D.

THEOREM 5.2. *The* $\text{BD}\mu(\mathcal{F}^{=}_{+})$, $\text{BD}\mu(\mathcal{F}^{=}_{\leftarrow+})$, *and* $\text{BD}\mu(\mathcal{F}^{=}_{\mathsf{aff}})$ *problems are* NP*-hard.*

We only show the proof for $\text{BD}\mu(\mathcal{F}^{=}_{+})$, as the remaining proofs are similar. The idea is to simulate range conditions in one attribute with equality conditions in multiple attributes.

Consider the instance $(\mathsf{K}, \mathcal{A}, \mathcal{B}, R_\mathsf{S}, R_\mathsf{T})$ of $\text{BD1}(\mathcal{F}^{\mathsf{R}}_{+})$, where $\mathcal{A} = \{\mathsf{A}\}$, and $a_1 < \cdots < a_\ell$ are values in $V_\mathsf{A}(R_\mathsf{S}, R_\mathsf{T})$ in order. The reduction is as follows: we construct the instance $(\mathsf{K}, \mathcal{A}', \mathcal{B}, R'_\mathsf{S}, R'_\mathsf{T})$. Here, $\mathcal{A}' = \{\mathsf{A}_1, \ldots, \mathsf{A}_{2\ell}\}$, and R'_S and R'_T are identical to R_S and R_T, respectively, except that the attribute $\mathsf{A} = a$ is replaced by $\mathsf{A}_1, \ldots, \mathsf{A}_{2\ell}$ in the following fashion.

Let $a^* = a_\ell + 1 \notin V_\mathsf{A}(R_\mathsf{S}, R_\mathsf{T})$. Define $\rho_i = [a_i, a_\ell]$ and $\rho_{\ell+i} = [a_1, a_i]$ for $i \in [\ell]$. For each tuple, its A_i value is assigned as its A value, except when the value is in the range ρ_i, in which case it is assigned to a^*, for $i \in [2\ell]$.

LEMMA 5.3. *Best diffs in* $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\mathsf{R}}_{+})$ *have cost* n *(in* BD1*) if and only if best diffs in* $\Delta(R'_\mathsf{S}, R'_\mathsf{T}, \mathcal{F}^{\le}_{+})$ *have cost* n *(in* $\text{BD}\mu$*).*

PROOF. The following pair of conditions are equivalent, matching corresponding tuples in the respective problems.

$$\mathsf{A} \in [a_i, a_j] \equiv \mathsf{A}_i = a^* \wedge \mathsf{A}_{\ell+j} = a^*$$
$$\mathsf{A} \in [a_1, a_j] \equiv \mathsf{A}_{\ell+j} = a^*$$
$$\mathsf{A} \in [a_i, a_\ell] \equiv \mathsf{A}_i = a^*$$

where $i, j \in [\ell]$, and

$$\mathsf{A} \in [a_j, a_j] \equiv \mathsf{A}_i = a_j$$

where $i, j \in [2\ell]$. Thus, the best diffs translate from one problem to the other. □

5.2 With At-most Conditions

The classification for the problem with the *assignment* modifier is unknown. The cases with the *increment*, *assignment/increment*, and *affine* modifiers are NP-hard, even if we restrict the number of read-only attributes to 2.

THEOREM 5.4. *The* $\text{BD}\mu(\mathcal{F}^{\le}_{+})$, $\text{BD}\mu(\mathcal{F}^{\le}_{\leftarrow+})$, *and* $\text{BD}\mu(\mathcal{F}^{\le}_{\mathsf{aff}})$ *problems are* NP*-hard, even with* 2 *read-only attributes.*

We only show the proof for $\text{BD}\mu(\mathcal{F}^{\le}_{+})$, as the remaining proofs are similar. The idea is to simulate range conditions in one attribute with at-most conditions in two attributes.

Consider the instance $(\mathsf{K}, \mathcal{A}, \mathcal{B}, R_\mathsf{S}, R_\mathsf{T})$ of $\text{BD1}(\mathcal{F}^{\mathsf{R}}_{+})$, where $\mathcal{A} = \{\mathsf{A}\}$. The reduction is as follows: we construct the instance $(\mathsf{K}, \mathcal{A}', \mathcal{B}, R'_\mathsf{S}, R'_\mathsf{T})$. Here, $\mathcal{A}' = \{\mathsf{A}_1, \mathsf{A}_2\}$, and R'_S and R'_T are identical to R_S and R_T, respectively, except that the attribute $\mathsf{A} = a$ is replaced by $\mathsf{A}_1 = a$ and $\mathsf{A}_2 = -a$.

LEMMA 5.5. *Best diffs in* $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_+)$ *have cost* n *(in* BD1*) if and only if best diffs in* $\Delta(R'_\mathsf{S}, R'_\mathsf{T}, \mathcal{F}^{\leq}_+)$ *have cost* n *(in* BDμ*).*

PROOF. The following pair of conditions are equivalent, matching corresponding tuples in the respective problems.

$$\mathsf{A} \in [a, z] \equiv \mathsf{A}_1 \leq z \wedge \mathsf{A}_2 \leq -a$$
$$\mathsf{A} \in [v^\mathsf{A}_{\min}, z] \equiv \mathsf{A}_1 \leq z$$
$$\mathsf{A} \in [a, v^\mathsf{A}_{\max}] \equiv \mathsf{A}_2 \leq -a$$

Thus, the best diffs translate from one problem to the other. □

5.3 With At-most/At-least, Range, or Union-of-Ranges Conditions

With *at-most/at-least*, *range*, or *union-of-ranges* conditions, the problem is NP-hard when using *increment*, *assignment/increment*, or *affine* modifiers, via trivial reductions.

THEOREM 5.6. *The* BD$\mu(\mathcal{F}^\phi_\omega)$ *problem, for*

$$\phi \in \{\leq \geq, \mathsf{R}, \mathsf{U}\} \text{ and}$$
$$\omega \in \{+, \leftarrow +, \mathsf{aff}\}$$

is NP*-hard, even with* 1 *read-only attribute.*

PROOF. These are generalizations from their BD1 counterparts, which are all NP-hard. □

The case with the *assignment* modifier is different. The classification for the problem with the *at-most/at-least* condition is unknown. While there is a version of the view synthesis problem that is similar to the *range* case, we provide a proof of NP-hardness via a different problem. The proof works even when we restrict the number of read-only attributes to 2.

THEOREM 5.7. *The* BD$\mu(\mathcal{F}^\mathsf{R}_\leftarrow)$ *problem is* NP*-hard, even with* 2 *read-only attributes.*

In order to prove Theorem 5.7, we provide a polynomial-time reduction from RECTANGLECOVER, which is a known NP-hard problem, defined as follows [12].

DEFINITION 5.8 (RECTANGLECOVER). *The* RECTANGLECOVER *decision problem is, given an orthogonal polygon* P *(on a plane) with* n *vertices, and a nonnegative integer* t*, determine whether there is a rectangle cover of* P *of size* t*; that is, whether there exists a set of* t *axis-aligned rectangles whose union is exactly* P*.*

PROOF. Consider an instance of the RECTANGLECOVER problem with an orthogonal polygon P with n vertices. Without loss of generality, let $[\ell]$ be the set of coordinates used by P, where $\ell \leq n$. (Essentially we perform a "rank-space reduction" [3], since stretching the polygon does not affect the size of the cover.) Construct an $\ell \times \ell$ grid and superimpose the polygon P on it.

We create 99 tuples for each of the $\ell \times \ell$ grid cells, with their A_1 and A_2 values corresponding to their x and y coordinates. Their B values are set as follows: in R_S, set all B values to distinct positive values; in R_T, set B to the same values as in R_S, except when the following condition applies: for the tuple with $\mathsf{A}_1 = x$ and $\mathsf{A}_2 = y$, the square with opposite corners (x, y) and $(x+1, y+1)$ is contained in (the superimposed) P; in which case B is set to 0.

The claim is that P has a rectangle cover of size t if and only if R_S and R_T has a diff under $\mathcal{F}^\mathsf{R}_\leftarrow$ of cost t. This is because using a rectangle with opposite corners (x_1, y_1) and (x_2, y_2), where $x_1 < x_2$ and $y_1 < y_2$, corresponds to setting $\mathsf{B} \leftarrow 0$ to the tuples matching the condition $\mathsf{A}_1 \in [x_1, x_2 - 1] \wedge \mathsf{A}_2 \in [y_1, y_2 - 1]$. To see why the off-by-one correction is needed, consider the case $x_1 = x_2$. The range $[x_1, x_2]$ is not empty (contains one element), but the rectangle defined by those x-coordinates have zero width.

The reason we need multiple tuples per grid cell is to prevent "unsetting" the B value from 0. Once the B values of these 99 tuples are set to 0 (or any value), they cannot be changed back into distinct B values again using assignment modifier.

Thus, the problem is NP-hard. □

THEOREM 5.9. *The* BD$\mu(\mathcal{F}^\mathsf{U}_\leftarrow)$ *problem is* NP*-hard, even with* 2 *read-only attributes.*

PROOF. This is a corollary of Theorem 5.7. □

6. CONCLUSIONS AND FUTURE WORK

This paper introduces the family of DATA-DIFF problems characterized by a particular set of modifiers and conditions of interest. It identifies the base case of 1 read-only and 1 write-only attribute and fully classifies the complexity across families of operations (Table 1). It also discusses the generalization to multiple read-only attributes, showing NP-hardness in most families of operations (Table 2).

Some remaining open problems are discussed earlier, particularly characterizing BD$\mu(\mathcal{F}^{\leq}_\leftarrow)$ and BD$\mu(\mathcal{F}^{\leq\geq}_\leftarrow)$. In addition, we have only discussed the settings with 1 write-only attribute and 0 read-write attributes. In particular, introducing read-write attributes creates a complexity where an operation may modify the values in the attributes used for conditions, and therefore the same condition may match a different set of tuples depending on when it is used, making the order of operations even more crucial. Characterizing the problem under relaxations of these constraints is therefore an interesting venue for further investigation.

Acknowledgements

We would like to thank Jeff Erickson for initial discussions; we would also like to thank Liqi Xu and Sheng Shen for practical implementations of DATA-DIFF.

7. REFERENCES

[1] A. Abouzied, D. Angluin, C. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 49–60. ACM, 2013.

[2] B. Alexe, B. Ten Cate, P. G. Kolaitis, and W.-C. Tan. Designing and refining schema mappings via data examples. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 133–144. ACM, 2011.

[3] S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 198–207, 2000.

[4] D. W. Barowy, S. Gulwani, T. Hart, and B. Zorn. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In *ACM SIGPLAN Notices*, volume 50, pages 218–228. ACM, 2015.

[5] A. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, and A. G. Parameswaran. Datahub: Collaborative data science & dataset version management at scale. *CIDR*, 2015.

[6] S. Bhattacherjee, A. Chavan, S. Huang, A. Deshpande, and A. Parameswaran. Principles of dataset versioning: Exploring the recreation/storage tradeoff. *Proceedings of the VLDB Endowment*, 8(12):1346–1357, 2015.

[7] A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases. In *18th International Conference on Extending Database Technology (EDBT)*, 2015.

[8] A. Bonifati, R. Ciucanu, A. Lemay, and S. Staworko. A paradigm for learning queries on big data. In *Proceedings of the First International Workshop on Bringing the Value of Big Data to Users (Data4U 2014)*, page 7. ACM, 2014.

[9] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive join query inference with jim. *Proceedings of the VLDB Endowment*, 7(13):1541–1544, 2014.

[10] A. Bonifati, R. Ciucanu, and S. Staworko. Learning join queries from user examples. *ACM Transactions on Database Systems (TODS)*, 40(4):24, 2016.

[11] B. T. Cate, V. Dalmau, and P. G. Kolaitis. Learning schema mappings. *ACM Transactions on Database Systems (TODS)*, 38(4):28, 2013.

[12] J. Culberson and R. Reckhow. Covering polygons is hard. *Journal of Algorithms*, 17(1):2 – 44, 1994.

[13] A. Das Sarma, A. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *Proceedings of the 13th International Conference on Database Theory*, pages 89–103. ACM, 2010.

[14] G. H. Fletcher, M. Gyssens, J. Paredaens, and D. Van Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *IEEE Transactions on Knowledge and Data Engineering*, 21(6):939–942, 2009.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[16] G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *Journal of the ACM (JACM)*, 57(2):6, 2010.

[17] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *POPL*, pages 317–330, 2011.

[18] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, 2012.

[19] W. R. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *ACM SIGPLAN Notices*, volume 46, pages 317–328. ACM, 2011.

[20] D. Kini and S. Gulwani. Flashnormalize: Programming by examples for text normalization. In *IJCAI*, pages 776–783, 2015.

[21] E. Lu, R. Bodik, and B. Hartmann. Quicksilver: Automatic synthesis of relational queries. Technical report, Tech. Rep. UCB/EECS-2013-68, UC-Berkeley, 2013.

[22] M. Maddox, D. Goehring, A. J. Elmore, S. Madden, A. Parameswaran, and A. Deshpande. Decibel: The relational dataset branching system. *Proceedings of the VLDB Endowment*, 9(9):624–635, 2016.

[23] K. Panev and S. Michel. Reverse engineering top-k database queries with paleo. In *EDBT*, pages 113–124, 2016.

[24] R. Singh and S. Gulwani. Learning Semantic String Transformations from Examples. *PVLDB*, 5(8):740–751, 2012.

[25] R. Singh and S. Gulwani. Synthesizing number transformations from input-output examples. In *International Conference on Computer Aided Verification*, pages 634–651. Springer, 2012.

[26] R. Singh and S. Gulwani. Transforming spreadsheet data types using examples. In *ACM SIGPLAN Notices*, volume 51, pages 343–356. ACM, 2016.

[27] V. Timkovskii. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics and Systems Analysis*, 25(5):565–580, 1989.

[28] Q. T. Tran, C. Y. Chan, and S. Parthasarathy. Query by output. In *Proc. of ACM SIGMOD*, 2009.

[29] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query reverse engineering. *The VLDB Journal*, 23(5):721–746, 2014.

[30] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. Reverse engineering complex join queries. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 809–820. ACM, 2013.

[31] S. Zhang and Y. Sun. Automatically synthesizing sql queries from input-output examples. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 224–234. IEEE, 2013.

APPENDIX

A. POLYNOMIAL-TIME RESULTS

In this section, we discuss cases of the DATA-DIFF problem with polynomial time algorithms in more detail.

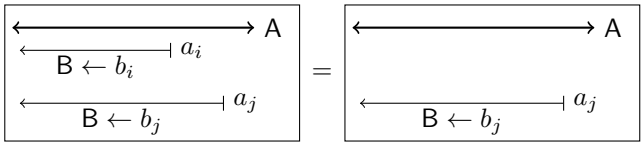
PROOF OF THEOREM 4.1 ($\mathcal{F}^{=}_{\leftarrow}$, $\mathcal{F}^{=}_{+}$, $\mathcal{F}^{=}_{\leftarrow+}$, $\mathcal{F}^{=}_{\mathsf{aff}}$). Each operation affects the value of all tuples with one value of A, and there is not a reason to apply two operations for the same value of A. Therefore, one simply needs to sort the tuples by their A value, iterate through all values of A, and create an appropriate operation to modify the B value to the right value, if possible. □

PROOF OF THEOREM 4.2 ($\mathcal{F}^{\leq}_{\leftarrow}$). We first consider the following proposition: if $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\leq}_{\leftarrow})$ is nonempty, then it contains a best diff $F = (f_1, \ldots, f_m)$ in which for all $i, j \in [m]$, if $i < j$ and

$$f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow b_i) \text{ and}$$
$$f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow b_j)$$

then $a_i > a_j$.

The proof follows. Suppose $a_i \leq a_j$, then the diff F' constructed by removing f_i from F achieves the same result—that is, $F'(R_\mathsf{S}) = F(R_\mathsf{S}) = R_\mathsf{T}$—because changes caused by f_i are rendered moot by f_j, and thus F is not a best diff.



Therefore, one simply needs to sort the tuples by their A, and in decreasing order of A, create an appropriate operation to modify the B value to the right value, if possible. □

PROOF OF THEOREM 4.3 ($\mathcal{F}^{\leq}_{+}$). We first consider the following proposition: if $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\leq}_{+})$ is nonempty, then it contains a best diff $F = (f_1, \ldots, f_m)$ in which for all $i, j \in [m]$, if $i < j$ and

$$f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i) \text{ and}$$
$$f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow \mathsf{B} + b_j)$$

then $a_i > a_j$. The correctness of the proposition follows from the fact that the operations are commutative.

Therefore, one simply needs to sort the tuples in decreasing order of A, create an appropriate operation to modify the B value to the right value, if possible. □

PROOF OF THEOREM 4.4 ($\mathcal{F}^{\leq}_{\leftarrow+}$). We first consider the following proposition: if $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\leq}_{\leftarrow+})$ is nonempty, then it contains a best diff $F = (f_1, \ldots, f_m)$ in which for all $i, j \in [m]$, if $i < j$ and one of the following is true:

(a) $f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow b_i)$, $f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow b_j)$
(b) $f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, $f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow b_j)$
(c) $f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, $f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow \mathsf{B} + b_j)$
(d) $f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow b_i)$, $f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow \mathsf{B} + b_j)$

then $a_i > a_j$. Equivalently, we define an *inversion* as a pair (i, j) such that the preconditions hold but instead $a_i \leq a_j$, and we claim that there exists a best diff without inversions.

The proof follows. For cases (a) and (b), the proof is the same as in Theorem 4.2. For cases (c) and (d), the proof is as follows. Here, an inversion, as defined above, is a pair $(i, j) \in [m]$ such that $i < j$, the condition for f_i is $\mathsf{A} \leq a_i$, the condition for f_j is $\mathsf{A} \leq a_j$, and $a_i > a_j$.

If $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\leq}_{+})$ is nonempty, then it contains a best diff $F = (f_1, \ldots, f_m)$ that does not violate (a), or (b), with the fewest inversions. We show that F has zero inversions.

Suppose F has an inversion of type (c) or type (d), that is, there are $i, j \in [m]$ such that $i < j$ and either

$$f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i) \text{ and}$$
$$f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow \mathsf{B} + b_j)$$

(for an inversion of type (c))

$$f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow b_i) \text{ and}$$
$$f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow \mathsf{B} + b_j)$$

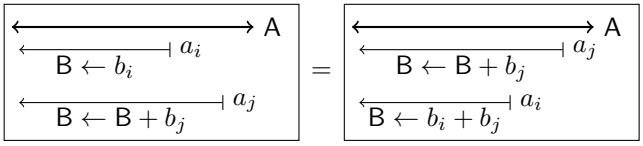
(for an inversion of type (d)), and $a_i \leq a_j$. Without loss of generality, let (i, j) be such a pair where the index difference $j - i$ is the smallest. It must be the case that $j - i = 1$, for otherwise f_k where $i < k < j$ will create a violation for (a) or (b).

If (f_i, f_j) constitutes an inversion of type (c), we may create a new diff F' is equivalent to F, but with f_i and f_j switched. $F'(R_\mathsf{S}) = F(R_\mathsf{S}) = R_\mathsf{T}$ since the increment updates are commutative (and there are no operations f_k, $i < k < j$ with assignment updates to break said commutativity since $j - i = 1$).

Otherwise, (f_i, f_j) constitutes an inversion of type (d). In this case, let $F' = (f_1, \ldots, f_{i-1}, f_j, g, f_{j+1}, \ldots, f_m)$ where

$$g = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow b_i + b_j)$$

then $F'(R_\mathsf{S}) = F(R_\mathsf{S}) = R_\mathsf{T}$. In either case (inversion of type (c) or of type (d)), F' has one fewer inversion than F, a contradiction to the fact that F has the fewest inversions.



Therefore, one simply needs to sort the tuples by their A, and compute the smallest number of operations required using dynamic programming. □

PROOF OF THEOREM 4.5 ($\mathcal{F}^{\leq}_{\mathsf{aff}}$). We first consider the following proposition: if $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\leq}_{\mathsf{aff}})$ is nonempty, then it contains a best diff $F = (f_1, \ldots, f_m)$ in which for all $i, j \in [m]$, if $i < j$ and

$$f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow b_i\mathsf{B} + c_i) \text{ and}$$
$$f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow b_j\mathsf{B} + c_j)$$

then $a_i > a_j$. Equivalently, we define an *inversion* as a pair (i, j) such that the preconditions hold but instead $a_i \leq a_j$, and we claim that there exists a best diff without inversions.

The proof follows. Suppose F has an inversion, that is, there are $i, j \in [m]$ such that $i < j$ and

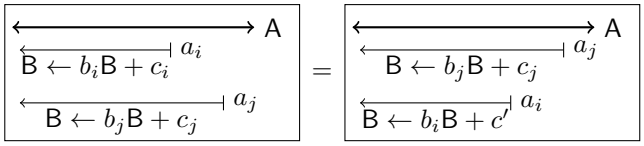
$$f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow b_i\mathsf{B} + c_i) \text{ and}$$
$$f_j = (\mathsf{A} \leq a_j, \mathsf{B} \leftarrow b_j\mathsf{B} + c_j)$$

and $a_i \le a_j$. Without loss of generality, let (i, j) be such a pair where the index difference $j - i$ is the smallest. It must be the case that $j - i = 1$, for otherwise f_k where $i < k < j$ will be such that either (i, k) or (k, j) is an inversion with a smaller index difference.

Let $F' = (f_1, \ldots, f_{i-1}, f_j, g, f_{j+1}, \ldots, f_m)$ where

$$g = (\mathsf{A} \le a_i, \mathsf{B} \leftarrow b_i\mathsf{B} + c') \text{ and}$$
$$c' = c_j + b_j c_i - c_j b_i$$

then $F'(R_\mathsf{S}) = F(R_\mathsf{S}) = R_\mathsf{T}$, but F' has one fewer inversion than F, a contradiction to the fact that F has the fewest inversions.



Label the tuples $1, \ldots, n$ in order of increasing A. We can use dynamic programming to compute $f(m)$, the cost of modifying tuples 1 throught m in order to match R_T. The process effectively segments the tuples into blocks, each of which containing tuples that can be transformed together using one affine transformation, i.e., lie on the same "line", taking extra care of constant transformations—those with the modifier $\mathsf{B} \leftarrow b\mathsf{B} + c$ with $b = 0$.

The final answer $f(n)$ can be computed in $O(N \log N)$. □

PROOF OF THEOREM 4.6 ($\mathcal{F}^{\le\ge}_{\leftarrow}$). We first consider the following proposition: if $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\le\ge}_{\leftarrow})$ is nonempty, then it contains a best diff $F = (f_1, \ldots, f_n)$ in which there are no operations

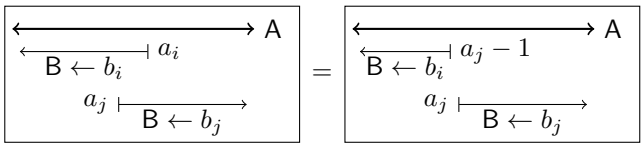
$$f_i = (\mathsf{A} \le a_i, \mathsf{B} \leftarrow b_i) \text{ and}$$
$$f_j = (\mathsf{A} \ge a_j, \mathsf{B} \leftarrow b_j)$$

such that $a_i \ge a_j$. In other words, F contains no two "overlapping" operations.

The proof follows. Let $F = (f_1, \ldots, f_m)$ be a best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\le\ge}_{\leftarrow})$ with the smallest total length. Assume to the contrary that there exists operations f_i and f_j such that

$$f_i = (\mathsf{A} \le a_i, \mathsf{B} \leftarrow b_i) \text{ and}$$
$$f_j = (\mathsf{A} \ge a_j, \mathsf{B} \leftarrow b_j)$$

and $a_i \ge a_j$. If $i < j$, then let $g = (\mathsf{A} \le a_j - 1, \mathsf{B} \leftarrow b_i)$, and $F' = (f_1, \ldots, f_{i-1}, g, f_{i+1}, \ldots, f_m)$. Then, $F'(R_\mathsf{S}) = F(R_\mathsf{S}) = R_\mathsf{T}$, but F' has smaller total length than F.



Otherwise, if $i > j$, then let $g = (\mathsf{A} \ge a_i + 1, \mathsf{B} \leftarrow b_j)$, and $F' = (f_1, \ldots, f_{j-1}, g, f_{j+1}, \ldots, f_m)$. Then, $F'(R_\mathsf{S}) = F(R_\mathsf{S}) = R_\mathsf{T}$, but F' has smaller total length than F. In either case, it contradicts with the fact that F has the smallest total length.

Therefore, one simply needs to sort the tuples by their A, decide on the "breakpoint" that separates the $\mathsf{A} \le a$ conditions from the $\mathsf{A} \ge a$ conditions, then use the algorithm similar to the one given in Theorem 4.2 on each side. □

PROOF OF THEOREM 4.20 ($\mathcal{F}^{\mathsf{R}}_{\leftarrow}$). If $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^{\mathsf{R}}_{\leftarrow})$ is nonempty, then it contains a best diff $F = (f_1, \ldots, f_n)$ in which for any two operations $f_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow b_i)$ and $f_j = (\mathsf{A} \in [a_j, z_j], \mathsf{B} \leftarrow b_j)$ where $i < j$, either $z_i < a_j$ or $z_j < a_i$ or $a_i \le a_j \le z_j \le z_i$. The proof is similar to that given in Theorem 4.6.

Label the tuples $1, \ldots, n$ in order of increasing A. We can use dynamic programming to compute $f(m_1, m_2, \delta)$, the cost of modifying tuples m_1 through m_2 in order to match R_T where all tuples have B values set to δ, unless δ is NULL in which case all tuples have the original B values like in R_S.

The final answer $f(1, n, \text{NULL})$ can be computed in $O(N^4)$. □

B. APPROXIMATION RESULTS

For some NP-hard cases of the DATA-DIFF problem, we are able to provide polynomial-time approximation algorithms. In fact, these algorithms are discussed earlier, since they provide exact results for different condition and modifier settings.

B.1 Approximation for $\mathrm{BD1}(\mathcal{F}^{\le\ge}_{+})$

THEOREM B.1. *For the* $\mathrm{BD1}(\mathcal{F}^{\le\ge}_{+})$ *problem an additive* 1*-approximation can be found in* $O(N \log N)$ *time.*

PROOF. In short, we show that if we only use the one of the "at most" and "at least" condition type exclusively, and the cost of the best diff only increases by at most one.

Let $F = (f_1, \ldots, f_m)$ be the best diff between R_S and R_T, such that $I_\le$ is the set of indices $i \in [m]$ where f_i is in the form $(\mathsf{A} \le a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, and $I_\ge$ is the set of indices $i \in [m]$ where f_i is in the form $(\mathsf{A} \ge a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$. Then, $F' = (f'_1, \ldots, f'_m, g)$ where, for $i \in [m]$,

$$f'_i = \begin{cases} (\mathsf{A} \le a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i) & \text{if } i \in I_\le \\ (\mathsf{A} \le a_i - 1, \mathsf{B} \leftarrow \mathsf{B} - b_i) & \text{if } i \in I_\ge \end{cases}$$
$$g = (\mathsf{A} \le v^{\max}_\mathsf{A}, \mathsf{B} \leftarrow \mathsf{B} + \sum_{i \in I_\ge} b_i)$$

is also a diff between R_S and R_T. By Theorem 4.3, a diff better or as good as F' can be found in $O(N \log N)$. □

B.2 Approximation for $\mathrm{BD1}(\mathcal{F}^{\mathsf{R}}_{+})$

THEOREM B.2. *For the* $\mathrm{BD1}(\mathcal{F}^{\mathsf{R}}_{+})$ *problem, a multiplicative* 2*-approximation can be found in* $O(N \log N)$ *time.*

PROOF. Let $F = (f_1, \ldots, f_m)$ be the best diff between R_S and R_T, where $f_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + b_i)$ for $i \in [m]$. Then, $F' = (f'_1, \ldots, f'_{2m})$ where, for $i \in [m]$,

$$f'_{2i-1} = (\mathsf{A} \le a_i - 1, \mathsf{B} \leftarrow \mathsf{B} - b_i)$$
$$f'_{2i} = (\mathsf{A} \le z_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$$

is also a diff between R_S and R_T. By Theorem 4.3, a diff better or as good as F' can be found in $O(N \log N)$. □

As a side note, there is the following reduction to the edge-cost flow problem. Construct a flow network where each vertex corresponds to a tuple. Let $v_1, \ldots, v_n$ be vertices corresponding to tuples in increasing order of A. Construct an edge between every pair of vertices. Assume B of a vertex

changes from b to b'. If $b < b'$, then let the supply of that vertex be $b' - b$. If $b > b'$, then let the demand of that vertex be $b - b'$.

The claim is that there is that there is a diff of cost m if and only if there is a flow of cost m. Thus, any approximation scheme for edge-cost flow can also be used for BD.

C. THE 2DISTINCTSCS PROBLEM

We prove that 2DISTINCTSCS is NP-hard, via a polynomial-time reduction from 2SCS.

Consider an instance of 2SCS with set S of strings of length two and a nonnegative integer t. Let C be the set of symbols c where cc is in S.

For each symbol $c \in C$, create two new symbols c_1 and c_2. Let $S' = \{f_1(u)f_2(v) \mid uv \in S\}$ where, for $i \in [2]$,

$$f_i(c) = \begin{cases} c_i & \text{if } c \in C \\ c & \text{otherwise} \end{cases}$$

THEOREM C.1. *S has a supersequence of length at most t iff S' has a supersequence of length at most t.*

PROOF. ($\Leftarrow$) Let s' be a supersequence of S' of length at most t. Changing c_1 and c_2 to c for $c \in C$ from s' and S' preserves the supersequence constraint: characters at the same indices can be removed from s' (or s) to obtain each string in S' (or S).

($\Rightarrow$) Let $s = w_1 \dots w_m$ be a supersequence of S of length $m \leq t$. Let s' be identical to s, with the following change: for each $c \in C$, change its first occurrence in s to c_1 and its last occurrence in s to c_2. By definition, each symbol in c must have at least two occurrences in s, so there is no conflict.

Consider $uv \in S$. Let $i, j \in [m]$ be indices such that $i < j$ and $w_i = u$ and $w_j = v$. The first occurence of u in s is at index $i' \leq i$ and the last occurence of v in s is at index $j' \geq j$. Thus, removing symbols other than at indices i' and j' from s' would give $w_{i'}w_{j'} = f_1(u)f_2(v)$. Therefore, s' is a supersequence of S' of length at most t. □

Therefore 2DISTINCTSCS is NP-hard.

D. ADDITIONAL PROOFS

PROOF OF THEOREM 4.27. ($\Leftarrow$) By Lemma 4.11, let $F = (f_1, \dots, f_n)$ be a bounded best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^{\leq\geq})$ of cost n. Entries in the A attribute in R_S and R_T are integers in $\{0, \dots, n\}$. We can construct F' from F as follows.

- If $f_i = (\mathsf{A} \leq a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, then we construct $f_i' = (\mathsf{A} \in [0, a_i], \mathsf{B} \leftarrow \mathsf{B} + b_i)$, since $\mathsf{A} \leq a_i$ if and only if $\mathsf{A} \in [0, a_i]$.
- If $f_i = (\mathsf{A} \geq a_i, \mathsf{B} \leftarrow \mathsf{B} + b_i)$, then we construct $f_i' = (\mathsf{A} \in [a_i, n], \mathsf{B} \leftarrow \mathsf{B} + b_i)$, since $\mathsf{A} \geq a_i$ if and only if $\mathsf{A} \in [a_i, n]$.

Hence, $F' = (f_1', \dots, f_n')$ is a best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^\mathsf{R})$.

($\Rightarrow$) By Lemma 4.26, let $F = (f_1, \dots, f_n)$, where $f_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + b_i)$ for $i \in [n]$, be a bounded best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^\mathsf{R})$ of cost n such that $|C(F)| = 0$. Consider a directed graph $G = (V, E)$ where $V = \{0, \dots, n+1\}$ and $E = \{(a_i, z_i + 1) \mid i \in [m]\}$. That is, for each operation f_i, there is a corresponding edge $(a_i, z_i + 1)$ in E.

From the construction of G, vertex 0 has in-degree 0, and vertex $n + 1$ has out-degree 0. In addition, any vertex in V cannot have in-degree greater than 1. Assume the contrary: $\exists i, j$ s.t. $z_i + 1 = z_j + 1 = k$ then $z_i = z_j$. Likewise, any vertex in V cannot have out-degree greater than 1. A directed graph whose maximum in-degree and out-degree is 1 can be decomposed into vertex-disjoint paths and cycles. However, E only contains edges (u, v) such that $u < v$, so G does not contain cycles. Thus, G can be decomposed into vertex-disjoint paths.

Let $P = (v_1, \dots, v_p)$ be a path in the vertex-disjoint path decomposition of G. We prove that $v_1 = 0$ or $v_p = n + 1$. Assume for contradiction that $v_1 \neq 0$ and $v_p \neq n + 1$. From the degree requirements, we also have $v_1 \neq n + 1$ and $v_p \neq 0$. For each $k \in \{1, \dots, p-1\}$, let $f_k^P = (\mathsf{A} \in [v_k, v_{k+1} - 1], \mathsf{B} \leftarrow \mathsf{B} + b_k^P)$ be an operation from $\{f_1, \dots, f_n\}$ corresponding to the edge (v_k, v_{k+1}).

Note that f_{k-1}^P and f_k^P are the only two operations in $\{f_1, \dots, f_n\}$ that can affect the difference in the B attribute between the tuples at $\mathsf{K} = \mathsf{A} = v_k - 1$ and at $\mathsf{K} = \mathsf{A} = v_k$. In particular, it must be the case that $b_1^P = s_{v_1} > 0$ and $-b_{p-1}^P = s_{v_p - 1} > 0$ and $b_k^P - b_{k-1}^P = s_{v_k} > 0$ in order for $F(R_\mathsf{S})$ to agree with R_T. However, these facts imply $0 < b_1^P < b_2^P < \dots < b_{p-1}^P$ and $b_{p-1}^P < 0$, a contradiction. Therefore, $v_1 = 0$ or $v_p = n + 1$.

Define $b_0^P = b_p^P = 0$. If $v_1 = 0$, we create the operation $f_i' = (\mathsf{A} \leq v_{i+1} - 1, \mathsf{B} \leftarrow \mathsf{B} + (b_i^P - b_{i+1}^P))$ for $i \in \{1, \dots, p-1\}$. Otherwise, if $v_p = n+1$, we create the operation $f_i' = (\mathsf{A} \geq v_i, \mathsf{B} \leftarrow \mathsf{B} + (b_i^P - b_{i-1}^P))$ for $i \in \{1, \dots, p-1\}$. It follows that $F_P' = (f_1', \dots, f_{p-1}')$ are equivalent to $F_P = (f_1^P, \dots, f_{p-1}^P)$, and therefore $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^{\leq\geq})$ contains a bounded best diff of cost n. □

PROOF OF THEOREM 4.29. ($\Leftarrow$) Any diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_+^\mathsf{R})$ is also a diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_{\leftarrow+}^\mathsf{R})$.

($\Rightarrow$) Let $F = (f_1, \dots, f_n)$ be a best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}_{\leftarrow+}^\mathsf{R})$ of cost n that has the smallest number of assignment modifiers and, among the best diffs with the smallest number of assignment modifiers, has the smallest total length. We show that F has no assignment modifiers.

The proof follows. Assume to the contrary, and let i be the smallest index in $[n]$ such that $f_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow b_i)$ has an assignment modifier.

Case 1: There is an operation $f_j = (\mathsf{A} \in [a_j, z_j], \mathsf{B} \leftarrow \mathsf{B} + b_j)$ where $j < i$ and $a_j < a_i \leq z_j \leq z_i$. Then, let $f_j' = (\mathsf{A} \in [a_j, a_i - 1], \mathsf{B} \leftarrow \mathsf{B} + b_j)$. If F' is defined as F where f_j is replaced with f_j', then F' would still yield $F'(R_\mathsf{S}) = (R_\mathsf{T})$, but the total length of F' is smaller than that of F.

Case 2: There is an operation $f_j = (\mathsf{A} \in [a_j, z_j], \mathsf{B} \leftarrow \mathsf{B} + b_j)$ where $j < i$ and $a_i \leq a_j \leq z_i < z_j$. This case has an argument symmetric to Case 1.

Case 3: There is an operation $f_j = (\mathsf{A} \in [a_j, z_j], \mathsf{B} \leftarrow \mathsf{B} + b_j)$ where $j < i$ and $a_i \leq a_j < z_j \leq z_i$. If F' is defined as F where f_j is removed, then F' would still yield $F'(R_\mathsf{S}) = (R_\mathsf{T})$, but the cost of F' is smaller than that of F.

Case 4: None of the above. Then, all tuples matching

$\mathsf{A} \in [a_i, z_i]$ still has the same value in the B attribute, say β, in $F''(R_\mathsf{S})$, where $F'' = (f_1, \ldots, f_{i-1})$. Then, let $f'_i = (\mathsf{A} \in [a_i, z_i], \mathsf{B} \leftarrow \mathsf{B} + (b_i - \beta))$. If F' is defined as F where f_i is replaced with f'_i, then F' would still yield $F'(R_\mathsf{S}) = (R_\mathsf{T})$, but F' has fewer assignment modifiers than F.

Therefore, F has no assignment modifiers. Thus, F is also a best diff in $\Delta(R_\mathsf{S}, R_\mathsf{T}, \mathcal{F}^\mathsf{R}_+)$. □

PROOF OF LEMMA 4.34. Suppose a diff between R_S and R_T contains an operation

$$f = (\mathsf{A} \in \bigcup_{j=1}^{r} [a_j, z_j], \mathsf{B} \leftarrow b)$$

that matches $\mathsf{A} = 4k$—that is, there exists $j \in [r]$ such that $a_j \leq 4k \leq z_j$—for some $k \in \{0, \ldots, n\}$. Because, by construction, there are two tuples with $\mathsf{A} = 4k$, this operation changes their B values to the same value b. However, these tuples have different B values in R_T (-1 and -2), and assignment operators cannot assign different B values to them, a contradiction. □

PROOF OF THEOREM 5.1. The problem is similar to the view synthesis problem with unions of conjunctive queries with equality predicates, which is NP-hard [13].

When a view V respective to attributes in $\mathcal{A}$ is desired, we set the B values as follows: in R_S, set all B values to distinct positive values; in R_T, set B to the same values as in R_S, except when the tuple is in V, in which case B is set to 0.

The claim is that there is a view definition V of cost t if and only if R_S and R_T has a diff under $\mathcal{F}^=_\leftarrow$ of cost t. This is because including a conjunctive query into the view V corresponds to using the same query to set $\mathsf{B} \leftarrow 0$.

In fact, one modification to the reduction above is required to prevent "unsetting" the B value from 0. For each tuple, make multiple, say 99, copies preassigned with different positive B values. Once the B values of these 99 tuples are set to 0 (or any value), they cannot be changed back into distinct B values again using assignment modifier.

Thus, the problem is NP-hard. □