

# Reordering Columns for Smaller Indexes

Daniel Lemire<sup>a,\*</sup>, Owen Kaser<sup>b</sup>

<sup>a</sup>*LICEF, Université du Québec à Montréal (UQAM), 100 Sherbrooke West, Montreal, QC,  
H2X 3P2 Canada*

<sup>b</sup>*Dept. of CSAS, University of New Brunswick, 100 Tucker Park Road, Saint John, NB,  
Canada*

---

## Abstract

Column-oriented indexes—such as projection or bitmap indexes—are compressed by run-length encoding to reduce storage and increase speed. Sorting the tables improves compression. On realistic data sets, permuting the columns in the right order before sorting can reduce the number of runs by a factor of two or more. Unfortunately, determining the best column order is NP-hard. For many cases, we prove that the number of runs in table columns is minimized if we sort columns by increasing cardinality. Experimentally, sorting based on Hilbert space-filling curves is poor at minimizing the number of runs.

*Key words:* Data Warehousing, Indexing, Compression, Gray codes

---

## 1. Introduction

Many database queries have low selectivity. In these instances, we may need to load the content of entire columns. To improve performance and reduce memory usage, we compress columns with lightweight techniques such as run-length encoding (RLE). Yet RLE compression is better if there are long runs of identical values within columns.

Meanwhile, sorting reduces the number of these *column runs*. In fact, sorting the table before indexing can improve the speed of an index by nearly a factor of ten [39], while reducing the memory usage in a comparable manner.

Yet there are many ways to sort a table, and we are motivated to sort the table in the best possible manner. Adabi et al. recommend lexicographic sorting with “low cardinality columns serv[ing] as the leftmost sort orders” [1]. We want to justify this empirical recommendation.

For uniformly distributed tables, we show that sorting lexicographically with the columns in increasing cardinality is asymptotically optimal—for large column cardinalities. Furthermore, we show how to extend this result to all column cardinalities. As an additional contribution, we bound the suboptimality

---

\*Corresponding author. Tel.: 00+1+514 987-3000 ext. 2835; fax: 00+1+514 843-2160.

*Email addresses:* [lemire@acm.org](mailto:lemire@acm.org) (Daniel Lemire), [o.kaser@computer.org](mailto:o.kaser@computer.org) (Owen Kaser)

of sorting lexicographically for the problem of minimizing the number of runs. With this analytical bound, we show that for several realistic tables, sorting is 3-optimal or better as long as the columns are ordered in increasing cardinality.

We present our results in four steps: modeling (§ 2), a priori bounds (§ 3 and § 4), analysis of synthetic cases (§ 5) and experiments (§ 6). Specifically, the paper is organized as follows:

- There are many possible RLE implementations. In § 2, we propose to count column runs as a simplified cost model.
- In § 3, we prove that minimizing the number of runs by row reordering is NP-hard.
- In § 4, we review several orders used to sort tables in databases : the lexicographical order, the reflected Gray-code order, and so on. We regroup many of these orders into a family: the recursive orders. In § 4.1, we bound the suboptimality of sorting as a heuristic to minimize the number of runs. In § 4.2, we prove that determining the best column order is NP-hard.
- In § 5, we analytically determine the best column order for some synthetic cases. Specifically, in § 5.1, we analyze tables where all possible tuples are present. In § 5.2, we consider the more difficult problem of uniformly distributed tables. We first prove that for high cardinality columns, organizing the columns in increasing cardinality is best at minimizing the number of runs (see Theorem 2). In § 5.2.1 and § 5.2.2, we show how to extend this result to low cardinality columns for the lexicographical and reflected Gray-code orders.
- Finally, we experimentally verify the importance of column ordering in § 6, and assess other factors such as column dependencies. We show that an order based on Hilbert space-filling curves [26] is not competitive to minimize the number of runs.

## 2. Modeling RLE compression by the number of column runs

RLE compresses long runs of identical values: it replaces any run by the number of repetitions followed by the value being repeated. For example, the sequence 11111000 becomes 5-1, 3-0. In column-oriented databases, RLE makes many queries faster: sum, average, median, percentile, and arithmetic operations over several columns [46].

There are many variations on RLE:

- Counter values can be stored using fixed-length counters. In this case, any run whose length exceeds the capacity of the counter is stored as multiple runs. For example, Adabi et al. [1] use a fixed number of bits for the tuple's value, start position, and run length. We can also use variable-length counters [4, 9, 41, 55, 60, 62–64] or quantized codes [31].

- When values are represented using fewer bits than the counter values, we may add the following convention: a counter is only present after the same value is repeated twice.
- In the same spirit, we may use a single bit to indicate whether a counter follows the current value. This is convenient if we are transmitting 7-bit ASCII characters using 8-bit words [8, 32].
- It might be inefficient to store short runs using value-counter pair. Hence, we may leave short runs uncompressed (BBC [5], WAH [61] or EWAH [39]).
- Both the values and the counters have some statistical distributions. If we know these distributions, more efficient encodings are possible by combining statistical compression with RLE—such as Golomb coding [24], Lempel-Ziv, Huffman, or arithmetic encoding. Moreover, if we expect the values to appear in some specific order, we can store a delta instead of the value [30]. For example, the list of values 00011122, can be coded as the (differed-values,counter) pairs (1,3)(1,3), (1,2). This can be used to enhance compression further.
- To support binary search within an RLE array, we may store not only the value and the repetition count, but also the location of the run [8, 11, 42], or we may use a B-tree [17].
- Instead of compressing the values themselves, we may compress their bits. In bitmap indexes, for any given column, several bitmaps can be individually compressed by RLE.

It would be futile to attempt to analyze mathematically all possible applications of RLE to database indexes. Instead, we count runs of identical values. That is, if  $r_i$  is the number of runs in column  $i$  and there are  $c$  columns, we compute  $\sum_{i=1}^c r_i$ <sup>1</sup> (henceforth `RUNCOUNT`).

### 3. Minimizing the number of runs by row reordering is NP-hard

We want to minimize `RUNCOUNT` by row reordering. Consider a related problem over Boolean matrices [34]: minimizing the number of runs of ones in rows by column reordering. This “Consecutive Block Minimization” problem (CBMP) is NP-hard [23, SR17],[25]<sup>2</sup>. Yet, even if we transpose the matrix, CBMP is not equivalent to the `RUNCOUNT` minimization problem. Indeed, both sequences 001100 and 000011 have a single run of ones. Yet the sequence 001100 has three runs whereas the second sequence (000011) has only two runs. Moreover, the `RUNCOUNT` minimization problem is not limited to binary data. To our knowledge, there is no published proof that minimizing `RUNCOUNT` by row reordering is NP-hard. Hence, we provide the following result.

---

<sup>1</sup>A table of notation can be found in Appendix A.

<sup>2</sup>Another NP-hardness proof was later given by Pinar and Heath [48].

$$\begin{array}{cccc}
& \underbrace{\hspace{2cm}} & \underbrace{\hspace{2cm}} & \underbrace{\hspace{2cm}} \\
& m \text{ columns} & 5m \text{ columns} & 5m \text{ columns} \\
s \rightarrow & \dots & 11 \dots 11 & 00 \dots 00 \\
& \dots & 00 \dots 00 & 00 \dots 00 \\
& \dots & \vdots & \vdots \\
\text{incidence matrix} & 00 \dots 00 & 00 \dots 00 & 00 \dots 00 \\
& \dots & \vdots & \vdots \\
& \dots & 00 \dots 00 & 00 \dots 00 \\
t \rightarrow & \dots & 00 \dots 00 & 11 \dots 11
\end{array} \left. \vphantom{\begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}} \right\} n \text{ rows}$$

Figure 1: Matrix described in the proof of Lemma 1

**Lemma 1.** *Minimizing RUNCOUNT by row reordering is NP-hard.*

**Proof.** We prove the result by reduction from the Hamiltonian path problem, which remains NP-hard even if a starting and ending vertex are specified [GT39][23]. Consider any connected graph  $G$  having  $n$  vertices and  $m$  edges, and let  $s$  and  $t$  be respectively the beginning and end of the required Hamiltonian path.

Consider the incidence matrix of such a graph. There is a row for each vertex, and a column for each edge. The value of the matrix is one if the edge connects with the vertex, and zero otherwise. Each column has only two ones; thus it has either

1. two runs (if the ones are consecutive, and either at the top or bottom of the column)
2. three runs (if the ones are consecutive but not at the top or bottom of the column, or if there are ones at the top and bottom)
3. four runs (if the ones are not consecutive, but a one is at the top or at the bottom), or
4. five runs (in all other cases).

Thus, the number of column runs in this incidence matrix is less than  $5m$ .

We modify the incidence matrix by adding  $10m$  new columns. These columns contain only zeros, except that  $5m$  columns have the value one on the row corresponding to vertex  $s$ , and  $5m$  other columns have the value one on the row corresponding to vertex  $t$  (see Fig. 1). These new columns have either 2 runs or 3 runs depending on whether the rows corresponding to  $s$  and  $t$  are first, last or neither.

Suppose that the row corresponding to  $s$  is not first or last. Then the number of runs in the newly added  $10m$  columns is at least  $3 \times 5m + 2 \times 5m = 25m$  (or  $30m$  if both  $s$  and  $t$  are neither first nor last). Meanwhile, the number of runs in the original incidence matrix is less than  $5m$ . Thus, any row order minimizing the number of runs will have the rows corresponding to  $s$  and  $t$  first and last. Without loss of generality, we assume  $s$  is first.

A minimum-run solution is obtained from a Hamiltonian path from  $s$  to  $t$  by putting rows into the order they appear along the path. Such a solution has two columns with two runs,  $n - 3$  columns with three runs (or  $n - 2$  columns with three runs, if  $(s, t)$  were an edge of  $G$ ), and the columns for the other edges in  $G$  each have five runs. Finally, the  $10m$  added columns have two runs each. Yet having so few runs implies that an  $s$ - $t$  Hamiltonian path exists. Hence, we have reduced the  $s$ - $t$  Hamiltonian path problem to minimizing the `RUNCOUNT` by row reordering.  $\square$

#### 4. Lexicographic and Gray-code sorting

While the row reordering problem is NP-hard, sorting is an effective heuristic to enhance column-oriented indexes [39, 56]. Yet there are many ways to sort rows.

A total order over a set is such that it is transitive ( $a \leq b$  and  $b \leq c$  implies  $a \leq c$ ), antisymmetric ( $a \leq b$  and  $b \leq a$  implies  $a = b$ ) and total ( $a \leq b$  or  $b \leq a$ ). A list of tuples is *discriminating* [12] if all duplicates are listed consecutively. Orders are discriminating.

We consider sorting functions over tuples. We say that an order over  $c$ -tuples *generates* an order over  $c - 1$ -tuples if and only if the projection of all sorted lists of  $c$ -tuples on the first  $c - 1$  components is discriminating. When this property applies recursively, we say that we have a recursive order:

**Definition 1.** *A recursive order over  $c$ -tuples is such that it generates a recursive order over  $c - 1$ -tuples. All orders over 1-tuples are recursive.*

An example of an order that is *not* recursive is  $(1,0,0)$ ,  $(0,1,1)$ ,  $(1,0,1)$ , since its projection on the first two components is not discriminating:  $(1,0)$ ,  $(0,1)$ ,  $(1,0)$ . We consider several recursive orders, including lexicographic order and two Gray-code orders.

*Lexicographic order.* The lexicographic order is also commonly known as the dictionary order. When comparing two tuples  $a$  and  $b$ , we use the first component where they differ ( $a_j \neq b_j$  but  $a_i = b_i$  for  $i < j$ ) to decide which tuple is smaller (see Fig. 3a).

Let  $N_i$  be the cardinality of column  $i$  and  $n$  be the number of rows. Given all possible  $N_{1,c} \equiv \prod_{i=1}^c N_i$  tuples, we have  $N_{1,c}$  runs in the last column,  $N_{1,c-1}$  runs in the second last column and so on. Hence, we have a total of  $\sum_{j=1}^c N_{1,j}$  runs. If the  $N_i$ 's have the same value  $N_i = N$  for all  $i$ 's, then we have  $N^c + N^{c-1} + \dots + N = \frac{N^{c+1}-1}{N-1} - 1$  runs.

*Gray-code orders.* We are also interested in the more efficient Gray-code orders. A Gray code is a list of tuples such that the Hamming distance—alternatively the Lee metric [3]—between successive tuples is one [14, 20]. Knuth [37, pp. 18–20] describes two types of decimal Gray codes.

<b>0</b>	<b>0</b>	<b>0</b>
0	0	<b>1</b>
0	<b>1</b>	1
0	1	<b>0</b>
<b>1</b>	1	0
1	1	<b>1</b>
1	<b>0</b>	1
1	0	<b>0</b>
<b>2</b>	0	0
2	0	<b>1</b>
2	<b>1</b>	1
2	1	<b>0</b>

Figure 2: A table sorted in a (reflected) Gray-code order. Except for the first row, there is exactly one new run initiated in each of the  $N_{1,c}$  rows (in bold). Thus, the table has  $c - 1 + N_{1,c} = 3 - 1 + 3 \times 2 \times 2 = 14$  column runs.

- Reflected Gray decimal ordering is such that each digit goes from 0 to 9, up and down alternatively: 000, 001, ..., 009, 019, 018, ..., 017, 018, 028, 029, ..., 099, 090, ...
- Modular Gray decimal is such that digits always increase from 1 modulo 10: 000, 001, ..., 009, 019, 010, ..., 017, 018, 028, 029, 020, ...

The extension to the mixed-radix case [3, 52] from the decimal codes is straightforward [51] (see Figs. 3b and 3c).

Because the Hamming distance between successive codes is one, if all possible  $N_{1,c}$  tuples are represented, there are exactly  $c - 1 + N_{1,c}$  runs. If  $N_i = N$  for all  $i$ , then we have  $c - 1 + N^c$  runs (see Fig. 2). All recursive Gray-code orders have  $N_1$  runs in the first column,  $N_1 N_2 - N_1 + 1$  runs in the second column, and the number of runs in column  $j$  is given by

$$r_j = 1 + (N_j - 1)N_{1,j-1}. \tag{1}$$

(Being a recursive order, the values from the first  $j - 1$  columns form  $N_{1,j-1}$  blocks, where rows in each block agree on their first  $j - 1$  components. Being a Gray-code order, at any transition from one block to the next, values in column  $j$  must match.) If we assume  $N_i > 1$  for all  $i \in \{1, \dots, c\}$ , then later columns always have more runs.

From a software-implementation point of view, the lexicographic order is more convenient than the reflected and modular Gray codes. A common approach to sorting large files in external memory is divide the file into smaller files, sort them, and then merge the result. Yet, with these Gray codes, it is not possible to sort the smaller files independently: a complete pass through the entire data set may be required before sorting. Indeed, consider the these two lists sorted in reflected Gray-code order:

- Anna Awkland, Anna Bibeau, Greg Bibeau, Greg Awkland;
- Bob Awkland, Bob Bibeau.

Because we sorted the first list without knowing about the first name “Bob” a simple merging algorithm fails. For this reason, it may be faster to sort data by the lexicographic order.

Similarly, while a binary search through a lexicographically sorted list only requires comparing individual values (such as Bob and Anna), binary searches through a reflected or modular Gray-code ordered list may require the complete list of values in each column.

*Non-recursive orders.* There are balanced and nearly balanced Gray codes [21, 22, 37]. Unlike the other types of Gray codes, the number of runs in all columns is nearly the same when sorting all possible tuples for  $N_1 = N_2 = \dots = N_c$ . However, they cannot be recursive.

Some authors have used Hilbert space-filling curves to order data points [26, 28, 35] (see Fig. 3d). This order is not recursive. Indeed, the following 2-tuples are sorted in Hilbert order: (1,1), (2,1),(2,2), (1,2). Yet their projection on the first component is not discriminating: 1, 2, 2, 1. It is a balanced Gray code when all column cardinalities are the same power of two [26]. Beyond two dimensions, there are many possible orders based on Hilbert curves [2]. There are also many other alternatives such as Sierpiński-Knopp order, Peano order [47], the Gray-coded curve [19], Z-order [38] and H-index [45]. They are often selected for their locality properties [27].

If not balanced, non-recursive orders can be *column-oblivious* if the number of runs per column is independent of the order of the columns. As a trivial example, if you reorder the columns before sorting the table lexicographically, then the initial order of the columns is irrelevant.

#### 4.1. Significance of column order

Recursive orders depend on the column order. For lexicographic or reflected Gray-code orders, permuting the columns generates a new row ordering. The next proposition shows that the effect of the column ordering grows linearly with the number of columns.

**Proposition 1.** *For tables with  $c$  columns, the number of column runs after the application of any recursive-order function can vary by a factor arbitrarily close to  $c$  under the permutation of the columns.*

**Proof.** The proof is by construction. Given a recursive-order function, we find a  $c$ -column table that has many runs when processed by that function. However, swapping any column with the first yields a table that—recursively sorted in any way—has few runs.

Consider a column made of  $n$  distinct values, given in sorted order: A, B, C, D, . . . This column is the first column of a  $c$ -column table. For every odd row,

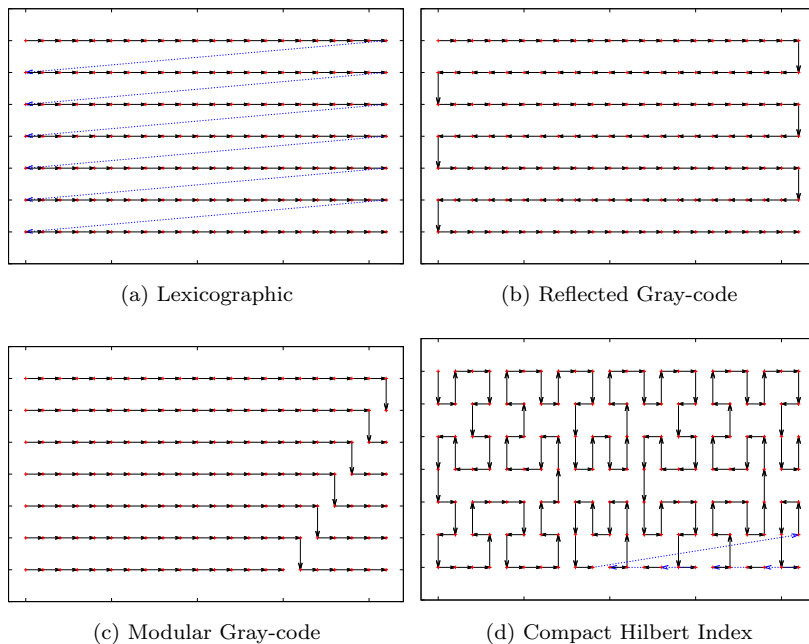


Figure 3: Various orderings of the points in a two-dimensional array

fill all remaining columns with the value 0, and every even row with the value 1:

A	0	...	0
B	1	...	1
C	0	...	0
D	1	...	1
⋮	⋮	...	⋮

This table has  $nc$  runs and is already sorted. But putting any other column first, any recursive order reduces the number of runs to  $n + 2(c - 1)$ . For  $n$  large,  $\frac{cn}{n + 2(c - 1)} \rightarrow c$  which proves the result.  $\square$

The construction in the proof uses a high cardinality column. However, we could replace this single high cardinality column by  $\lceil \log_N n \rceil$  columns having a cardinality of at most  $N$ , and the result would still hold.

Hence, recursive orders can generate almost  $c$  times more runs than an optimal order. Yet no row-reordering heuristic can generate more than  $c$  times the number of column runs than the optimal ordering solution: there are at least  $n$  column runs given  $n$  distinct rows, and no more than  $cn$  column runs in total. Hence—as row-reordering heuristics—recursive orders have no useful worst-case guarantee over arbitrary tables. We shall show that the situation differs when column reordering is permitted.



K	Y
A	Y
A	D
Z	D
Z	B
A	B
A	C
W	C
W	E
F	E
F	C
H	C
H	J

Figure 4: A table such that no recursive ordering is optimal.

Suppose we consider a sorting algorithm that first applies a known reordering to columns, then applies some recursive-order function. The proposition’s bound still applies, because we can make an obvious modification to the construction, placing the non-binary column in a possibly different position. The next refinement might be to consider a sorting algorithm that—for a given table—tries out several different column orders. For the construction we have used in the proof, it always finds an optimal ordering.

Unfortunately, even allowing the enumeration of all possible column reorderings is insufficient to make recursive ordering optimal. Indeed, consider the table in Fig. 4. The Hamming distance between any two consecutive tuples is one. Thus each new row initiates exactly one new run, except for the first row. Yet, because all tuples are distinct, this is a minimum: a Hamming distance of zero is impossible. Thus, this row ordering has a minimal number of column runs. We prove that no recursive ordering can be similarly optimal.

We begin by analyzing the neighbors of a tuple, where two tuples are neighbors if they have a Hamming distance of one:

- The tuple (K,Y) has only one neighbor: (A,Y).
- The tuple (H,J) has only one neighbor: (H,C).
- The tuples of the form (Z,·) only have neighbors of the form (A,·).
- The tuples of the form (·, E) only have neighbors of the form (·,C).

In effect, we must consider all Hamiltonian paths in the graph of neighbors. The ordered list must begin and end with (K,Y) and (H,J), if it is optimal. A recursive order must be discriminating on the first column. Without loss of generality, suppose that the list begins by (K,Y). Thus, all tuples of the form (A,·) must follow by recursivity. Then tuples of the form (Z,·) must follow. At this point, we cannot continue the list by jumping from neighbor to neighbor.

Hence, no recursive ordering is optimal. A similar argument shows that flipping the two columns leads to the same result: no recursive ordering can be optimal.

**Lemma 2.** *There are tables where no recursive order minimizes the number of runs—even after reordering the columns.*

Determining a tight bound on the suboptimality of recursive ordering remains open. Recursive orders applied to the example of Fig. 4 generate at least 15 runs whereas 14 runs is possible, for a suboptimality ratio of  $\frac{15}{14}$ . If we allow arbitrarily long two-dimensional tables, we can generalize our construction to obtain ratios arbitrarily close to  $\frac{13}{12}$ . Thus, the suboptimality ratio of recursive orders ranges between  $\frac{13}{12}$  and  $c$ . However, a computer search through 100,000 uniformly distributed tridimensional tables with 10 rows and six distinct column values failed to produce a single case where recursive ordering is suboptimal. That is, among the row orderings minimizing the number of runs, at least one is recursive after some reordering of the columns. Hence, it is possible that recursive ordering is rarely suboptimal.

The next proposition gives a simple suboptimality bound on any recursive order. This result implies that recursive ordering is 3-optimal or better for several realistic tables (see Table 1).

**Proposition 2.** *Consider a table with  $n$  distinct rows and column cardinalities  $N_i$  for  $i = 1, \dots, c$ . Recursive ordering is  $\mu$ -optimal for the problem of minimizing the runs where*

$$\mu = \frac{\sum_{j=1}^c \min(n, N_{1,j})}{n + c - 1}.$$

*The bound  $\mu$  can be made stronger if the recursive order is a Gray code:*

$$\mu_{GC} = \frac{\sum_{j=1}^c \min(n, 1 + (N_j - 1)N_{1,j-1})}{n + c - 1}.$$

*but  $\mu_{GC} > \frac{\min_i(N_i-1)}{\min_i(N_i)} \mu$*

**Proof.** Given a table in any recursive order, the number of runs in the  $i^{\text{th}}$  column is bounded by  $N_{1,i}$  and by  $n$ . Thus the number of runs in the table is no more than  $\min(N_1, n) + \min(N_{1,2}, n) + \dots + \min(N_{1,c}, n)$ . Yet there are at least  $n + c - 1$  runs in the optimally-ordered table. Hence, the result follows. The tighter bound for Gray-code orders follows similarly, and the relationship between  $\mu_{GC}$  and  $\mu$  follows by straightforward algebra ( $\mu_{GC} > \sum_{j=1}^c \min(\beta n, \beta N_{1,j})$  where  $\beta = \frac{\min_i(N_i)-1}{\min_i(N_i)}$ ).  $\square$

As an example, consider the list of all dates (month, day, year) for a century ( $N_1 = 12, N_2 = 31, N_3 = 100, n = 12 \times 31 \times 100$ ): then  $\mu \approx 1.01$  so that lexicographic sorting is within 1% of minimizing the number of runs. The optimality bound given by Proposition 2 is tighter when the columns are ordered in non-decreasing cardinality ( $N_1 \leq N_2 \leq \dots \leq N_c$ ). This fact alone can be an argument for ordering the columns in increasing cardinality.

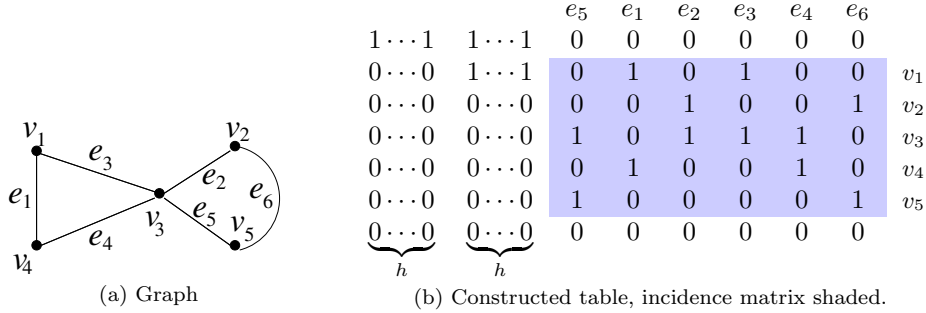


Figure 5: Table built from graph on the left. There are  $h$  copies of the column that begins 10... and the column that begins with 11...

#### 4.2. Determining the optimal column order is NP-hard

For lexicographic sorting, it is NP-hard to determine which column ordering will result in least cost under the RUNCOUNT model, even when the tables have only two values. We consider the following decision problem:

*Column-Ordering-for-Lex-Runcount (COLR).* Given table  $T$  with binary values and given integer  $K$ , is there a column ordering such that the lexicographically sorted  $T$  has at most  $K$  runs?

**Theorem 1.** *COLR is NP-complete.*

**Proof.** Clearly the problem is in NP. Its NP-hardness is shown by reduction from the variant of Hamiltonian Path where the starting vertex is given [23, GT39]. Given an instance  $(V, E)$  of Hamiltonian Path, without loss of generality let  $v_1 \in V$  be the specified starting vertex. We construct a table  $T$  as follows: first, start with the incidence matrix. Let  $V = \{v_1, v_2, \dots, v_{|V|}\}$  and  $E = \{e_1, e_2, \dots, e_m\}$ . Recall that this matrix has a column for each edge and a row for each vertex;  $a_{i,j} = 1$  if edge  $e_j$  has vertex  $v_i$  as an endpoint and otherwise  $a_{i,j} = 0$ . Vertex  $v_1$  corresponds to the first row. We prepend and append a row of zeros to the incidence matrix. Next we prepend  $h$  columns with values  $10^{|V|+1}$  (i.e., 100...0) and  $h$  columns with  $110^{|V|}$ ; see Fig. 5 for an example. The value of  $h$  is “large”; we compute the exact value later.

We show the resulting instance, with table  $T$  and bound  $K = 4h + 3(|V| - 1) + 5(m - |V| + 1)$ , satisfies the requirements for COLR if and only if  $(V, E)$  contains a Hamiltonian path starting at  $v_1$ .

First, suppose that we have a suitable Hamiltonian path in  $(V, E)$ . Let  $\epsilon_i \in E$  be the  $i^{\text{th}}$  edge along this path. Edge  $\epsilon_1$  is incident upon  $v_1$ .

Reorder the columns of  $T$ : leave the first  $2h$  columns in their current order. Next, place the columns corresponding to  $\epsilon_i$  in order  $\epsilon_1, \epsilon_2, \dots, \epsilon_{|V|-1}$ . The remaining columns follow in an arbitrary order. See Fig. 6, where it is apparent

		$e_1$	$e_4$	$e_2$	$e_6$	$e_3$	$e_5$	
$1 \cdots 1$	$1 \cdots 1$	0	0	0	0	0	0	
$0 \cdots 0$	$1 \cdots 1$	1	0	0	0	1	0	$v_1$
$0 \cdots 0$	$0 \cdots 0$	1	1	0	0	0	0	$v_4$
$0 \cdots 0$	$0 \cdots 0$	0	1	1	0	1	1	$v_3$
$0 \cdots 0$	$0 \cdots 0$	0	0	1	1	0	0	$v_2$
$0 \cdots 0$	$0 \cdots 0$	0	0	0	1	0	1	$v_5$
$0 \cdots 0$	$0 \cdots 0$	0	0	0	0	0	0	

Figure 6: A lexicographically sorted table with the required RUNCOUNT bound is obtained from the Hamiltonian path consisting of edges  $e_1, e_4, e_2, e_6$ .

that the constructed table is already lexicographically sorted<sup>3</sup> Also, the first  $2h$  columns have 2 runs each, and the  $|V| - 1$  columns for  $\epsilon_i$  have three runs each (each has the value  $0^i 110^{|V|-i}$ ). The remaining  $m - |V| + 1$  columns have five runs each: all patterns with adjacent ones have been used (and there are no duplicates); hence, all remaining patterns are of the form  $0^+ 10^+ 10^+$ . Thus the bound is met.

Next, suppose  $T$  satisfies the requirements of COLR with the given bound  $K = 4h + 3(|V| - 1) + 5(m - |V| + 1)$ . We show this implies  $(V, E)$  has a Hamiltonian path starting with  $v_1$ .

If  $h$  is large enough, we can guarantee that the first two rows have not changed their initial order. This is enforced by the  $2h$  columns that were initially placed leftmost. Their column values must end with 0 (the row of zeros is always last after lexicographic sorting). If we analyze the RUNCOUNT cost of these columns, they cost  $4h$  when the first two rows remain in their initial order, otherwise they cost  $5h$  or  $6h$ . If  $h$  is large enough, this penalty will outweigh any possible gain from having a column order that, when sorted, moves the first two rows.

Knowing the first row, we deduce that every column begins with a one if it is one of the  $2h$  columns, but it begins with a zero in every remaining column. We now focus on these remaining columns, which correspond to edges in  $E$ . Since each column value begins and ends with zero and has exactly two ones, its pattern is either  $0^+ 110^+$  (3 runs) or  $0^+ 10^+ 10^+$  (5 runs). The specified RUNCOUNT bound implies that we must have  $|V| - 1$  columns with 3 runs. The edges for these columns form the desired Hamiltonian path that starts at  $v_1$ .

To finish, we must choose  $h$  such that the penalty (for choosing a column ordering that disrupts the order of the first two rows after lexicographic sorting) exceeds any possible gain. The increased cost from  $4h$  is at least  $5h$ , a penalty of at least  $h$ . An upper bound on the gain from the other columns is  $3m$  because the RUNCOUNT is no more than  $5m$  and cannot be decreased below  $2m$ . Choose  $h = 3m + 1$ .  $\square$

---

<sup>3</sup>We sort with 1 ordered before 0.

This result can be extended to the reflected Gray-code order and, we conjecture, to all recursive orders. A related problem tries to minimize the maximum number of runs in any table column. This problem is also NP-hard (see Appendix D).

Moreover, given a very large number of rows, it might be impractical to try more than one column order. Indeed, evaluating each new solution implies sorting the table, a potentially expensive step. Thus, heuristics which only consider a few easily computed statistics, such as cardinality, are preferable.

## 5. Increasing-cardinality-order minimizes runs

Consider a sorted table. The table might be sorted in lexicographic order or in reflected Gray-code order. Can we prove that sorting the columns in increasing cardinality is a sensible heuristic to minimize the number of runs? We consider analytically two cases: (1) complete tables and (2) uniformly distributed tables.

### 5.1. Complete tables

Consider a  $c$ -column table with column cardinalities  $N_1, N_2, \dots, N_c$ . A *complete* table is one where all  $N_{1,c}$  possible tuples are present. In practice, even if a table is not complete, the projection on the first few columns might be complete.

Using a lexicographic order, a complete table has  $\sum_{j=1}^c N_{1,j}$  runs, hence the RUNCOUNT is minimized when the columns are ordered in non-decreasing cardinality:  $N_i \leq N_{i+1}$  for  $i = 1, \dots, c - 1$ . Using Gray-code ordering, a complete table has only  $c - 1 + N_{1,c}$  runs (the minimum possible) no matter how the columns are ordered. Hence, for Gray-code, the RUNCOUNT of complete tables is not sensitive to the column order.

Somewhat artificially, we can create a family of recursive orders for which the RUNCOUNT is not minimized over complete tables when the columns are ordered in increasing cardinality. Consider the following family: “when  $N_1$  is odd, use reflected Gray code order. Otherwise, use lexicographic order.” For  $N_1 = 2$  and  $N_2 = 3$ , we have 8 runs using lexicographic order. With  $N_1 = 3$  and  $N_2 = 2$ , we have 7 runs using any recursive Gray-code order. Hence, we cannot extend our analysis to all families of recursive orders from Gray-code and lexicographic orders. Nevertheless, if we assume that all column cardinalities are large, then the number of runs tends to  $N_{1,c}$  and all column orders become equivalent.

The benefits of Gray-code orders—all Gray-code orders, not just recursive Gray-code orders—over lexicographic orders are small for complete tables having high cardinalities as the next proposition shows (see Fig. 7).

**Proposition 3.** *Consider the number of runs in complete tables with columns having cardinality  $N$ . The relative benefit of Gray-code orders over lexicographic orders grows monotonically with  $c$  and is at most  $1/N$ .*

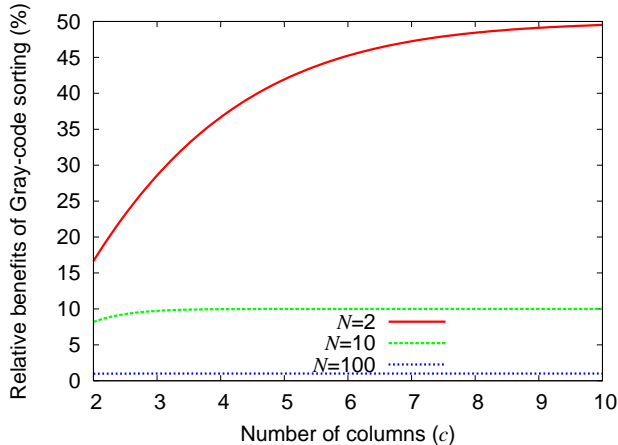


Figure 7: Relative benefits of Gray-code sorting against lexicographic orders for complete  $c$ -column table where all column cardinalities are  $N$

**Proof.** The relative benefits of Gray-code sorting for complete tables with all columns having cardinality  $N$  is  $\frac{\frac{N^{c+1}-1}{N-1} - 1 - (N^c + c - 1)}{\frac{N^{c+1}-1}{N-1} - 1}$ . As  $c$  grows, this quantity converges to  $1/N$  from below.  $\square$

### 5.2. Uniformly distributed case

We consider tables with column cardinalities  $N_1, N_2, \dots, N_c$ . Each of the  $N_{1,c}$  possible tuples is present with probability  $p$ . When  $p = 1$ , we have complete tables.

For recursive orders over uniformly distributed tables, knowing how to compute the expected number of runs in the second column of a two-column table is almost sufficient to analyze the general case. Indeed, given a 3-column table, the second column behaves just like the second column in 2-column table with  $p \leftarrow 1 - (1 - p)^{N_3}$ . Similarly, the third column behaves just like the second column in a 2-column table with  $N_1 \leftarrow N_1 N_2$  and  $N_2 \leftarrow N_3$ .

This second column is divided into  $N_1$  blocks of  $N_2$  tuples, each tuple having a probability  $p$  of being present. The expected number of tuples present in the table is  $N_1 N_2 p$ . However,  $N_1 N_2 p$  is an overestimate of the number of runs in the second column. We need to subtract the expected number of *seamless joins* between blocks: two blocks have a seamless join if the first one terminates with the first value of the second block. The expected number of seamless joins is no larger than the expected number of non-empty blocks minus one:  $N_1 \rho_{N_2} - 1$  where  $\rho_{N_2} \equiv 1 - (1 - p)^{N_2}$ . While for complete tables, all recursive Gray-code orders agree on the number of runs and seamless joins per column, the same is not true for uniformly distributed tables. Different recursive Gray-code orders have different expected numbers of seamless joins.

Nevertheless, we wish to prove a generic result for tables having large column cardinalities ( $N_i \gg 1$  for all  $i$ 's). Consider a two-column table having uniform column cardinality  $N$ . For any recursive order, the expected number of seamless joins is less than  $N\rho_N$ . However, the expected sum of the number of runs and seamless joins is  $N\rho_N$  in the first column and  $N^2p$  in the second, for a total of  $N\rho_N + N^2p$ . For a fixed table density, the ratio  $N\rho_N/(N\rho_N + N^2p)$  goes to zero as  $1/N$  since  $\rho_N \rightarrow 1$  exponentially. Hence, for tables having large column cardinalities, the expected number of seamless joins is negligible compared to the expected number of runs. The following lemma makes this result precise.

**Lemma 3.** *Let  $S_i$  and  $R_i$  be the expected number of seamless joins and runs in column  $i$ . For all recursive orders, we have*

$$\frac{\sum_{i=1}^c S_i}{\sum_{i=1}^c S_i + \sum_{i=1}^c R_i} \leq \frac{1}{\min_{k \in \{1, 2, \dots, c\}} N_k}$$

over uniformly distributed tables.

**Proof.** Column  $i + 1$  has an expected total of runs and seamless joins of  $S_{i+1} + R_{i+1} = N_{1,i+1}\rho_{N_{i+2}\dots N_c}$ . It has less than  $N_{1,i}\rho_{N_{i+1}\dots N_c}$  seamless joins. We can verify that  $\rho_{N_{i+1}\dots N_c} \leq \rho_{N_{i+2}\dots N_c}$  for all  $p \in [0, 1]$ . Thus  $S_{i+1}/(S_{i+1} + R_{i+1}) \leq 1/N_{i+1}$ .

Hence, we have  $N_i S_i \leq S_i + R_i$ . This implies that  $\min_{k \in \{1, 2, \dots, c\}} N_k S_i \leq S_i + R_i$ . Therefore, we have  $\min_{k \in \{1, 2, \dots, c\}} N_k \sum_{i=1}^c S_i \leq \sum_{i=1}^c S_i + \sum_{i=1}^c R_i$  which proves the result.  $\square$

Therefore, for large column cardinalities, we can either consider the number of runs, or the sum of the runs and seamless joins. In this context, the next proposition shows that it is best to order columns in increasing cardinality.

**Proposition 4.** *The expected sum of runs and seamless joins is the same for all recursive orders. Moreover, it is minimized over uniformly distributed tables if the columns are sorted by increasing cardinality.*

**Proof.** For all recursive orders, the expected number of runs and seamless joins for columns  $i$  and  $i + 1$  is  $N_{1,i}\rho_{N_{i+1}\dots N_c} + N_{1,i+1}\rho_{N_{i+2}\dots N_c}$ . The second term—corresponding to column  $i + 1$ —is invariant under a permutation of columns  $i$  and  $i + 1$ . We focus our attention on the first term:  $N_{1,i}\rho_{N_{i+1}\dots N_c}$ . After permuting  $i$  and  $i + 1$ , it becomes  $N_{1,i-1}N_{i+1}\rho_{N_i N_{i+2}\dots N_c}$ .

To simplify the notation, rewrite  $\rho_{N_{i+1}\dots N_c}$  and  $\rho_{N_i N_{i+2}\dots N_c}$  as  $\rho_{N_{i+1}}$  and  $\rho_{N_i}$  by substituting  $\rho_{N_{i+2}\dots N_c}$  for  $p$  and let  $i = 1$ . Thus, we compare  $N_1\rho_{N_2}$  and  $N_2\rho_{N_1}$ .

To prove the result, it is enough to show that  $N_1\rho_{N_2} < N_2\rho_{N_1}$  implies  $N_1 < N_2$  for  $p \in (0, 1]$ . Suppose that it is not the case: it is possible to have  $N_1\rho_{N_2} < N_2\rho_{N_1}$  and  $N_1 > N_2$ . Pick such  $N_1, N_2$ . Let  $x = 1 - p$ , then  $N_1\rho_{N_2} - N_2\rho_{N_1}$  is  $N_1(1 - x^{N_2}) - N_2(1 - x^{N_1})$ . The polynomial is positive for  $x = 0$  since  $N_1 > N_2$ . Because  $N_1\rho_{N_2} < N_2\rho_{N_1}$  is possible (for some value of  $x$ ), the polynomial must be negative at some point in  $(0, 1)$ , hence it

must have a root in  $(0, 1)$ . However, the polynomial has only 3 terms so that it cannot have more than 2 positive roots (e.g., by Descartes' rule of signs). Yet it has a root of multiplicity two at  $x = 1$ : after dividing by  $x - 1$ , we get  $N_1(1 + x + \dots + x^{N_2-1}) - N_2(1 + x + \dots + x^{N_1-1})$  which is again zero at  $x = 1$ . Thus, it has no such root and, by contradiction,  $N_1\rho_{N_2} \leq N_2\rho_{N_1}$  implies  $N_1 \leq N_2$  for  $p \in (0, 1]$ . The proof is concluded.  $\square$

**Theorem 2.** *Given*

1. *the expected number of runs  $R^\uparrow$  in a table sorted using any recursive order with an ordering of the column in increasing cardinality and*
2.  $R^{\text{optimal}}$ , *the smallest possible expected number of runs out of all possible recursive orders on the table (with the columns ordered in any way),*

*then*

$$\frac{R^\uparrow - R^{\text{optimal}}}{R^\uparrow} \leq \frac{1}{\min_{k \in \{1, 2, \dots, c\}} N_k}$$

*over uniformly distributed tables. That is, for large column cardinalities— $\min_{k \in \{1, 2, \dots, c\}} N_k$  is large—sorting a table recursively with the columns ordered in increasing cardinality is asymptotically optimal.*

**Proof.** Whenever  $a \geq b$ , then  $1 - a \leq 1 - b$ . Applying this idea to the statement of Lemma 3, we have

$$1 - \frac{\sum_{i=1}^c S_i}{\sum_{i=1}^c S_i + \sum_{i=1}^c R_i} \geq 1 - \frac{1}{\min_{k \in \{1, 2, \dots, c\}} N_k}$$

or

$$\sum_{i=1}^c R_i \geq \frac{\min_{k \in \{1, 2, \dots, c\}} N_k - 1}{\min_{k \in \{1, 2, \dots, c\}} N_k} \left( \sum_{i=1}^c S_i + \sum_{i=1}^c R_i \right).$$

Let  $S^\uparrow$  and  $S^{\text{optimal}}$  be the expected number of seamless joins corresponding to  $R^\uparrow$  and  $R^{\text{optimal}}$ . We have

$$\begin{aligned} R^{\text{optimal}} &\geq \frac{\min_{k \in \{1, 2, \dots, c\}} N_k - 1}{\min_{k \in \{1, 2, \dots, c\}} N_k} (R^{\text{optimal}} + S^{\text{optimal}}) \\ &\geq \frac{\min_{k \in \{1, 2, \dots, c\}} N_k - 1}{\min_{k \in \{1, 2, \dots, c\}} N_k} (R^\uparrow + S^\uparrow) \text{ by Prop. 4} \\ &\geq \frac{\min_{k \in \{1, 2, \dots, c\}} N_k - 1}{\min_{k \in \{1, 2, \dots, c\}} N_k} R^\uparrow \end{aligned}$$

from which the result follows.  $\square$

From this theorem, we can conclude that—over uniformly distributed tables having large column cardinalities—sorting lexicographically with the column ordered in increasing cardinality is as good as any other recursive sorting.



$$\begin{aligned}
& [1 \ 2 \ \cdots \ \underbrace{k \ \cdots \ N_2}_{N_2-k}] [1 \ \underbrace{2 \ \cdots \ k}_{k-1} \ \cdots \ N_2] \\
& \text{(a) Lexicographic} \\
& [1 \ 2 \ \cdots \ \underbrace{k \ \cdots \ N_2}_{N_2-k}] [\underbrace{N_2 \ N_2 - 1 \ \cdots \ k \ \cdots \ 1}_{N_2-k}] \\
& \text{(b) Reflected GC (successive blocks)} \\
& [1 \ 2 \ \cdots \ \underbrace{k \ \cdots \ N_2}_{N_2-k}] [\underbrace{N_2 \ 1 \ 2 \ \cdots \ k \ \cdots \ N_2 - 1}_{k \bmod N_2}] \\
& \text{(c) Modular GC (successive blocks)} \\
& [1 \ 2 \ \cdots \ \underbrace{k \ \cdots \ N_2}_{N_2-k}] [\underbrace{s \ s + 1 \ \cdots \ k \ \cdots \ s - 1}_{(k-1+y) \bmod N_2}] \\
& \text{(d) Modular GC (separated by } y - 1 \text{ blocks)}
\end{aligned}$$

Figure 8: Two consecutive non-empty blocks and the number of missing tuples needed to form a seamless join. The last figure shows the pattern where  $y - 1$  empty blocks separate the two non-empty blocks, and the count sequence in the second block starts at  $s = 1 + (-y \bmod N_2)$ .

The expected benefits of seamless joins are small, at least for uniformly distributed tables. Yet they cause runs from different columns to partially overlap. Such partial overlaps might prevent some computational optimizations. For this reason, Bruno [10] avoids seamless joins in RLE-compressed columns: each seamless join becomes the start of a new run. In this model, Proposition 4 already shows that ordering the columns in increasing cardinality minimizes the expected number of runs—for uniformly distributed tables.

### 5.2.1. Best column order for lexicographic sorting

While Theorem 2 states that the best column ordering—for all recursive orders—is by increasing cardinality, the result is only valid asymptotically (for large column cardinalities). For the lexicographic order, we want to prove that the best column ordering is by increasing cardinality, irrespective of the column cardinalities.

The  $N_1$  blocks in the second column of a lexicographically ordered are ordered from 1 to  $N_2$ . Let  $P_{\Downarrow N_2}$  be the probability that any two non-empty such blocks have a seamless join. The probability that the first  $x$  tuples in a block are not present whereas the  $x + 1^{\text{th}}$  tuple is present is  $(1 - p)^x p / (1 - (1 - p)^{N_2})$ . To obtain a seamless join, we need a run of precisely  $N_2 - 1$  missing tuples, and it can begin anywhere between the second possible tuple in the first block and the first possible tuple in the second block. (See Fig. 8a.) Hence, we have  $P_{\Downarrow N_2} = \frac{N_2 p^2 (1 - p)^{N_2 - 1}}{(1 - (1 - p)^{N_2})^2} = \frac{N_2 p^2 (1 - \rho_{N_2})}{(1 - p) \rho_{N_2}^2}$ . Let  $P_{\Downarrow N_2, p'}$  and  $\rho_{N_2, p'}$  be  $P_{\Downarrow N_2}$  and  $\rho_{N_2}$  with  $p'$  substituted in place of  $p$ .

To prove that ordering the columns by increasing cardinality minimizes the number of runs, it is enough to prove that permuting the columns two-by-two, so as to put the column with lesser cardinality first, never increases the number of runs. To prove this result, we need the following technical lemma.

**Lemma 4.** *For  $1 \leq N_2 < N_3 \leq 30$  and  $0 < p \leq 1$ , we have*

$$(1 - P_{\Downarrow N_3})\rho_{N_3}N_2 - P_{\Downarrow N_2, \rho_{N_3}}\rho_{N_2, \rho_{N_3}} < (1 - P_{\Downarrow N_2})\rho_{N_2}N_3 - P_{\Downarrow N_3, \rho_{N_2}}\rho_{N_3, \rho_{N_2}}.$$

**Proof.** Observe that  $1 - \rho_{N_3} = (1 - p)^{N_3}$  and  $\rho_{N_2, \rho_{N_3}} = 1 - (1 - p)^{N_2 N_3} = \rho_{N_2 N_3}$ . To prove the result, we show that:

- For  $p$  sufficiently close to 1, the result holds.
- We can turn the inequality into a polynomial in  $p$  with no root in  $(0, 1)$ .

The first item is easy: taking the limit as  $p \rightarrow 1$  on both sides of the inequality, we get  $N_2 < N_3$ . To conclude the proof, we have to show that  $(1 - P_{\Downarrow N_3})\rho_{N_3}N_2 - P_{\Downarrow N_2, \rho_{N_3}}\rho_{N_2, \rho_{N_3}} - (1 - P_{\Downarrow N_2})\rho_{N_2}N_3 + P_{\Downarrow N_3, \rho_{N_2}}\rho_{N_3, \rho_{N_2}}$  is never zero for  $p \in (0, 1)$ . We multiply this quantity by  $\rho_{N_2 N_3}$ . We proceed to show that the result is a polynomial.

Since  $1 - z^N = (1 - z)(1 + z + \dots + z^{N-1})$ , we have that the polynomial  $\rho_{N_2 N_3}$  is divisible by both  $\rho_{N_2}$  and  $\rho_{N_3}$  by respectively setting  $z = (1 - p)^{N_2}$  and  $z = (1 - p)^{N_3}$ . Hence,  $(1 - P_{\Downarrow N_3})\rho_{N_3}\rho_{N_2 N_3}$  and  $(1 - P_{\Downarrow N_2})\rho_{N_2}\rho_{N_2 N_3}$  are polynomials.

We also have that  $P_{\Downarrow N_2, \rho_{N_3}}\rho_{N_2, \rho_{N_3}} = \frac{N_2 \rho_{N_3}^2 (1 - \rho_{N_3})^{N_2 - 1}}{\rho_{N_2 N_3}}$  and similarly for  $P_{\Downarrow N_3, \rho_{N_2}}\rho_{N_3, \rho_{N_2}}$  so that  $(P_{\Downarrow N_2, \rho_{N_3}}\rho_{N_2, \rho_{N_3}} - P_{\Downarrow N_3, \rho_{N_2}}\rho_{N_3, \rho_{N_2}})\rho_{N_2 N_3}$  is a polynomial.

Hence, for any given  $N_2$  and  $N_3$ , we can check that the result holds by applying Sturm's method [7] to the polynomial over the interval  $(0, 1]$ . Because there is no root at  $p = 1$ , we have to check that the total root count over  $(0, 1]$  is always zero. We proved this result using a computer algebra system (see Appendix B) for values of  $N_2$  and  $N_3$  up to 30. This concludes the proof.  $\square$

There are  $N_1 - 1$  pairs of blocks immediately adjacent,  $N_1 - 2$  pairs of blocks separated by a single block, and so on. Hence, the expected number of seamless joins in the second column is<sup>4</sup>  $S_{N_1, N_2}^{\text{lexico}} = P_{\Downarrow N_2}\rho_{N_2}^2 \sum_{k=0}^{N_1 - 2} (N_1 - 1 - k)(1 - \rho_{N_2})^k$  or  $S_{N_1, N_2}^{\text{lexico}} = P_{\Downarrow N_2}(\rho_{N_2}N_1 + (1 - \rho_{N_2})^{N_1} - 1) = P_{\Downarrow N_2}\rho_{N_2}N_1 + \epsilon$  for  $|\epsilon| \leq 1$ .

**Proposition 5.** *Consider a table with  $c$  independent and uniformly distributed columns having cardinalities  $N_1, N_2, \dots, N_c$  (let  $2 \leq N_i \leq N_{i+1} \leq 30$  for  $i = 1, \dots, c - 1$ ). We can sort the table by lexicographic order according to various column orders. The column order  $N_1, N_2, \dots, N_c$  minimizes the number of column runs—up to a term no larger than  $c$  in absolute value.*

<sup>4</sup>We use the identity  $\sum_{k=0}^{N-2} (N - 1 - k)x^k = \frac{(1-x)N + x^N - 1}{(1-x)^2}$ .

**Proof.** Define  $T_{N_1, N_2, \rho_{N_3}}^{\text{lexico}} = N_1 N_2 \rho_{N_3} - P_{\downarrow\downarrow N_2, \rho_{N_3}} \rho_{N_2, \rho_{N_3}} N_1$  as the number of expected number of runs—up to a constant term no larger than one in absolute value—in the second column of a 3-column table with cardinalities  $N_1, N_2, N_3$  and uniform distribution. Define  $T_{N_1 N_2, N_3, p}^{\text{lexico}}$ ,  $T_{N_1, N_3, \rho_{N_2}}^{\text{lexico}}$  and  $T_{N_1 N_3, N_2, p}^{\text{lexico}}$  similarly. It is sufficient to prove that  $T_{N_1, N_2, \rho_{N_3}}^{\text{lexico}} + T_{N_1 N_2, N_3, p}^{\text{lexico}} \leq T_{N_1, N_3, \rho_{N_2}}^{\text{lexico}} + T_{N_1 N_3, N_2, p}^{\text{lexico}}$  whenever  $N_2 \leq N_3$ , irrespective of the value of  $N_1$  (allowing  $N_1 > N_3$ ). We have

$$\begin{aligned}
T_{N_1, N_2, \rho_{N_3}}^{\text{lexico}} + T_{N_1 N_2, N_3, p}^{\text{lexico}} &= N_1 N_2 \rho_{N_3} - P_{\downarrow\downarrow N_2, \rho_{N_3}} \rho_{N_2, \rho_{N_3}} N_1 \\
&\quad + N_1 N_2 N_3 p - P_{\downarrow\downarrow N_3} \rho_{N_3} N_1 N_2 \\
&= (1 - P_{\downarrow\downarrow N_3}) \rho_{N_3} N_1 N_2 \\
&\quad - P_{\downarrow\downarrow N_2, \rho_{N_3}} \rho_{N_2, \rho_{N_3}} N_1 \\
&\quad + N_1 N_2 N_3 p \\
&\leq (1 - P_{\downarrow\downarrow N_2}) \rho_{N_2} N_1 N_3 \\
&\quad - P_{\downarrow\downarrow N_3, \rho_{N_2}} \rho_{N_3, \rho_{N_2}} N_1 \\
&\quad + N_1 N_2 N_3 p \text{ (by Lemma 4)} \\
&= N_1 N_3 \rho_{N_2} - P_{\downarrow\downarrow N_3, \rho_{N_2}} \rho_{N_3, \rho_{N_2}} N_1 \\
&\quad + N_1 N_2 N_3 p - P_{\downarrow\downarrow N_2} \rho_{N_2} N_1 N_3 \\
&= T_{N_1, N_3, \rho_{N_2}}^{\text{lexico}} + T_{N_1 N_3, N_2, p}^{\text{lexico}}.
\end{aligned}$$

This proves the result.  $\square$

We conjecture that a similar result would hold for all values of  $N_i$  larger than 30. Given arbitrary values of  $N_1, N_2, \dots, N_c$ , we can quickly check whether the result holds using a computer algebra system.

### 5.2.2. Best column order for reflected Gray-code sorting

For the reflected Gray-code order, we want to prove that the best column ordering is by increasing cardinality, irrespective of the column cardinalities. Blocks in reflected Gray-code sort are either ordered from 1 to  $N_2$ , or from  $N_2$  to 1. When two non-empty blocks of the same type are separated by empty blocks, the probability of having a seamless join is  $P_{\downarrow\downarrow N_2}$ . Otherwise, the probability of seamless join is  $P_{\uparrow N_2} = \frac{p^2 + (1-p)^2 p^2 + \dots + (1-p)^{2N_2-2} p^2}{(1-(1-p)^{N_2})^2} = \frac{p^2(1-(1-p)^{2N_2})}{(1-(1-p)^{N_2})^2(1-(1-p)^2)}$  for  $p \in (0, 1)$ .

There are  $N_1 - 1$  pairs of blocks immediately adjacent and with opposite orientations (e.g., from 1 to  $N_2$  and then from  $N_2$  to 1; see Fig. 8b),  $N_1 - 2$  pairs of blocks separated by a single block and having identical orientations, and so on. Hence, the expected number of seamless joins is  $P_{\uparrow N_2} \rho_{N_2}^2 \sum_{k=0}^{\lfloor (N_1-1)/2 \rfloor} (N_1 - 1 - 2k)(1 - \rho_{N_2})^{2k} + P_{\downarrow\downarrow N_2} \rho_{N_2}^2 \sum_{k=0}^{\lfloor (N_1-3)/2 \rfloor} (N_1 - 2 - 2k)(1 - \rho_{N_2})^{2k+1}$ .

We want a simpler formula for the number of runs, at the expense of introducing an error of plus or minus one run. So consider the scenario where we have an infinitely long column, instead of just  $N_1$  blocks. However, we count only the number of seamless joins between a block in the first  $N_1$  blocks and a block

following it. Clearly, there can be at most one extra seamless join, compared to the number of seamless joins within the  $N_1$  blocks.

We have the formula  $x \sum_{k=0}^{\infty} (1-x)^{2k} = \frac{x}{1-(1-x)^2} = \frac{1}{2-x}$ . Hence, this new number of seamless joins is  $S_{N_1, N_2}^{\text{reflected}} = P_{\uparrow N_2} \rho_{N_2}^2 \sum_{k=0}^{\infty} N_1 (1-\rho_{N_2})^{2k} + P_{\downarrow N_2} \rho_{N_2}^2 \sum_{k=0}^{\infty} N_1 (1-\rho_{N_2})^{2k+1} = \frac{P_{\uparrow N_2} \rho_{N_2}^2 N_1}{2-\rho_{N_2}} + \frac{P_{\downarrow N_2} \rho_{N_2}^2 (1-\rho_{N_2}) N_1}{2-\rho_{N_2}}$ .

Let  $\lambda_{N_2}^{\text{reflected}} = \frac{P_{\uparrow N_2} + (1-\rho_{N_2}) P_{\downarrow N_2}}{2-\rho_{N_2}}$ , then  $S_{N_1, N_2}^{\text{reflected}} = \lambda_{N_2}^{\text{reflected}} \rho_{N_2} N_1$ .

**Lemma 5.** *We have that  $\frac{1-x^{N_2 N_3}}{1-x^{N_3}} = 1 + x^{N_3} + x^{2N_3} + \dots + x^{N_3(N_2-1)}$ , for all positive integers  $N_2, N_3$ .*

**Lemma 6.** *If  $2 \leq N_2 < N_3 \leq 30$ , then*

$$(1 - \lambda_{N_3}^{\text{reflected}}) \rho_{N_3} N_2 - \lambda_{N_2, \rho_{N_3}}^{\text{reflected}} \rho_{N_2, \rho_{N_3}} < (1 - \lambda_{N_2}^{\text{reflected}}) \rho_{N_2} N_3 - \lambda_{N_3, \rho_{N_2}}^{\text{reflected}} \rho_{N_3, \rho_{N_2}}.$$

**Proof.** The proof is similar to the proof of Lemma 4. We want to show that:

- For some value of  $p$  in  $(0,1)$ , the result holds.
- We can turn the inequality into a polynomial in  $p$  with no root in  $(0,1)$ .

The first item follows by evaluating the derivative of both sides of the inequality at  $p = 1$ . (Formally, our formula is defined for  $p \in (0,1)$ , so we let the values and derivatives of our functions at 1 be implicitly defined as their limit as  $p$  tends to 1.) For all  $N \geq 2$  and at  $p = 1$ , we have that  $\frac{dP_{\uparrow N}}{dp} = 2$ ,  $\frac{dP_{\downarrow N}}{dp} = 0$ ,  $\frac{d\rho_N}{dp} = 0$ , and  $\frac{d\lambda_N^{\text{reflected}}}{dp} = 2$ ; moreover, we have  $\rho_N = 1$  and  $\lambda_N^{\text{reflected}} = 1$  at  $p = 1$ . The derivatives of  $P_{\uparrow N, \rho_{N'}}$ ,  $P_{\downarrow N, \rho_{N'}}$  and  $\lambda_{N, \rho_{N'}}^{\text{reflected}}$  are also zero for all  $N, N' \geq 2$  at  $p = 1$ . Hence, the derivative of the left-hand-side of the inequality at  $p = 1$  is  $-2N_2$  whereas the derivative of the right-hand-side is  $-2N_3$ . Because  $N_3 > N_2$  and equality holds at  $p = 1$ , we have that the left-hand-side must be smaller than the right-hand-side at  $p = 1 - \xi$  for some sufficiently small  $\xi > 0$ .

To conclude the proof, we have to show that the value  $(1 - \lambda_{N_3}^{\text{reflected}}) \rho_{N_3} N_2 - \lambda_{N_2, \rho_{N_3}}^{\text{reflected}} \rho_{N_2, \rho_{N_3}} - (1 - \lambda_{N_2}^{\text{reflected}}) \rho_{N_2} N_3 + \lambda_{N_3, \rho_{N_2}}^{\text{reflected}} \rho_{N_3, \rho_{N_2}}$  is never zero for  $p \in (0,1)$ . We multiply this quantity by  $(2 - \rho_{N_2 N_3}) \rho_{N_2 N_3}$  and call the result  $\Upsilon$ . We first show that  $\Upsilon$  is a polynomial.

Because  $P_{\uparrow N_3} \rho_{N_3}^2$  and  $P_{\downarrow N_3} \rho_{N_3}^2$  are polynomials (respectively  $p^2 + (1-p)^2 p^2 + \dots + (1-p)^{2N_2-2} p^2$  and  $N_2 p^2 (1-p)^{N_2-1}$ ), we have that  $\lambda_{N_3}^{\text{reflected}}$  can be written as a polynomial divided by  $(2 - \rho_{N_3}) \rho_{N_3}^2$ . Hence,  $\lambda_{N_3}^{\text{reflected}} \rho_{N_3} (2 - \rho_{N_2 N_3}) \rho_{N_2 N_3}$  is a polynomial times  $\frac{(2 - \rho_{N_2 N_3}) \rho_{N_2 N_3}}{(2 - \rho_{N_3}) \rho_{N_3}}$ . In turn, this fraction is  $\frac{1 - (1-p)^{2N_2 N_3}}{1 - (1-p)^{2N_3}}$  which is a polynomial by Lemma 5. Hence,  $\lambda_{N_3}^{\text{reflected}} \rho_{N_3} (2 - \rho_{N_2 N_3}) \rho_{N_2 N_3}$  is a polynomial. By symmetrical arguments,  $\lambda_{N_2}^{\text{reflected}} \rho_{N_2} (2 - \rho_{N_2 N_3}) \rho_{N_2 N_3}$  is also a polynomial.

Recall that  $\rho_{N_2, \rho_{N_3}} = \rho_{N_2 N_3}$ . We have that  $\lambda_{N_2, \rho_{N_3}}^{\text{reflected}}$  is a polynomial divided by  $(2 - \rho_{N_2 N_3}) \rho_{N_2 N_3}^2$ . Hence, it is immediate that  $\lambda_{N_2, \rho_{N_3}}^{\text{reflected}} \rho_{N_2, \rho_{N_3}}$  multiplied by

$(2 - \rho_{N_2 N_3})\rho_{N_2 N_3}$  is polynomial, merely by canceling the terms in the denominator. A symmetrical argument applies to  $\lambda_{N_3, \rho_{N_2}}^{\text{reflected}} \rho_{N_3, \rho_{N_2}}$ .

Hence,  $\Upsilon$  is a polynomial. As in Lemma 4, for any given  $N_2$  and  $N_3$ , we can check that there are no roots by applying Sturm’s method to the polynomial over the interval  $(0, 1]$ . Because there is a root at  $p = 1$ , it is sufficient to check that the total root count over  $(0, 1]$  is always one. (Alternatively, we could first divide the polynomial by  $x - 1$  and check that there is no root.) We proved this result using a computer algebra system (see Appendix B). This concludes the proof.  $\square$

**Proposition 6.** *Consider a table with  $c$  independent and uniformly distributed columns having cardinalities  $N_1, N_2, \dots, N_c$  (let  $2 \leq N_i \leq N_{i+1} \leq 30$  for  $i = 1, \dots, c - 1$ ). We can sort the table by reflected Gray-code order according to various column orders. The column order  $N_1, N_2, \dots, N_c$  minimizes the number of column runs—up to a term no larger than  $c$  in absolute value.*

**Proof.** The proof is similar to Proposition 5, see Appendix C.  $\square$

## 6. EXPERIMENTS

To complete the mathematical analysis, we ran experiments on realistic data sets. We are motivated by the following questions:

- For columns with few columns, is recursive sorting nearly optimal? (§ 6.3)
- How likely is it that alternative column order are preferable to the increasing-cardinality order? (§ 6.4)
- How significant can the effect of the column order be? Are reflected Gray-code orders better than lexicographical orders? (§ 6.5)
- How does an Hilbert order compare to lexicographical orders? (§ 6.6)
- How large is the effect of skew and column dependency? (§ 6.8)
- Do our results extend to other column-compression techniques? (§ 6.9)

### 6.1. Software

We implemented the various sorting techniques using Java and the Unix command `sort`. For all but lexicographic ordering, hexadecimal values were prepended to each line in a preliminary pass over the data, before the command `sort` was called. (This approach is recommended by Richards [51].) Beside recursive orders, we also implemented sorting by Compact Hilbert Indexes (henceforth Hilbert) [26]—also by prepending hexadecimal values. By default, we order values within columns alphabetically.

Table 1: Characteristics of data sets used

	rows	distinct rows	cols	$\sum_i n_i$	size	$\mu$
<b>Census-Income</b>	199 523	178 867	4	102 609	2.96 MB	2.63
<b>Census1881</b>	4 277 807	4 262 238	7	343 422	305 MB	5.09
<b>DBGEN</b>	13 977 980	11 996 774	4	402 544	297 MB	1.02
<b>Netflix</b>	100 480 507	100 480 507	4	500 146	2.61 GB	2.00
<b>KJV-4grams</b>	877 020 839	363 412 308	4	33 553	21.6 GB	2.19

### 6.2. Realistic data sets

We used five data sets (see Table 1) representative of tables found in applications: Census-Income [29], Census1881 [50], DBGEN [57], Netflix [43] and KJV-4grams [39]. The Census-Income table has 4 columns: *age*, *wage per hour*, *dividends from stocks* and a numerical value<sup>5</sup> found in the 25<sup>th</sup> position of the original data set. The respective cardinalities are 91, 1 240, 1 478 and 99 800. The Census1881 came from a publicly available SPSS file 1881\_sept2008.SPSS.rar [50] that we converted to a flat file. In the process, we replaced the special values “ditto” and “do.” by the repeated value, and we deleted all commas within values. The column cardinalities are 183, 2 127, 2 795, 8 837, 24 278, 152 365, 152 882. For DBGEN, we selected dimensions of cardinality 7, 11, 2 526 and 400 000. The Netflix table has 4 dimensions: UserID, MovieID, Date and Rating, with cardinalities 480 189, 17 770, 2 182, and 5. Each of the four columns of KJV-4grams contains roughly 8 thousand distinct stemmed words: 8 246, 8 387, 8 416, and 8 504.

Table 1 also gives the suboptimality factor  $\mu$  from Proposition 2. For DBGEN, any recursive order minimizes the number of runs optimally—up to a factor of 1%. For Netflix and KJV-4-grams, recursive ordering is 2-optimal. Only for Census1881 is the bound on optimality significantly weaker: in this instance, recursive ordering is 5-optimal.

### 6.3. Recursive sorting is “safe” for low dimensionality

Since our 7-dimensional data set yields a much looser bound than the 4-dimensional data sets, we investigate the relationship between  $\mu$  and the number of dimensions. Rather than use our arbitrarily chosen low-dimensional projections, we randomly generated many projections (typically 1000) of each original data set, computed  $\mu$  for each projection, then showed the  $\mu$  values for each dimensionality (i.e., all  $\mu$  values for 3-dimensional projections were averaged and reported; likewise all  $\mu$  values for 4-dimensional projects were averaged and reported). One difficulty arose: computing  $\mu$  for a projection required the number of distinct rows, and we projected from data sets that are at least a large fraction of our main-memory size. Gathering this data exactly appears too expensive. Instead, we computed the projection sizes in a few passes over

<sup>5</sup>The associated metadata says this column should be a 10-valued migration code.

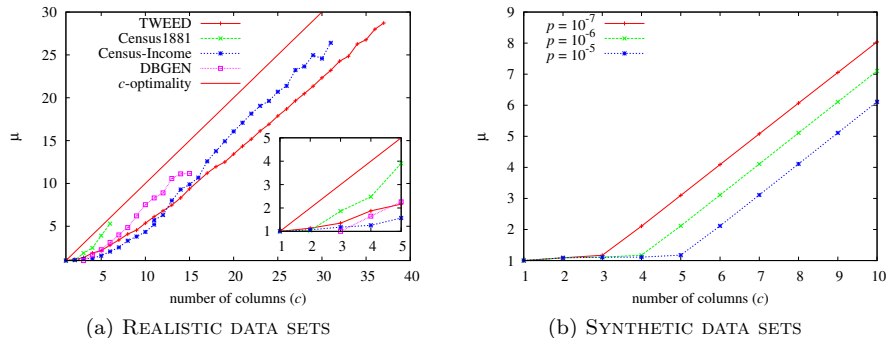


Figure 9: Approximate  $\mu$  versus columns, when sampling projections of realistic data sets and synthetic data sets (10-dimensional uniformly distributed table with  $N_1 = N_2 = \dots = N_{10} = 10$ )

our full data sets, using a probabilistic counting technique due to Cai et al. [13] that was shown by Aouiche and Lemire [6] to have a good performance. As an extra step, we corrected the distinct-row estimates so that they never exceeded the product of column cardinalities. To validate our estimates, we computed exact  $\mu$  values for two smaller data sets (TWEED [18, 59] with 52 dimensions and 11k rows, and another with 13 dimensions and 581k rows) and observed our average  $\mu$  estimates changed by less than 2%. KJV-4grams and Netflix only had 4 dimensions, and thus we used TWEED to get another high-dimensional data set.

Figure 9a shows that, after about 3 dimensions,  $\mu$  grew roughly linearly with the number of dimensions: the  $\mu$  formula’s  $\min(n, N_{1,i})/(n + c - 1)$  terms would typically approximate 1 for all but the first few dimensions. To illustrate this, we computed the expected value of  $\mu$  for projections of uniformly distributed tables with various densities  $p$  (see Fig. 9b).

This does *not* mean that any particular recursive sorting algorithm will be this far from optimal. Our  $\mu$  is an upper bound on suboptimality, so it merely means that we have not given evidence that recursive sorting is necessarily good for higher-dimension data sets. For high-dimensional data sets, there could still be a significant advantage in going beyond lexicographic sorting or other recursive sorting approaches. However, for 2 or 3 dimensions, our  $\mu$  values show that lexicographic sorting cannot be improved much. Of course, such projections may be nearly complete tables (cf. § 5.1).

#### 6.4. The column-reordering heuristic is reliable

We showed in § 5 that reordering the columns in increasing cardinality minimized the expected number of runs. To assess the reliability of this heuristic, consider a two-dimensional table model where the first column’s values are selected uniformly at random from 1 to  $N_1$ , and the second column’s values are selected uniformly from 1 to  $N_1 + 1$ : we want the second column to have just

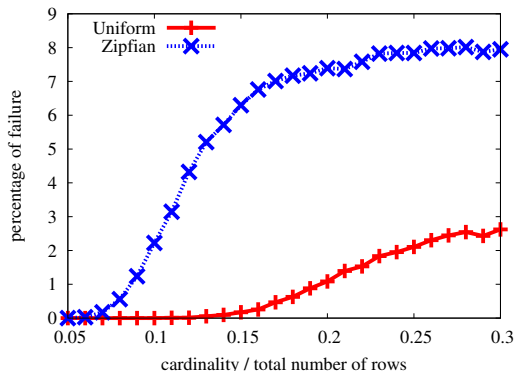


Figure 10: Percentage of failure of the increasing-cardinality column-reordering heuristic

barely a higher cardinality than the first. Using this model, we generated 100 000 100-row tables for each column cardinality  $N_1$  from 5 to 30. Of course, we can expect a few missing values when selecting 100 items (with replacement) from  $N_1$ . Some tables had more missing values in the second column, so we kept only the randomly generated tables where the second column had the higher actual cardinality. We then determined the percentage of tables where the increasing-cardinality column-reordering heuristic failed to be at least as good as the alternative column order (see Fig. 10). The expected relative difference between the cardinalities ranges from  $\approx 1/5$  to  $\approx 1/30$ . We see that the rate of failure increases as the relative difference between the column cardinalities goes to zero. Even so, the rate of failure is relatively low in this test (less than 3%) despite the small relative difference in cardinality.

Although our theoretical results assume uniformity, we have observed that ordering columns in ascending order also tends to improve results with skewed data. To assess reliability, we repeated the same test for Zipfian-distributed columns and found the rate of failure was larger. However, it still remained moderate (less than 9%). Moreover, even with Zipfian distributions, the rate of failure is close to zero when the relative difference between the column cardinalities is large (0.05).

### 6.5. Column order matters, Gray codes do not

Results for realistic data sets are given in Table 2. For these data sets, there is no noticeable benefit (within 1%) to Gray codes as opposed to lexicographic orders. The only data set showing some benefit ( $\approx 1\%$ ) is KJV-4grams.

Relative to the shuffled case, ordering the columns in increasing cardinality reduced the number of runs by a factor of two (Census and Census1881), three (DBGEN and Netflix) or nine (KJV-4grams). Except for Netflix and KJV-4grams, these gains drop to  $\approx 50\%$  when using the wrong column order (by



Table 2: RUNCOUNT after sorting various tables using different orderings. The up and down arrows indicate whether the columns were ordered in increasing or decreasing cardinality before sorting. Best results for each data set are in bold.

table	shuffled	order	lexico.	Gray	Hilbert
Census-Income	$4.6 \times 10^5$	↓	$3.2 \times 10^5$	$3.2 \times 10^5$	$3.4 \times 10^5$
		↑	<b><math>1.9 \times 10^5</math></b>	<b><math>1.9 \times 10^5</math></b>	$3.4 \times 10^5$
Census1881	$2.7 \times 10^7$	↓	$1.8 \times 10^7$	$1.8 \times 10^7$	$2.0 \times 10^7$
		↑	<b><math>1.3 \times 10^7</math></b>	<b><math>1.3 \times 10^7</math></b>	$2.0 \times 10^7$
DBGEN	$4.5 \times 10^7$	↓	$3.3 \times 10^7$	$3.3 \times 10^7$	$4.3 \times 10^7$
		↑	<b><math>1.2 \times 10^7</math></b>	<b><math>1.2 \times 10^7</math></b>	$4.3 \times 10^7$
Netflix	$3.8 \times 10^8$	↓	$2.5 \times 10^8$	$2.5 \times 10^8$	$3.3 \times 10^8$
		↑	<b><math>1.2 \times 10^8</math></b>	<b><math>1.2 \times 10^8</math></b>	$3.3 \times 10^8$
KJV-4grams	$3.4 \times 10^9$	↓	$3.9 \times 10^8$	<b><math>3.8 \times 10^8</math></b>	$8.2 \times 10^8$
		↑	$3.9 \times 10^8$	<b><math>3.8 \times 10^8</math></b>	$8.2 \times 10^8$

decreasing cardinality). On Netflix, the difference between the two column orders is a factor of two ( $2.5 \times 10^8$  versus  $1.2 \times 10^8$ ).

The data set KJV-4grams appears oblivious to column reordering. We are not surprised given that columns have similar cardinalities and distributions.

### 6.6. Compact Hilbert Indexes are not competitive

Hilbert is effective at improving the compression of database tables [15, 16] using tuple difference coding techniques [44]. Moreover, for complete tables where the cardinality of all columns is the same power of two, sorting by Hilbert minimizes the number of runs (being a Gray code § 4). However, we are unaware of any application of Hilbert to column-oriented indexes.

To test Hilbert, we generated a small random table (see Table 3) with moderately low density ( $p = 0.01$ ). The RUNCOUNT result is far worse than recursive ordering, even when all column cardinalities are the same power of 2. In this test, Hilbert is column-order oblivious. We have similar results over realistic data sets (see Table 2). In some instances, Hilbert is nearly as bad as a random shuffle of the table, and always inferior to a mere lexicographic sort. For KJV-4grams, Hilbert is relatively effective—reducing the number of runs by a factor of 4—but it is still half as effective as lexicographically sorting the data.

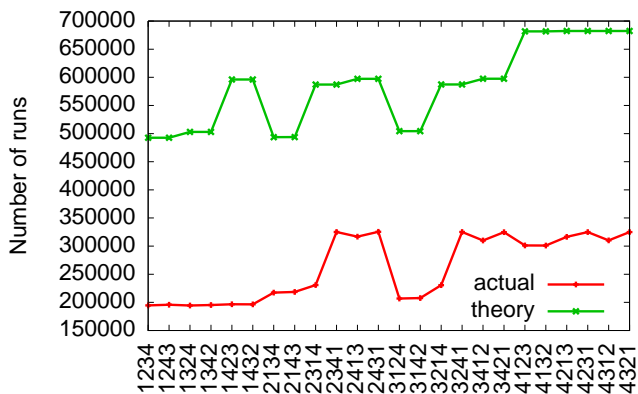
### 6.7. The order of values is irrelevant

For several recursive orders (lexicographic and Gray codes), we reordered the attribute values by their frequency—putting the most frequent values first [36]. While the number of runs in sorted uniformly distributed tables is oblivious to the order of attribute values, we may see some benefits with tables having

Table 3: Comparison of Compact Hilbert Indexes with other orderings for a uniformly distributed table ( $p = 0.01$ ,  $c = 5$ ) and various column cardinalities. The number of runs is given in thousands.

cardinalities	shuffled	lexico.	reflected Gray	modular Gray	Hilbert
4,8,16,32,64	47.8	18.9	<b>18.7</b>	18.7	35.3
64,32,16,8,4	47.8	28.5	<b>28.1</b>	28.2	35.3
16,16,16,16,16	49.7	23.7	<b>23.3</b>	23.4	35.3

Figure 11: Number of runs for the Census-Income data set, including the expected number assuming that the table is uniformly distributed. The columns are indexed from 1 to 4 in increasing cardinality. Hence, the label 1234 means that the columns are in increasing cardinality.



skewed distributions. However, on the realistic data sets, the differences were small—less than 1% on all metrics for recursive ordering.

For Hilbert, reordering the attribute values had small and inconsistent effects. For Census, the number of runs went up from  $3.4 \times 10^5$  to  $3.6 \times 10^5$  (+6%), whereas for Netflix, it went down from  $3.3 \times 10^8$  to  $3.2 \times 10^8$  (-3%). The strongest effect was observed with KJV-4grams where the number of runs went down from  $8.2 \times 10^8$  to  $7.6 \times 10^8$  (-7%). These differences are never sufficient to make Hilbert competitive.

#### 6.8. Skew and column dependencies reduce the number of runs

We can compute the expected number of runs for uniformly distributed tables sorted lexicographically by the proof of Proposition 5. For Census-Income, we compared this value with the number of runs for all possible column orders (see Fig. 11). Distribution skew and dependencies between columns make a substantial difference: the number of runs would be twice as high were Census-Income uniformly distributed with independent columns.

### 6.9. Effect of column order on alternative column-compression techniques

For some implementations of RLE, the compressed size of the columns is proportional to the `RUNCOUNT`. Thus, the `RUNCOUNT` reduction translate directly into proportionally smaller tables and faster decompression. But what about other compression techniques? To answer this question, we implemented three column-compression schemes from the SAP NetWeaver platform [40]: Prefix, Sparse and Indirect Coding. We found that for a given compression scheme, the decompression time is roughly proportional to the compressed size. Thus, we only report the compressed sizes (see Table 4). The benefits of ordering the columns in increasing cardinality can be large (up to a factor of two). However, there are also instances where ordering the columns in decreasing cardinality is slightly better (by up to 5%). Overall, our recommendation to order the columns in increasing cardinality remains valid.

## CONCLUSION

Unsurprisingly, an effective heuristic to minimize the number of runs and column-oriented index sizes is to sort lexicographically after reordering the columns in increasing cardinality. This heuristic is often recommended [1, 39]. However, our results stress the importance of reordering the columns. Picking the wrong column order can result in a moderate reduction of the number of runs (50%) whereas a large reduction is possible ( $2-3\times$ ) when using the right column order.

The benefit of recursive Gray-code orders over lexicographic orders is small. Sorting the values within columns has also small effects ( $\leq 1\%$ ) for several recursive orders.

## FUTURE WORK

The first step toward the estimation of the size of column indexes under sorting is to assume that columns are statistically independent. However, it might possible to lift this assumption by modeling the dependency between columns [49, 58].

From a practical point of view, we found that sorting tables lexicographically is effective, especially when the columns are ordered in increasing cardinality. Nevertheless, we might sometimes want to spend more time reordering rows, even for modest gains. Thus, we are currently investigating more expensive row reordering techniques [33, 53].

## Acknowledgments

This work is supported by NSERC grants 155967 and 261437.

Table 4: Compressed sizes under different compression schemes (in MB). The up and down arrows indicate whether the columns were ordered in increasing or decreasing cardinality before sorting.

(a) Sparse Coding			
table	shuffled	order	lexico.
Census-Income	0.18	↓ ↑	0.17 <b>0.14</b>
Census1881	11.2	↓ ↑	<b>8.3</b> 8.8
DBGEN	14.6	↓ ↑	12.9 <b>10.3</b>
(b) Indirect Coding			
table	shuffled	order	lexico.
Census-Income	0.26	↓ ↑	0.20 <b>0.15</b>
Census1881	12.6	↓ ↑	8.3 <b>7.9</b>
DBGEN	18.9	↓ ↑	10.9 <b>10.3</b>
(c) Prefix Coding			
table	shuffled	order	lexico.
Census-Income	0.27	↓ ↑	0.26 <b>0.12</b>
Census1881	12.6	↓ ↑	<b>10.1</b> 10.3
DBGEN	13.9	↓ ↑	13.1 <b>10.1</b>

## References

- [1] D. Abadi, S. Madden, M. Ferreira, Integrating compression and execution in column-oriented database systems, in: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2006, pp. 671–682.
- [2] J. Alber, R. Niedermeier, On multidimensional curves with Hilbert property, Theory of Computing Systems 33 (4) (2000) 295–312.

- [3] M. Anantha, B. Bose, B. AlBdaiwi, Mixed-radix Gray codes in Lee metric, *IEEE Transactions on Computers* 56 (10) (2007) 1297–1307.
- [4] V. N. Anh, A. Moffat, Inverted index compression using word-aligned binary codes, *Information Retrieval* 8 (1) (2005) 151–166.
- [5] G. Antoshenkov, Byte-aligned bitmap compression, in: *Proceedings of the Conference on Data Compression*, IEEE Computer Society, Washington, DC, USA, 1995, p. 476.
- [6] K. Aouiche, D. Lemire, A comparison of five probabilistic view-size estimation techniques in OLAP, in: *Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*, ACM, New York, NY, USA, 2007, pp. 17–24.
- [7] S. Barnard, *Higher algebra*, Barnard Press, 2008.
- [8] M. Bassiouni, Data compression in scientific and statistical databases, *IEEE Transactions on Software Engineering* 11 (10) (1985) 1047–1058.
- [9] B. Bhattacharjee, L. Lim, T. Malkemus, G. Mihaila, K. Ross, S. Lau, C. McArthur, Z. Toth, R. Sherkat, Efficient index compression in DB2 LUW, *Proceedings of the VLDB Endowment* 2 (2009) 1462–1473.
- [10] N. Bruno, Teaching an old elephant new tricks, in: *Conference on Innovative Data Systems Research*, 2009.
- [11] S. Büttcher, C. L. A. Clarke, Index compression is good, especially for random access, in: *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, 2007, pp. 761–770.
- [12] J. Cai, R. Paige, Using multiset discrimination to solve language processing problems without hashing, *Theoretical Computer Science* 145 (1-2) (1995) 189–228.
- [13] M. Cai, J. Pan, Y.-K. Kwok, K. Hwang, Fast and accurate traffic matrix measurement using adaptive cardinality counting, in: *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, 2005, pp. 205–206.
- [14] J.-C. Chen, C.-H. Tsai, Conditional edge-fault-tolerant hamiltonicity of dual-cubes, *Information Sciences* 181 (3) (2011) 620 – 627.
- [15] F. Dehne, T. Eavis, B. Liang, Compressing data cube in parallel OLAP systems, *Data Science Journal* 6 (0) (2007) 184–197.
- [16] T. Eavis, D. Cueva, A Hilbert space compression architecture for data warehouse environments, *Lecture Notes in Computer Science* 4654 (2007) 1–12.

- [17] M. Y. Eltabakh, W.-K. Hon, R. Shah, W. G. Aref, J. S. Vitter, The SBC-tree: an index for run-length compressed sequences, in: Proceedings of the 11th international conference on Extending database technology: Advances in database technology, 2008, pp. 523–534.
- [18] J. O. Engene, Five decades of terrorism in Europe: The TWEED dataset, *Journal of Peace Research* 44 (1) (2007) 109–121.
- [19] C. Faloutsos, Multiattribute hashing using Gray codes, *SIGMOD Record* 15 (2) (1986) 227–238.
- [20] J.-F. Fang, The bipancycle-connectivity of the hypercube, *Information Sciences* 178 (24) (2008) 4679 – 4687.
- [21] M. Flahive, Balancing cyclic R-ary Gray codes II, *Electronic Journal of Combinatorics* 15 (R128) (2008) 1.
- [22] M. Flahive, B. Bose, Balancing cyclic R-ary Gray codes, *Electronic Journal of Combinatorics* 14 (R31) (2007) 1.
- [23] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
- [24] S. W. Golomb, Run-length encodings, *IEEE Transactions on Information Theory* 12 (1966) 399–401.
- [25] S. Haddadi, A note on the NP-hardness of the consecutive block minimization problem, *International Transactions in Operational Research* 9 (6) (2002) 775–777.
- [26] C. H. Hamilton, A. Rau-Chaplin, Compact Hilbert indices: Space-filling curves for domains with unequal side lengths, *Information Processing Letters* 105 (5) (2007) 155–163.
- [27] H. Haverkort, F. van Walderveen, Locality and bounding-box quality of two-dimensional space-filling curves, *Computational Geometry: Theory and Applications* 43 (2010) 131–147.
- [28] H. J. Haverkort, F. van Walderveen, Four-dimensional Hilbert curves for R-trees, in: *Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments*, 2009, pp. 63–73.
- [29] S. Hettich, S. D. Bay, The UCI KDD archive, <http://kdd.ics.uci.edu> (Last checked 01-04-2011) (2000).
- [30] A. L. Holloway, D. J. DeWitt, Read-optimized databases, in depth, *Proceedings of the VLDB Endowment* 1 (1) (2008) 502–513.
- [31] A. L. Holloway, V. Raman, G. Swart, D. J. DeWitt, How to barter bits for chronons: Compression and bandwidth trade offs for database scans, in: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, ACM, New York, NY, USA, 2007, pp. 389–400.

- [32] B. R. Iyer, D. Wilhite, Data compression support in databases, in: Proceedings of the 20th International Conference on Very Large Data Bases, 1994, pp. 695–704.
- [33] D. Johnson, S. Krishnan, J. Chhugani, S. Kumar, S. Venkatasubramanian, Compressing large boolean matrices using reordering techniques, in: Proceedings of the Thirtieth international conference on Very large data bases, VLDB Endowment, 2004, pp. 13–23.
- [34] D. S. Johnson, S. Krishnan, J. Chhugani, S. Kumar, S. Venkatasubramanian, Compressing large boolean matrices using reordering techniques, in: Proceedings of the Thirtieth international conference on Very large data bases, VLDB Endowment, San Jose, CA, USA, 2004, pp. 13–23.
- [35] I. Kamel, C. Faloutsos, Hilbert R-tree: An improved R-tree using fractals, in: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994, pp. 500–509.
- [36] O. Kaser, D. Lemire, Attribute value reordering for efficient hybrid OLAP, *Information Sciences* 176 (16) (2006) 2304–2336.
- [37] D. E. Knuth, *The Art of Computer Programming*, vol. 4, chap. fascicle 2, Addison Wesley, Reading, MA, USA, 2005.
- [38] H. Lebesgue, *Leçons sur l'intégration et la recherche des fonctions primitives: professées au Collège de France*, Gauthier-Villars, 1904.
- [39] D. Lemire, O. Kaser, K. Aouiche, Sorting improves word-aligned bitmap indexes, *Data & Knowledge Engineering* 69 (1) (2010) 3–28.
- [40] C. Lemke, K.-U. Sattler, F. Faerber, A. Zeier, Speeding up queries in column stores, in: *Data Warehousing and Knowledge Discovery*, vol. 6263 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2010, pp. 117–129.
- [41] A. Moffat, L. Stuiver, Binary interpolative coding for effective index compression, *Information Retrieval* 3 (1) (2000) 25–47.
- [42] A. Moffat, J. Zobel, Self-indexing inverted files for fast text retrieval, *ACM Transactions on Information Systems* 14 (4) (1996) 349–379.
- [43] Netflix, Inc., Netflix Prize, <http://www.netflixprize.com> (Last checked 01-04-2011) (2007).
- [44] W. Ng, C. Ravishankar, Block-oriented compression techniques for large statistical databases, *IEEE Transactions on Knowledge and Data Engineering* 9 (2) (1997) 314–328.
- [45] R. Niedermeier, K. Reinhardt, P. Sanders, Towards optimal locality in mesh-indexings, *Discrete Applied Mathematics* 117 (1-3) (2002) 211–237.

- [46] P. O’Neil, D. Quass, Improved query performance with variant indexes, in: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, 1997, pp. 38–49.
- [47] G. Peano, Sur une courbe, qui remplit toute une aire plane, *Mathematische Annalen* 36 (1) (1890) 157–160.
- [48] A. Pinar, M. T. Heath, Improving performance of sparse matrix-vector multiplication, in: Proceedings of the 1999 ACM/IEEE conference on Supercomputing, ACM, New York, NY, USA, 1999, Article No. 30.
- [49] V. Poosala, Y. E. Ioannidis, Selectivity estimation without the attribute value independence assumption, in: Proceedings of the 23rd International Conference on Very Large Data Bases, 1997, pp. 486–495.
- [50] Programme de recherche en démographie historique, PRDH 1881, <http://www.prdh.umontreal.ca/census/en/main.aspx>, last checked 01-04-2011 (2009).
- [51] D. Richards, Data compression and Gray-code sorting, *Information Processing Letters* 22 (4) (1986) 201–205.
- [52] C. Savage, A survey of combinatorial Gray codes, *SIAM Review* 39 (1997) 605–629.
- [53] M. Schaller, Reclustering of high energy physics data, in: Proceedings of the 11th International Conference on Scientific and Statistical Database Management, IEEE Computer Society, Washington, DC, USA, 1999, pp. 194–203.
- [54] W. Schelter, et al., Maxima, a computer algebra system, <http://maxima.sourceforge.net/> (Last checked 01-04-2011) (1998).
- [55] F. Scholer, H. Williams, J. Yiannis, J. Zobel, Compression of inverted indexes for fast query evaluation, in: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, NY, USA, 2002, pp. 222–229.
- [56] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O’Neil, P. O’Neil, A. Rasin, N. Tran, S. Zdonik, C-Store: a column-oriented DBMS, in: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, San Jose, CA, USA, 2005, pp. 553–564.
- [57] Transaction Processing Performance Council, DBGEN 2.4.0, <http://www.tpc.org/tpch/> (Last checked 01-04-2011) (2006).
- [58] B. D. Vo, K.-P. Vo, Compressing table data with column dependency, *Theoretical Computer Science* 387 (3) (2007) 273–283.



- [59] H. Webb, O. Kaser, D. Lemire, Pruning attribute values from data cubes with diamond dicing, in: Proceedings of the 2008 international symposium on Database engineering & applications, 2008, pp. 121–129.
- [60] I. H. Witten, A. Moffat, T. C. Bell, Managing gigabytes (2nd ed.): compressing and indexing documents and images, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
- [61] K. Wu, E. J. Otoo, A. Shoshani, Optimizing bitmap indices with efficient compression, *ACM Transactions on Database Systems* 31 (1) (2006) 1–38.
- [62] H. Yan, S. Ding, T. Suel, Inverted index compression and query processing with optimized document ordering, in: Proceedings of the 18th international conference on World wide web, 2009, pp. 401–410.
- [63] J. Zhang, X. Long, T. Suel, Performance of compressed inverted list caching in search engines, in: Proceeding of the 17th international conference on World Wide Web, 2008, pp. 387–396.
- [64] J. Zobel, A. Moffat, Inverted files for text search engines, *ACM Computing Surveys* 38 (2) (2006) 6.

## A. Table of Notation

Notation	explanation	defined	used in
$r_i$	runs in column $i$	p. 3	§ 1
$c$	number of columns	p. 3	throughout
$n$	number of rows	p. 5	throughout
$N_i$	cardinality of column $i$	p. 5	throughout
$N_{i,j}$	$\prod_{k=i}^j N_k$	p. 5	throughout
$\mu$	recursive sorting is $\mu$ -optimal for the run minimization problem	p. 10	throughout
$\rho_{N_i}$	probability that a block of $N_i$ tuples is nonempty	p. 14	§ 5.2
$\rho_{N_i,p'}$	same except individual tuples present with probability $p'$ rather than default $p$		§ 5.2
$P_{\Downarrow N_2}$	with lexicographic sorting, probability that two nonempty blocks in column 2 seamlessly join	p. 17	§ 5.2
$P_{\Downarrow N_2,p'}$	same except individual tuples present with probability $p'$		§ 5.2
$P_{\Uparrow N_2}$	with reflected Gray sorting, probability that two nonempty blocks in column 2 seamlessly join	p. 19	§ 5.2
$P_{\Uparrow N_2,p'}$	same except individual tuples present with probability $p'$		§ 5.2

## B. Maxima Computer Algebra System code

For completing some of the proofs, we used Maxima version 5.12.0 [54]. Scripts ran during about 49 hours on a Mac Pro with two double-core Intel Xeon processors (2.66 GHz) and 2 GiB of RAM.

The proof of Lemma 4 uses the following code which ran for 185 minutes:

```

r(N2,p):=1-(1-p)**N2;
Pdd(N2,p):=N2*p**2*(1-r(N2,p))/((1-p)*r(N2,p)**2);
P:(1-Pdd(N3,p))*r(N3,p)*N2- (1-Pdd(N2,p))*r(N2,p)*N3
-Pdd(N2,r(N3,p))*r(N2*N3,p)+Pdd(N3,r(N2,p))*r(N2*N3,p);
P2:P*r(N2*N3,p);

```

```

for n2:2 unless n2>30 do
  (display(n2),
  for n3:n2+1 unless n3>100 do
    ( nr: nroots(factor(subst([N2=n2,N3=n3],P2)),0,1),
    if(not(nr=0)) then display("ERROR",n2,n3,nr)));

```

The proof of Lemma 6 uses this code which ran for 46 hours:

```

r(N2,p):=1-(1-p)**N2;
Pdd(N2,p):=N2*p**2*(1-r(N2,p))/((1-p)*r(N2,p)**2);
Pud(N2,p):=p**2*(2-r(N2,p))/(r(N2,p)*(1-(1-p)**2));
Lambda(N2,p):=(Pud(N2,p)+(1-r(N2,p))*Pdd(N2,p))/(2-r(N2,p));
P:(1-Lambda(N3,p))*r(N3,p)*N2- (1-Lambda(N2,p))*r(N2,p)*N3
-Lambda(N2,r(N3,p))*r(N2*N3,p)+Lambda(N3,r(N2,p))*r(N2*N3,p);
P2:P*(2-r(N2*N3,p))*r(N2*N3,p);

```

```

for n2:2 unless n2>30 do
  (display(n2),
  for n3:n2+1 unless n3>100 do
    ( nr: nroots(factor(subst([N2=n2,N3=n3],P2)),0,1),
    if(not(nr=1)) then display("ERROR",n2,n3,nr)));

```

### C. Proof of Proposition 6

**Proof.** Define  $T_{N_1, N_2, \rho_{N_3}}^{\text{reflected}} = N_1 N_2 \rho_{N_3} - S_{N_1, N_2, \rho_{N_3}}^{\text{reflected}}$  where  $S_{N_1, N_2, \rho_{N_3}}^{\text{reflected}}$  is defined as  $S_{N_1, N_2}^{\text{reflected}}$  after substituting  $\rho_{N_3}$  for  $p$ . Define  $\lambda_{N_2, \rho_{N_3}}^{\text{reflected}}$ ,  $T_{N_1 N_2, N_3, p}^{\text{reflected}}$ ,  $T_{N_1, N_3, \rho_{N_2}}^{\text{reflected}}$  and  $T_{N_1 N_3, N_2, p}^{\text{reflected}}$  similarly. As in the proof of Proposition 5, it is sufficient to prove that  $T_{N_1, N_2, \rho_{N_3}}^{\text{reflected}} + T_{N_1 N_2, N_3, p}^{\text{reflected}} \leq T_{N_1, N_3, \rho_{N_2}}^{\text{reflected}} + T_{N_1 N_3, N_2, p}^{\text{reflected}}$  whenever  $N_2 \leq N_3$ , irrespective of the value of  $N_1$  (allowing  $N_1 > N_3$ ).

We have

$$\begin{aligned}
T_{N_1, N_2, \rho_{N_3}}^{\text{reflected}} + T_{N_1 N_2, N_3, p}^{\text{reflected}} &= N_1 N_2 \rho_{N_3} - \lambda_{N_2, \rho_{N_3}}^{\text{reflected}} \rho_{N_2, \rho_{N_3}} N_1 \\
&\quad + N_1 N_2 N_3 p - \lambda_{N_3}^{\text{reflected}} \rho_{N_3} N_1 N_2 \\
&= (1 - \lambda_{N_3}^{\text{reflected}}) \rho_{N_3} N_1 N_2 \\
&\quad - \lambda_{N_2, \rho_{N_3}}^{\text{reflected}} \rho_{N_2, \rho_{N_3}} N_1 \\
&\quad + N_1 N_2 N_3 p \\
&\leq (1 - \lambda_{N_2}^{\text{reflected}}) \rho_{N_2} N_1 N_3 \\
&\quad - \lambda_{N_3, \rho_{N_2}}^{\text{reflected}} \rho_{N_3, \rho_{N_2}} N_1 \\
&\quad + N_1 N_2 N_3 p + \lambda_{N_2}^{\text{reflected}} \text{ (by Lemma 6)} \\
&= N_1 N_3 \rho_{N_2} - \lambda_{N_3, \rho_{N_2}}^{\text{reflected}} \rho_{N_3, \rho_{N_2}} N_1 \\
&\quad + N_1 N_2 N_3 p - \lambda_{N_2}^{\text{reflected}} \rho_{N_2} N_1 N_3 \\
&= T_{N_1, N_3, \rho_{N_2}}^{\text{reflected}} + T_{N_1 N_3, N_2, p}^{\text{reflected}}.
\end{aligned}$$

This proves the result.  $\square$

#### D. A Related NP-Completeness Result

In § 4.2 we showed it is NP-hard to order columns so as to minimize the RUNCOUNT value after lexicographic sorting. We now show a related problem is NP-complete.

*Column-Ordering-for-Minimax Lexicographic Runcount (COMLR).* Given a table  $T$ , an ordering on the values found in each column, and an integer  $K$ , is it possible to reorder the columns of the table, such that when the reordered table is lexicographically sorted, no column has more than  $K$  runs?

**Proposition 7.** *COMLR is NP-complete.*

**Proof.** Membership in NP is obvious. We show COMLR is NP-hard by reduction from 3SAT [23, LO2]. Suppose our 3SAT instance has variables  $v_1$  to  $v_{|V|}$  and clauses  $C_1$  to  $C_m$ . We assume that no clause contains both a variable and its negation because such a clause can be removed without affecting satisfiability.

For every variable  $v_i$ , the COMLR instance has three values that can appear in tables:  $w_i$ ,  $\bar{w}_i$  and  $0_{w_i}$ . They are ordered:  $w_i < \bar{w}_i < 0_{w_i}$ . Moreover, for  $a \in \{w_i, \bar{w}_i, 0_{w_i}\}$ ,  $b \in \{w_j, \bar{w}_j, 0_{w_j}\}$  and  $i \neq j$ , we have  $a < b$  if and only if  $i < j$ .

Two other values are used in the table,  $+\infty$  and  $-\infty$  whose orderings with respect to the other values are as expected.

We construct a table  $T$ , with  $3|V| + 2$  rows, and with a column for each possible literal and a column for each clause. Hence  $T$  has  $2|V| + m$  columns. We describe the columns from left to right, beginning with the columns for  $\bar{v}_1$  and  $v_1$ . See Fig. 12.

Consider the literal column associated with  $\bar{v}_1$ . It begins with a run of length  $3 \times 1 - 2$  with the  $-\infty$  value. It then contains  $\bar{w}_1, \bar{w}_1, 0_{w_1}$ . The remainder of the column is composed of  $+\infty$ . The next column is for  $v_1$ . It begins and ends similarly, but in the middle it has  $w_1, 0_{w_1}, w_1$ . The pairs of columns for the remaining variables then follow. The column for  $\bar{v}_i$  begins with a run containing  $3i - 2$  copies of the  $-\infty$  value, then has  $\bar{w}_i, \bar{w}_i, 0_{w_i}$ , whereas the column for  $v_i$  has  $w_i, 0_{w_i}, w_i$  between the run of  $-\infty$  and the run of  $+\infty$ . Thus, the left part of the table has blocks of size  $3 \times 2$  arranged diagonally. Above the diagonal, we have  $-\infty$ ; below the diagonal, we have  $+\infty$ . (Except that there is a row of  $-\infty$  above everything and a row of  $+\infty$  below everything.)

To complete the construction, we have one column per clause. Consider a clause  $\{l_i, l_j, l_k\}$  where  $l_i = v_i$  or  $l_i = \bar{v}_i$  and similarly for  $l_j$  and  $l_k$ . Each column begins with  $-\infty$  and ends with  $+\infty$ . Otherwise, the column copies the column for  $l_i$  within the *zone* of  $v_i$ , where the zone of variable  $v_i$  consists of rows  $3i - 2, 3i - 1, 3i$  in the table. The construction is such that no matter how columns are reordered, a lexicographic sort can rearrange rows only within their zones. Similarly, the column copies the columns for  $l_j$  and  $l_k$  within the zones of  $v_j$  and  $v_k$ , respectively. Otherwise, the part of the column that is in the zone of  $w_l$  ( $l \notin \{i, j, k\}$ ), contains  $0_{w_l}$ . See Fig. 12 for the table constructed for  $\{\{v_1, \bar{v}_2, v_3\}, \{\bar{v}_1, \bar{v}_2, v_3\}, \{\bar{v}_1, \bar{v}_3, v_4\}, \{v_1, v_3, v_4\}\}$ . Finally, we set the maximum-runs-per-column bound  $K = |V| + 7$ .

The construction creates literal columns that cannot have many runs no matter how we reorder columns and lexicographically sort the rows. Consequently these columns always meet the  $|V| + 7$  bound. For clause columns: after *any* column permutation and lexicographic sorting, a clause column can have at most  $|V| + 8$  runs:

- 2 for the  $-\infty$  and the  $+\infty$ ,
- $(|V| - 3)$  for the variables that are not in the clause,
- and at most 3 for each of the 3 variables that are in the clause.

Table  $T$  can have its columns reordered to have at most  $|V| + 7$  runs per column (after lexicographic sorting), if and only if the given instance of 3SAT is satisfiable.

Suppose we have a satisfying truth assignment. If  $v_i$  is true, permute the columns for  $\bar{v}_i$  and  $v_i$ . (Otherwise, leave them alone.) After permuting these columns, lexicographic sorting would swap the bottom two rows in the zone for  $v_i$ . Any clause containing  $v_i$  would find that this swap merges two runs of  $w_i$  in its column, and thus we would meet the  $|V| + 7$  bound for that clause's column. Likewise, if  $v_i$  is false, leave the two columns in their original relationship. The table as constructed was lexicographically sorted, and any clause containing  $\bar{v}_i$  would continue to have a run of  $\bar{w}_i$ 's and meet the run bound. Since we have a satisfying truth assignment, every clause column will contain at least one such run.

Conversely, suppose we have permuted table columns such that the lexicographically sorted table has no column with more than  $|V| + 7$  runs. Because

$\bar{v}_1$	$v_1$	$\bar{v}_2$	$v_2$	$\bar{v}_3$	$v_3$	$\bar{v}_4$	$v_4$	$c_1$	$c_2$	$c_3$	$c_4$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\bar{w}_1$	$w_1$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$w_1$	$\bar{w}_1$	$\bar{w}_1$	$w_1$
$\bar{w}_1$	$0_{w_1}$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$0_{w_1}$	$\bar{w}_1$	$\bar{w}_1$	$0_{w_1}$
$0_{w_1}$	$w_1$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$w_1$	$0_{w_1}$	$0_{w_1}$	$w_1$
$+\infty$	$+\infty$	$\bar{w}_2$	$w_2$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$\bar{w}_2$	$\bar{w}_2$	$0_{w_2}$	$0_{w_2}$
$+\infty$	$+\infty$	$\bar{w}_2$	$0_{w_2}$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$\bar{w}_2$	$\bar{w}_2$	$0_{w_2}$	$0_{w_2}$
$+\infty$	$+\infty$	$0_{w_2}$	$w_2$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$0_{w_2}$	$0_{w_2}$	$0_{w_2}$	$0_{w_2}$
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$\bar{w}_3$	$w_3$	$-\infty$	$-\infty$	$w_3$	$w_3$	$\bar{w}_3$	$w_3$
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$\bar{w}_3$	$0_{w_3}$	$-\infty$	$-\infty$	$0_{w_3}$	$0_{w_3}$	$\bar{w}_3$	$0_{w_3}$
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$0_{w_3}$	$w_3$	$-\infty$	$-\infty$	$w_3$	$w_3$	$0_{w_3}$	$w_3$
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$\bar{w}_4$	$w_4$	$0_{w_4}$	$0_{w_4}$	$w_4$	$w_4$
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$\bar{w}_4$	$0_{w_4}$	$0_{w_4}$	$0_{w_4}$	$0_{w_4}$	$0_{w_4}$
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$0_{w_4}$	$w_4$	$0_{w_4}$	$0_{w_4}$	$w_4$	$w_4$
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Figure 12: Example construction for  $\{c_1, c_2, c_3, c_4\}$ , where  $c_1 = \{v_1, \bar{v}_2, v_3\}$ ,  $c_2 = \{\bar{v}_1, \bar{v}_2, v_3\}$ ,  $c_3 = \{\bar{v}_1, \bar{v}_3, v_4\}$ , and  $c_4 = \{v_1, v_3, v_4\}$ .

lexicographic sorting is restricted to rearranging rows only within their zones, a clause's column must contain a length-two run of  $w_i$  or  $\bar{w}_i$ , for some  $1 \leq i \leq |V|$ . The construction guarantees that if any clause column contains a length-two run of  $w_i$ , then no column contains a length-two run of  $\bar{w}_i$ . Similarly, a length-two run of  $\bar{w}_i$  precludes a length-two run of  $w_i$ . Moreover, by construction we see that a column containing the length-two run of  $w_i$  must contain  $v_i$ . Hence, we set  $v_i$  to true. Likewise, for any run of  $\bar{w}_i$  we set  $v_i$  to false. Clearly, this truth setting satisfies the original 3SAT instance.  $\square$