To appear in *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms,* San Francisco, CA

Improving Table Compression with Combinatorial Optimization

Adam L. Buchsbaum* Glenn S. Fowler* Raffaele Giancarlo†

Abstract

We study the problem of compressing massive tables within the partition-training paradigm introduced by Buchsbaum et al. [SODA'00], in which a table is partitioned by an off-line training procedure into disjoint intervals of columns, each of which is compressed separately by a standard, on-line compressor like gzip. We provide a new theory that unifies previous experimental observations on partitioning and heuristic observations on column permutation, all of which are used to improve compression rates. Based on the theory, we devise the first on-line training algorithms for table compression, which can be applied to individual files, not just continuously operating sources; and also a new, off-line training algorithm, based on a link to the asymmetric traveling salesman problem, which improves on prior work by rearranging columns prior to partitioning. We demonstrate these results experimentally. On various test files, the on-line algorithms provide 35–55% improvement over gzip with negligible slowdown; the off-line reordering provides up to 20% further improvement over partitioning alone. We also show that a variation of the table compression problem is MAX-SNP hard.

1 Introduction

Table compression was introduced by Buchsbaum et al. [3] as a unique application of compression, based on several distinguishing characteristics. Tables are collections of fixed-length records and can grow to terabytes in size. They are often generated by continuously operating sources and can contain much redundancy. An example is a data warehouse at AT&T that each month stores one billion records pertaining to voice phone activity. Each record is several hundred bytes long and contains information about endpoint exchanges, times and durations of calls, tariffs, etc.

The goals of table compression are to be fast, on-line, and effective: eventual compression ratios of 100:1 or better are desirable. Storage reduction is an obvious benefit, but perhaps more important is the bandwidth savings realized during subsequent transmission. Tables of transactions, like phone calls and credit card usage, are typically stored once but shipped repeatedly to different parts of an organization: for fraud detection, billing, operations support, etc.

Prior work [3] distinguishes tables from general databases. Tables are written once and read many times, while databases are subject to dynamic updates. Fields in table records are fixed length, and records tend to be homogeneous; database records often contain intermixed fixed- and variable-length fields. Finally, the goals of compression differ. Database compression stresses index preservation, the ability to retrieve an arbitrary record, under compression [6]. Tables are typically not indexed at individual records; rather, they are scanned in toto by downstream applications.

Consider each record in a table to be a row in a matrix. A naive method of table compression is to compress the string derived from scanning the table in row-major order. Buchsbaum et al. [3] observe experimentally that partitioning the table into contiguous intervals of columns and compressing each interval separately in this fashion can achieve significant compression improvement. The partition is generated by a one-time, off-line training procedure, and the resulting compression strategy is applied on-line to the table. They also observe heuristically that certain rearrangements of the columns prior to partitioning further improve compression, by grouping dependent columns.

We generalize the partitioning approach into a unified theory that explains both contiguous partitioning and column rearrangement. The theory applies to a set of variables with a given, abstract notion of combination and cost; table compression is a concrete case. To test the theory, we design new algorithms for contiguous partitioning, which speed training to work on-line on single files in addition to off-line on continuously generated tables; and for reordering in the off-line training paradigm, which improves the compression rates achieved from contiguous partitioning alone. Experimental results support these conclusions. Before summarizing the results, we motivate the theoretical insights by considering the relationship between entropy and compression.

1.1 Compressive Estimates of Entropy.

Let $\mathcal{C}$ be a compression algorithm and $\mathcal{C}(x)$ its output on a string x. A large body of work in information theory establishes the existence of many optimal compression algorithms: i.e., algorithms such that $|\mathcal{C}(x)|/|x|$, the *compression rate*, approaches the entropy of the information source emitting x. For instance, the LZ77 algorithm [18] is optimal for certain classes of

*AT&T Labs, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932, USA, {alb,gsf}@research.att.com.

†Dipartimento di Matematica ed Applicazioni, Universitá di Palermo, Via Archirafi 34, 90123 Palermo, Italy, raffaele@altair.math.unipa.it. Work partially supported by AT&T Labs and the MURST Project of National Relevance Bioinformatica e Ricerca Genomica.

sources, e.g., those that are stationary and ergodic [7].

While entropy establishes a lower bound on compression rates, it is not straightforward to measure entropy itself. One empirical method inverts the relationship and estimates entropy by applying a provably good compressor to a sufficiently long, representative string. The compression rate then becomes a *compressive estimate of entropy.* These estimates themselves become benchmarks for other compressors. Another estimate is the *empirical entropy* of a string, which is based on the probability distribution of substrings of various lengths, without any statistical assumptions regarding the emitting source. Kosaraju and Manzini [13] exploit the synergy between empirical and true entropy.

The contiguous partitioning approach to table compression [3] exemplifies the practical exploitation of compressive estimates. Each column of the table can be seen as being generated by a separate source. The contiguous partitioning scheme measures the benefit of a particular partition empirically, by compressing the table with respect to that partition and using the output size as a cost. Thus, the partitioning method uses a compressive estimate of the joint entropy among columns. Prior work [3] demonstrates the benefit of this approach.

1.2 Method and Results. We are thus motivated to study table compression in terms of compressive estimates of the joint entropy of random variables. In Section 2, we formalize and study two problems on partitioning sets of variables with abstract notions of combination and cost; joint entropy forms one example. This generalizes the approach of Buchsbaum et al. [3], who consider the contiguous case only and when applied to table compression. We develop idealized algorithms to solve these problems in the general setting. In Section 3, we apply these problems to table compression and derive two new algorithms for contiguous partitioning and one new algorithm for general partitioning with reordering of columns. The reordering algorithm demonstrates a link between general partitioning and the classical asymmetric traveling salesman problem. We assess algorithm performance experimentally in Section 4.

The new contiguous partitioning algorithms are meant to be fast; effect more compression than off-the-shelf compressors like gzip (LZ77); but not be as good as the optimal, contiguous partitioning algorithm. The increased training speed (compared to optimal, contiguous partitioning) makes the new algorithms usable in ad hoc settings, when training time must be factored into the overall time to compress. For files from various sources, we achieve 35–55% compression improvement with less than a 1.7-factor slowdown, both compared to gzip. For files from genetic databases, which tend to be harder to compress, the compression improvement is 5–20%, with slowdown factors of 3–8.

For several of our files, the general partitioning with reordering algorithm yields compression improvements of at least 5% compared to optimal, contiguous partitioning without reordering, which itself improves over gzip by 20–50% for our files. In some cases, the additional improvement approaches 20%. Additional evidence suggests that the algorithm is nearly optimal (among partitioning algorithms). While training time can be ignored in the off-line training paradigm, we show the additional time for reordering is not significant.

Finally, in Section 5, we give some complexity results that link table compression to the classical shortest common superstring problem. We show that an orthogonal (column-major) variation of table compression is MAX-SNP hard when LZ77 is the underlying compressor. On the other hand, while we also show that the row-major problem is MAX-SNP hard when run length encoding (RLE) is the underlying compressor, we prove that the column-major variation for RLE is solvable in polynomial time. We conclude with open problems and directions in Section 6.

2 Partitions of Variables with Entropy-Like Functions

Let $X = \{x_1, \ldots, x_n\}$ be a set of discrete variables over some domain $\mathcal{D}$, and consider some *cost* function $H : \mathcal{D}^* \to \Re$. We use $H(X, Y)$ as a shorthand for $H(Z)$, where Z is the set composed of all the elements in X and Y: if X and Y are sets, then $Z = X \cup Y$; if X and Y are variables, then $Z = \{X, Y\}$; etc. For some partition $\mathcal{P}$ of X into subsets, define $H(\mathcal{P}) = \sum_{Y \in \mathcal{P}} H(Y)$. We are interested in the relationship between $H(X)$ and $H(\mathcal{P})$. For example, let X be a vector of random variables with joint probability distribution $p(X)$. Two vectors X and Y are *statistically independent* if and only if $p(x, y) = p(x)p(y)$, for all $\{x, y\}$; otherwise, X and Y are *statistically dependent*. Let $H(X) = -\sum_{\{x_1, \ldots, x_n\}} p(x_1, \ldots, x_n) \log p(x_1, \ldots, x_n)$ be the *joint entropy* of X. Then it is well known [7] that for any partition $\mathcal{P}$ of X, $H(X) \leq H(\mathcal{P})$, with equality if and only if the subsets in P are mutually independent.

To generalize to systems of variables with other cost functions, we introduce the following definitions. We call an element of $\mathcal{P}$, which is a subset of X, a *class*. We define two variables or sets of variables X and X' to be *combinatorially dependent* if $H(X, X') < H(X) + H(X')$; otherwise, X and X' are *combinatorially independent*. When $H(\cdot)$ is the entropy function over random variables, combinatorial dependence becomes statistical dependence. Considering unordered sets implies that $H(X, X') = H(X', X)$. Note that in general it is possible that $H(X, X') > H(X) + H(X')$, although not when $H(\cdot)$ is the entropy function over random variables. Finally, we define a class Y to be *contiguous* if $x_i \in Y$ and $x_j \in Y$ for any $i < j$ implies that $x_{i+1} \in Y$ and a partition $\mathcal{P}$ to be *contiguous* if each $Y \in \mathcal{P}$ is contiguous. We now define two problems of finding optimal partitions of T.

PROBLEM 2.1. *Find a contiguous partition $\mathcal{P}$ of X minimizing $H(\mathcal{P})$ among all such partitions.*

PROBLEM 2.2. *Find a partition $\mathcal{P}$ of X minimizing $H(\mathcal{P})$ among all partitions.*

Clearly, a solution to Problem 2.2 is at least as good in terms of cost as one to Problem 2.1. Problem 2.1 has a simple, fast algorithmic solution, however. Problem 2.2, while seemingly intractable, has an algorithmic heuristic that seems to work well in practice.

Assume first that combinatorial dependence is an equivalence relation on X. This is not necessarily true in practice, but we study the idealized case to provide some intuition for handling real instances, when we cannot determine combinatorial dependence or calculate the true cost function directly.

LEMMA 2.1. *If combinatorial dependence is an equivalence relation on X, then the partition $\mathcal{P}$ of X into equivalence classes $C_1, \ldots, C_k$ solves Problem 2.2.*

PROOF. Consider some partition $\mathcal{P}' \neq \mathcal{P}$; we show that $H(\mathcal{P}) \leq H(\mathcal{P}')$. Assume there exists a class $C' \in \mathcal{P}'$ such that $C' \supset C_i$ for some $1 \leq i \leq k$. Partition C' into subclasses $C'_1, \ldots, C'_\ell$ such that for each C'_j there is some C_i such that $C'_j \subseteq C_i$. Let $\mathcal{P}'' = (\mathcal{P}' \setminus \{C'\}) \cup \{C'_1, \ldots, C'_\ell\}$. Since the C_i's are equivalence classes, the C'_j's are mutually independent, so $H(C') \geq \sum_{j=1}^{\ell} H(C'_j)$, which implies $H(\mathcal{P}'') \leq H(\mathcal{P}')$. Set $\mathcal{P}' \leftarrow \mathcal{P}''$, and iterate.

If no such C' exists in $\mathcal{P}'$, then either $\mathcal{P}' = \mathcal{P}$, and we are done, or else $\mathcal{P}'$ contains two classes C' and D' such that $C' \cup D' \subseteq C_i$ for some i. The elements in C' and D' are mutually dependent, so $H(C', D') < H(C') + H(D')$. Unite each such pair of classes until $\mathcal{P}' = \mathcal{P}$. □

Lemma 2.1 gives a simple algorithm for solving Problem 2.2 when combinatorial dependence is an equivalence relation that can be computed: partition X according to the induced equivalence classes. When combinatorial dependence is not an equivalence relation, or when we can only calculate $H(\cdot)$ heuristically, we seek other approaches.

2.1 Solutions Without Reordering. In the general case, we can solve Problem 2.1 by dynamic programming. Let $E[i]$ be the cost of an optimal, contiguous partition of $x_1, \ldots, x_i$. $E[n]$ is thus the cost of a solution to Problem 2.1. Define $E[0] = 0$; then, for $1 \leq i \leq n$,

$$E[i] = \min_{0 \leq j < i} E[j] + H(x_{j+1}, \ldots, x_i). \tag{2.1}$$

The actual partition with cost $E[n]$ can be maintained by standard dynamic programming backtracking.

If combinatorial dependence actually is an equivalence relation and all dependent variables appear contiguously in X, a simple greedy algorithm also solves the problem. Start with class $C_1 = \{x_1\}$. In general, let i be the index of the current class and j be the index of the variable most recently added to C_i. While $j < n$, iterate as follows. If $H(C_i \cup \{x_{j+1}\}) < H(C_i) + H(x_{j+1})$, then set $C_i \leftarrow C_i \cup \{x_{j+1}\}$; otherwise, start a new class, $C_{i+1} = \{x_{j+1}\}$. An alternative algorithm assigns, for $1 \leq i < n$, x_i and x_{i+1} to the same class if and only if $H(x_i, x_{i+1}) < H(x_i) + H(x_{i+1})$. We call the resulting partition a greedy partition; formally, a *greedy partition* is one in which each class is a maximal, contiguous set of mutually dependent variables.

LEMMA 2.2. *If combinatorial dependence is an equivalence relation and all combinatorially dependent variables appear contiguously in X, then the greedy partition solves Problems 2.1 and 2.2.*

PROOF. By assumption, the classes in a greedy partition correspond to the equivalence classes of X. Lemma 2.1 thus shows that the greedy partition solves Problem 2.2. Contiguity therefore implies it also solves Problem 2.1. □

2.2 Solutions with Reordering. While Problem 2.2 seems intractable, we give a combinatorial approach that admits a practical heuristic. Define a weighted, complete, undirected graph, $G(X)$, with a vertex for each $x_i \in X$; the weight of edge $\{x_i, x_j\}$ is $w(x_i, x_j) = \min(H(x_i, x_j), H(x_i) + H(x_j))$. Let $P = (v_0, \ldots, v_\ell)$ be any path in $G(X)$. The *weight* of P is $w(P) = \sum_{i=0}^{\ell-1} w(v_i, v_{i+1})$. We apply the cost function $H(\cdot)$ to define the cost of P. Consider removing all edges $\{u, v\}$ from P such that u and v are combinatorially independent. This leaves a set of disjoint paths, $\mathcal{S}(P) = \{P_1, \ldots, P_k\}$ for some k. We define the *cost* of P to be $H(P) = \sum_{i=1}^{k} H(P_i)$, where P_i is the unordered set of vertices in the corresponding subpath. If P is a tour of $G(X)$, then $\mathcal{S}(P)$ corresponds to a partition of X.

We establish a relationship between the cost and weight of a tour P. Assume there are two distinct paths $P_i = (u_0, \ldots, u_k)$ and $P_j = (v_0, \ldots, v_\ell)$ in $\mathcal{S}(P)$ such that u_k and v_0 are combinatorially dependent and v_0 follows u_k in P. In P exist the edges $\{u_k, x\}$, $\{y, v_0\}$, and $\{v_\ell, z\}$. We can transform P into a new tour P' that unites P_i and P_j by substituting for these three edges the new edges: $\{u_k, v_0\}$, $\{v_\ell, x\}$, and $\{y, z\}$. We call this a *path coalescing transformation*. The following shows that it is like the standard traveling salesman 3-opt transformation, in that it always reduces the cost of a tour. It is restricted, as u_k and v_0 must be combinatorially dependent.

LEMMA 2.3. *If P' is formed from P by a path coalescing transformation, then $w(P') < w(P)$.*

PROOF (SKETCH). The reduction in weight is at least $H(u_k) + H(v_0) - w(u_k, v_0) > 0$. □

Repeated path coalescing groups combinatorially dependent variables. If a tour P admits no path coalescing

transformation, and if combinatorial dependence is an equivalence relation on X, then we can conclude that P is optimal by Lemma 2.1. That is, $\mathcal{S}(P)$ corresponds to an optimal partition of X, which solves Problem 2.2. Furthermore, Lemma 2.3 implies that a minimum weight tour P admits no path coalescing transformation.

When $H(\cdot)$ is sub-additive, i.e., $H(X,Y) \leq H(X) + H(Y)$, as is the entropy function, a sequence of path coalescing transformations yields a sequence of paths of non-increasing costs. That is, in Lemma 2.3, $w(P') < w(P)$ *and* $H(P') \leq H(P)$. We explore this connection between the two functions below, when we do not assume that combinatorial dependence is an equivalence relation or even that $H(\cdot)$ is sub-additive.

3 Partitions of Tables and Compression

We apply the results of Section 2 to table compression. Let T be a table of $n = |T|$ columns and some fixed, arbitrary number of rows. Let $T[i]$ denote the i'th column of T. Given two tables T_1 and T_2, let T_1T_2 be the table formed by their juxtaposition. That is, $T = T_1T_2$ is defined so that $T[i] = T_1[i]$ for $1 \leq i \leq |T_1|$ and $T[i] = T_2[i - |T_1|]$ for $|T_1| < i \leq |T_1| + |T_2|$. Any column is a one-column table, so $T[i]T[j]$ is the table formed by projecting the i'th and j'th columns of T; and so on. We use the shorthand $T[i,j]$ to represent the projection $T[i] \cdots T[j]$ for some $j \geq i$.

Fix a compressor $\mathcal{C}$: we use gzip, based on LZ77 [18]. Let $H_{\mathcal{C}}(T)$ be the size of the result of compressing table T as a string in row-major order using $\mathcal{C}$. Let $H_{\mathcal{C}}(T_1, T_2) = H_{\mathcal{C}}(T_1T_2)$. $H_{\mathcal{C}}(\cdot)$ is a cost function as discussed in Section 2, and the definitions of combinatorial dependence and independence apply to tables. In particular, two tables T_1 and T_2, which might be projections of columns from a common table T, are combinatorially dependent if compressing them together is better than compressing them separately and combinatorially independent otherwise.

Problems 2.1 and 2.2 now apply to compressing T. Problem 2.1 is to find a contiguous partition of T into intervals of columns minimizing the overall cost of compressing each interval separately. Problem 2.2 is to find a partition of T, allowing columns to be reordered, minimizing the overall cost of compressing each interval separately. Buchsbaum et al. [3] address Problem 2.1 experimentally and leave Problem 2.2 open save for some heuristic observations.

A few major issues arise in this application. Combinatorial dependence is not necessarily an equivalence relation. It is not necessarily even symmetric, so we can no longer ignore the order of columns in a class. Also, $H_{\mathcal{C}}(\cdot)$ need not be sub-additive. If $\mathcal{C}$ behaves according to entropy, however, then intuition suggests that our partitioning strategies will improve compression. Stated conversely, if $H_{\mathcal{C}}(T)$ is far from $H(T)$, the entropy of T as defined by Ziv and Lempel [18], there should be some partition P of T so that $H_{\mathcal{C}}(P)$ approaches $H(T)$, which is a lower bound on $H_{\mathcal{C}}(T)$. We will present algorithms for solving these problems and experiments assessing their performance.

3.1 Algorithms for Table Compression without Rearrangement of Columns. The dynamic program in Equation (2.1) finds an optimal, contiguous partition solving Problem 2.1. Buchsbaum et al. [3] demonstrate experimentally that it effectively improves compression results, and we will use their method as a benchmark. The dynamic program, however, requires $\Theta(n^2)$ steps, each applying $\mathcal{C}$ to an average of $\Theta(n)$ columns, for a total of $\Theta(n^3)$ column compressions. In the off-line training paradigm, this optimization time can be ignored. Faster algorithms, however, might allow some partitioning to be applied when compressing single, tabular files in addition to continuously generated tables.

The greedy algorithms from Section 2.1 apply directly in our framework. We denote by GREEDY the algorithm that grows class C_i incrementally by comparing $H_{\mathcal{C}}(C_iT[j+1])$ and $H_{\mathcal{C}}(C_i) + H_{\mathcal{C}}(T[j+1])$. We denote by GREEDYT the algorithm that assigns $T[i]$ and $T[i+1]$ to the same class when $H_{\mathcal{C}}(T[i,i+1]) < H_{\mathcal{C}}(T_i) + H_{\mathcal{C}}(T[i+1])$.

GREEDY performs $2(n-1)$ compressions, each of $\Theta(n)$ columns, for a total of $\Theta(n^2)$ column compressions. GREEDYT performs $2(n-1)$ compressions, each of one or two columns, for a total of $\Theta(n)$ column compressions, asymptotically at least as fast as applying $\mathcal{C}$ to T itself.

While combinatorial dependence is not an equivalence relation, we hypothesize that GREEDY and GREEDYT will produce partitions close in cost to the optimal contiguous partition produced by the dynamic program. We present experimental results testing this hypothesis in Section 4.

3.2 Algorithms for Table Compression with Rearrangement of Columns. We now consider Problem 2.2. Assuming that combinatorial dependence is not an equivalence relation, to the best of our knowledge, the only known algorithm to solve it exactly consists of generating all $n!$ column orderings and applying the dynamic program in Equation (2.1) to each. The relationship between compression and entropy, however, suggests that the approach in Section 2.2 can still be fruitfully applied.

Recall that in the idealized case, an optimal solution corresponds to a tour of $G(T)$ that admits no path coalescing transformation. Furthermore, such transformations always reduce the weight of such tours. The lack of symmetry in $H_{\mathcal{C}}(\cdot)$ further suggests that order within classes is important: it no longer suffices to coalesce paths globally.

We therefore hypothesize a strong, positive correlation between tour weight and compression cost. This would imply that a traveling salesman (TSP) tour of $G(T)$ would yield an optimal or near-optimal partition of T. To test this hypothesis, we generate a set of tours of various weights,

by iteratively applying standard optimizations (e.g., 3-opt, 4-opt). Each tour induces an ordering of the columns, which we optimally partition using the dynamic program. We present results of this experiment in Section 4.

4 Experiments

We report experimental results on several data sets. CARE contains 90-byte records from a customer care database of voice call activity. NETWORK contains 32-byte records from a system of network status monitors. CENSUS is a portion of the U.S. *1990 Census of Population and Housing Summary Tape File 3A* [4]. We used field group 301, level 090, for all states. Each record is 932 byes. LERG is from Telcordia's database describing local telephone switches. Each record was padded as necessary to a uniform 30 bytes. CARE, NETWORK, and CENSUS were used by Buchsbaum et al. [3].

We also use several files from genetic databases, which pose unique challenges to compression [9, 15]. These files can be viewed as two-dimensional, alphanumeric tables representing multiple alignments of proteins (amino acid sequences) and genomic coding regions (DNA sequences). EGF, LRR, PF00032, BACKPQQ, CALLAGEN, and CBS come from the PFAM database of multiple alignments of protein domains or conserved protein functions [1]. We chose tables of different sizes and representing protein domains with differing degrees of conservation: i.e., how closely two members of a family match characters in the alignment. CYTOB is from the AMMTDB database of multi-aligned sequences of Vertebrate mitochondrial genes for coding proteins [14] and is much wider than the other files.

Table 1 details the sizes of the files and how well gzip and the optimal partition via dynamic programming (using gzip as the underlying compressor) compress them. We use the pin/pzip system described by Buchsbaum et al. [3] to general optimal, contiguous partitions. For each file, we run the dynamic program on a small *training set* and compress the remainder of the data, the *test set*. Gzip results are with respect to the test sets only. Buchsbaum et al. [3] investigate the relationship between training size and compression performance and demonstrate a threshold after which more training data does not improve performance. Here we simply use enough training data to exceed this threshold and report this amount in Table 1. The training and test sets remain disjoint to support the validity of using a partition from a small amount of training data on a larger amount of subsequent data. In a real application, the training data would also be compressed. All experiments were performed on one 250 MHz IP27 processor in a 24-processor SGI Challenge, with 14 GB of main memory. Each time reported is the medians of five runs.

4.1 Greedy Algorithms. Our hypothesis that GREEDY and GREEDYT produce partitions close in cost to that of the optimal, contiguous partition, if true implies that we can substitute the greedy algorithms for the dynamic program (DP) in purely on-line applications that cannot afford off-line training time. We thus compare compression rates of GREEDY and GREEDYT against DP and gzip, to assess quality of the partitions; and we compare the time taken by GREEDY and GREEDYT (partitioning and compression) against gzip, to assess tractability. Table 2 shows the compressed sizes using partitions computed with GREEDY and GREEDYT. Table 3 gives the time results.

GREEDY compresses to within 2% of DP on seven of the files, including four of the genetic files. It is never more than 9% bigger than DP, and with the exception of BACKPQQ, always outperforms gzip. GREEDYT comes within 10% of DP on seven files, including four genetic files and outperforms gzip except on BACKPQQ and CYTOB. Both GREEDY and GREEDYT seem to outperform DP on CALLAGEN, although this would seem theoretically impossible. It is an artifact of the training/testing paradigm: we compress data distinct from that used to build the partitions.

Tables 2 and 3 show that in many cases, the greedy algorithms provide significant extra compression at acceptable time penalties. For the non-genetic files, greedy partitioning compression is less than 1.7 slower than gzip yet provides 35–55% more compression. For the genetic files, the slowdown is a factor of 3–8, and the extra compression is 5–20% (ignoring BACKPQQ). Thus, the greedy algorithms provide a good on-line heuristic for improving compression.

4.2 Reordering via TSP. Our hypothesis that tour weight and compression are correlated implies that generating a TSP tour (or approximation) would yield an optimal (or near optimal) partition. Although we do not know what the optimal partition is for our files, we can assess the correlation by generating a sequence of tours and, for each, measuring the resulting compression. We also compare the compression using the best partition from the sequence against that using DP on the original ordering, to gauge the improvement yielded by reordering.

For each file, we computed various tours on the corresponding graph $G(\cdot)$. We computed a close approximation to a TSP tour using a variation of Zhang's branch-and-bound algorithm [17], discussed by Cirasella et al. [5]. We also computed a 3-opt local optimum tour; and we used a 4-opt heuristic to compute a sequence of tours of various costs. Each tour induced an ordering of the columns. For each column ordering, we computed the optimal, contiguous partition by DP, except that we used GREEDYT on the orderings for CENSUS, due to computational limitations. Figure 1 plots the results for CARE, NETWORK, CENSUS, LERG, BACKPPQ, and CYTOB; plots for the remaining files have similar characteristics and will appear in the full paper.

The plots demonstrate a strong, positive correlation be-

Table 1: Files used. Bpr is bytes per record. Size is the original size of the file in bytes. Training size is the ratio of the size of the training set to that of the test set. Gzip and DP report compression results; DP is the optimal contiguous partition, calculated by dynamic programming. For each, Size is the size of the compressed file in bytes, and Rate is the ratio of compressed to original size. DP/Gzip shows the relative improvement yielded by partitioning.

			Training	Gzip		DP		
File	Bpr	Size	Size	Size	Rate	Size	Rate	DP/Gzip
care	90	8181810	0.0196	2036277	0.2489	1290936	0.1578	0.6340
network	126	60889500	0.0207	3749625	0.0616	1777790	0.0292	0.4741
census	932	332959796	0.0280	30692815	0.0922	21516047	0.0646	0.7010
lerg	30	3480030	0.0862	454975	0.1307	185856	0.0534	0.4085
EGF	188	533920	0.0690	72305	0.1354	56571	0.1060	0.7824
LRR	72	235440	0.0685	61745	0.2623	49053	0.2083	0.7944
PF00032	176	402512	0.0673	34225	0.0850	30587	0.0760	0.8937
backPQQ	81	22356	0.0507	7508	0.3358	7186	0.3214	0.9571
callagen	112	242816	0.0678	67338	0.2773	59345	0.2444	0.8813
cbs	134	73834	0.0635	23207	0.3143	19839	0.2687	0.8549
cytoB	1225	579425	0.0592	109681	0.1893	89983	0.1553	0.8204

Table 2: Performance of GREEDY and GREEDYT. For each, Size is the size of the compressed file using the corresponding partition; Rate is the corresponding compression rate; /Gzip is the size relative to gzip; and /DP is the size relative to using the optimal, contiguous partition.

	GREEDY				GREEDYT			
File	Size	Rate	/Gzip	/DP	Size	Rate	/Gzip	/DP
care	1307781	0.1598	0.6422	1.0130	1360160	0.1662	0.6680	1.0536
network	1784625	0.0293	0.4759	1.0038	2736366	0.0449	0.7298	1.5392
census	21541616	0.0647	0.7018	1.0012	21626399	0.0650	0.7046	1.0051
lerg	197821	0.0568	0.4348	1.0644	199246	0.0573	0.4379	1.0720
EGF	57016	0.1068	0.7885	1.0079	61178	0.1146	0.8461	1.0814
LRR	49778	0.2114	0.8062	1.0148	49393	0.2098	0.8000	1.0069
PF00032	31037	0.0771	0.9069	1.0147	31390	0.0780	0.9172	1.0263
backPQQ	7761	0.3472	1.0337	1.0800	7761	0.3472	1.0337	1.0800
callagen	58952	0.2428	0.8755	0.9934	56313	0.2319	0.8363	0.9489
cbs	21571	0.2922	0.9295	1.0873	21939	0.2971	0.9454	1.1059
cytoB	94128	0.1625	0.8582	1.0461	113160	0.1953	1.0317	1.2576

tween tour cost and compression performance. In particular, each plot shows that the least-cost tour (produced by Zhang's algorithm) produced the best compression result. Table 4 details the compression improvement from using the Zhang ordering. In five files, Zhang gives an extra improvement of at least 5% over DP on the original order; for CYTOB, the improvement is 20%. That the Zhang ordering for NETWORK underperforms the original order is again an artifact of the training/test paradigm. Figure 1 shows that the tour-cost/compression correlation remains strong for this file.

Table 4 also displays the time spent computing Zhang's tour for each file. This time is negligible compared to the time to compute the optimal, contiguous partition via DP. (The DP time on CENSUS is 168531 seconds, four orders of magnitude larger. For CYTOB, the DP time is 8640 seconds, an order of magnitude larger.) Finally, Table 4 shows that Zhang's tour always had cost close to the Held-Karp lower bound [11, 12] on the cost of the optimum TSP tour.

For off-line training, therefore, it seems that computing a good approximation to the TSP reordering before partitioning contributes significant compression improvement at minimal time cost. Furthermore, the correlation between tour cost and compression behaves similarly to what the theory in Section 2.2 would predict if $H_{\mathcal{C}}(\cdot)$ were sub-additive, which suggests the existence of some other, similar structure induced by $H_{\mathcal{C}}(\cdot)$ that would control this relationship.

Table 3: On-line performance of GREEDY and GREEDYT. For each, Time is the time in seconds to compute the partition and compress the file; /Gzip is the time relative to gzip.

	Gzip	GREEDY		GREEDYT	
File	Time	Time	/Gzip	Time	/Gzip
care	5.0260	7.1020	1.4131	6.4340	1.2801
network	15.0000	25.3790	1.6919	24.2750	1.6183
census	126.6450	160.7960	1.2697	147.1980	1.1623
lerg	1.5730	2.2800	1.4495	2.3080	1.4673
EGF	0.2350	0.8030	3.4170	0.7250	3.0851
LRR	0.1260	0.4530	3.5952	0.4450	3.5317
PF00032	0.1320	0.8950	6.7803	0.6290	4.7652
backPQQ	0.0180	0.3090	17.1667	0.3260	18.1111
callagen	0.2500	0.6050	2.4200	0.5300	2.1200
cbs	0.0530	0.4260	8.0377	0.4020	7.5849
cytoB	0.8230	3.7330	4.5358	2.1830	2.6525

Table 4: Performance of TSP reordering. For each, Size is the size of the compressed file using the Zhang ordering and optimal, contiguous partition (for CENSUS, using the GREEDYT partition); Rate is the corresponding compression rate; /Gzip is the size relative to gzip; /DP is the size relative to using the optimal, contiguous partition on the original ordering; the quality of Zhang's tour is expressed as per cent above the Held-Karp lower bound; and Time is the time in seconds to compute the tour.

	TSP					
File	Size	Rate	/Gzip	/DP	% above HK	Time
care	1199315	0.1466	0.5890	0.9290	0.438	0.110
network	1822065	0.0299	0.4859	1.0249	0.602	0.230
census	18113740	0.0544	0.5901	0.8419	0.177	28.500
lerg	183668	0.0528	0.4037	0.9882	0.011	0.010
EGF	50027	0.0937	0.6919	0.8843	0.314	0.450
LRR	48139	0.2045	0.7796	0.9814	0.354	0.050
PF00032	29625	0.0736	0.8656	0.9685	0.211	0.510
backPQQ	7131	0.3190	0.9498	0.9923	0.196	0.050
callagen	51249	0.2111	0.7611	0.8636	0.152	0.170
cbs	19092	0.2586	0.8227	0.9623	0.187	0.210
cytoB	71529	0.1234	0.6522	0.7947	0.027	735.440

5 Complexity of Table Compression

We now introduce a framework for studying the computational complexity of several versions of table compression problems. We start with a basic problem: Given a set of strings, we wish to compute an order in which to catenate the strings into a superstring X so as to minimize the cost of compressing X using a fixed compressor $\mathcal{C}$. To isolate the complexity of finding an optimal order, we restrict $\mathcal{C}$ to prevent it from reordering the input itself.

Let $x = \sigma_1 \cdots \sigma_n$ be a string over some alphabet Σ, and let $\mathcal{C}(x)$ denote the output of $\mathcal{C}$ on input x. We allow $\mathcal{C}$ arbitrary time and space, but we require that it process x monotonically. That is, it reads the symbols of x in order; after reading each symbol, it may or may not output a string. Let $\mathcal{C}(x)_j$ be the catenation of all the strings output, in order, by $\mathcal{C}$ after processing $\sigma_1 \cdots \sigma_j$. If $\mathcal{C}$ actually outputs a (non-null) string after reading σ_j, then we require that $\mathcal{C}(x)_j$ must be a prefix of $\mathcal{C}(\sigma_1 \cdots \sigma_j y)$ for any suffix y. We assume a special end-of-string character not in Σ that implicitly terminates every input to $\mathcal{C}$. Intuitively, this restriction precludes $\mathcal{C}$ from reordering its input to improve the compression. Many compression programs used in practice work within this restriction: e.g., gzip and compress.

We use $|\mathcal{C}(x)|$ to abstract the length of $\mathcal{C}(x)$. For example, when considering LZ77 compression [18], $|\mathcal{C}(x)|$ denotes the number of phrases in the LZ77 parsing of x,

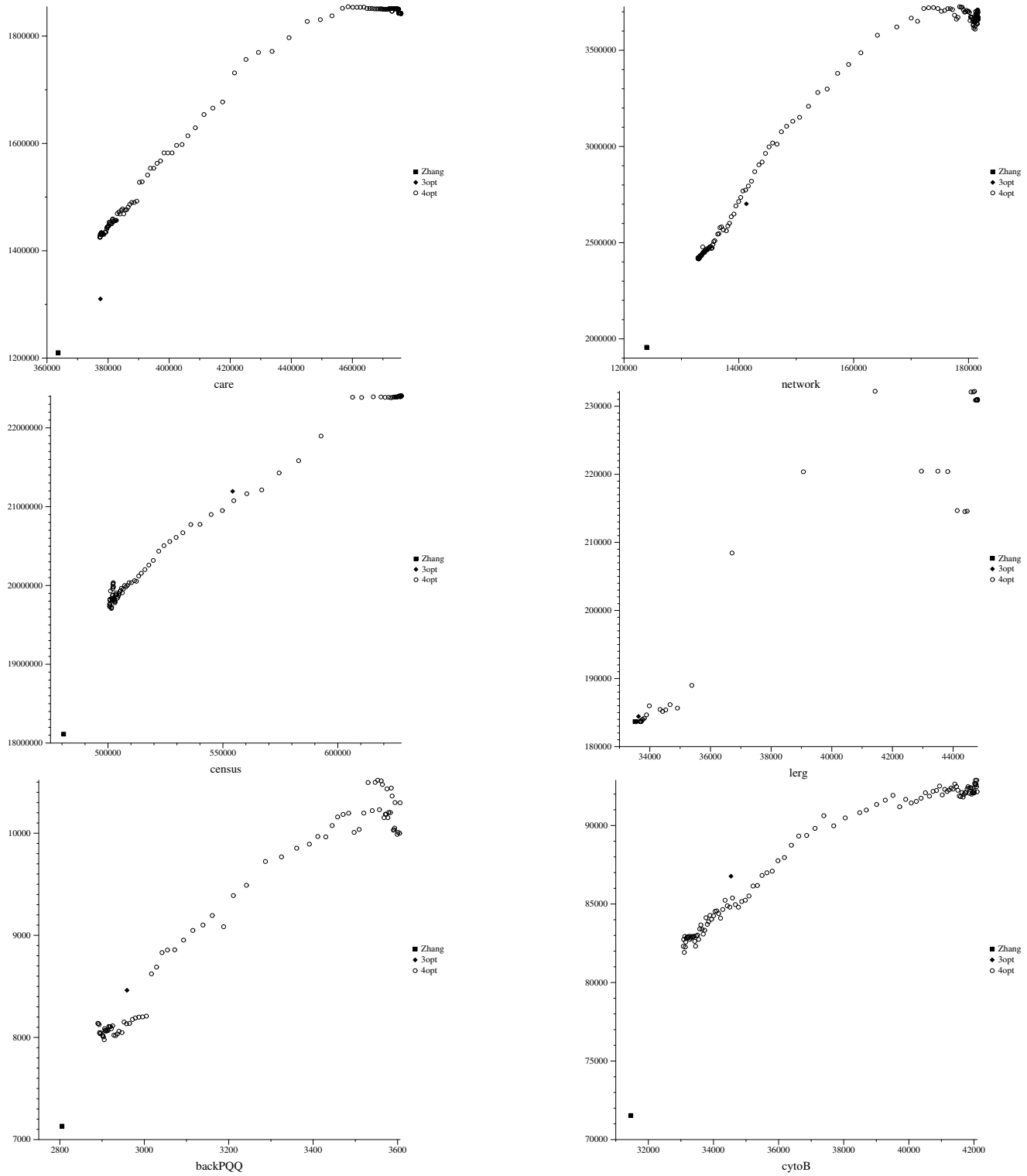

Figure 1: Relationship between tour cost (x-axes) and compression size (y-axes) for CARE, NETWORK, CENSUS, LERG, BACKPPQ, and CYTOB, using the result of Zhang's algorithm, a 3-opt local optimum, and a sequence of tours generated by a series of 4-opt changes.

since the output size is linear in this number.

Let $X = \{x_1, \ldots, x_n\}$ be a set of strings. A *batch of* X is an ordered subset of X. A *schedule of* X is a partition of X into batches. A batch $B = (x_{i_1}, \ldots, x_{i_s})$ is *processed* by $\mathcal{C}$ by computing $\mathcal{C}(B) = \mathcal{C}(x_{i_1} \cdots x_{i_s})$; i.e., by compressing the superstring formed by catenating the strings in B in order. A schedule $\mathcal{S}$ of X is *processed* by $\mathcal{C}$ by processing its batches, one by one, in any order. While $\mathcal{C}(\mathcal{S})$ is ambiguous, $|\mathcal{C}(\mathcal{S})|$ is well defined: $|\mathcal{C}(\mathcal{S})| = \sum_{B \in \mathcal{S}} |\mathcal{C}(B)|$. Our main problem can be stated as follows.

PROBLEM 5.1. *Let X be a set of strings. Find a schedule $\mathcal{S}$ of X minimizing $|\mathcal{C}(\mathcal{S})|$ among all schedules.*

The shortest common superstring (SCS) problem is an example. For two strings x and y, let *pref*(x, y) be the prefix of x that ends at the longest suffix-prefix match of x and y. Let X be a set of n strings, and let π be a permutation of the integers in $[1, n]$. Define $S(X, \pi) =$ *pref*(x_{π_1}, x_{π_2})*pref*$(x_{\pi_2}, x_{\pi_3}) \cdots$ *pref*$(x_{\pi_{n-1}}, x_{\pi_n})x_{\pi_n}$. $S(X, \pi)$ is a superstring of X; π corresponds to a schedule of X; and the SCS of X is $S(X, \pi)$ for some π [10]. Therefore, finding the SCS is an instance of Problem 5.1, where $\mathcal{C}(\cdot)$ is $S(\cdot)$. Since finding the SCS is MAX-SNP hard [2], Problem 5.1 is MAX-SNP hard in general. Different results can hold for specific compressors, however.

Now consider a table T as a set of n columns. A batch is a subset of columns. A variation of table compression in which the batches are compressed in column-major (instead of row-major) order is an instance of Problem 5.1. In row-major order, it is not the same, because the input strings are intermixed. This distinction is crucial. When $\mathcal{C}$ is run length encoding, column-major table compression can be solved in polynomial time, while row-major table compression is MAX-SNP hard. The connection between table compression and SCS through Problem 5.1 makes these problems theoretically elegant as well as practically motivated.

5.1 Complexity with LZ77. Recall the LZ77 parsing rule [18], which is used by compressors like gzip. Consider a string z, and, if $|z| \geq 1$, let z^- denote the prefix of z of length $|z| - 1$. If $|z| \geq 2$, then define $z^{--} = (z^-)^-$. LZ77 parses z into *phrases*, each a substring of z. Assume that LZ77 has already parsed the prefix $z_1 \cdots z_{i-1}$ of z into phrases $z_1, \ldots, z_{i-1}$, and let z' be the remaining suffix of z. LZ77 selects the i'th phrase z_i as the longest prefix of z' that can be obtained by adding a single character to a substring of $(z_1 \cdots z_{i-1} z_i)^-$. Therefore, z_i has the property that z_i^- is a substring of $(z_1 z_2 \cdots z_{i-1} z_i)^{--}$, but z_i is not a substring of $(z_1 z_2 \cdots z_{i-1} z_i)^-$. This recursive definition is sound [13].

We outline a reduction that shows that Problem 5.1 is MAX-SNP hard when $\mathcal{C}$ is LZ77. We leave details of the proof for the full paper. Consider TSP(1,2), the traveling salesman problem where each distance is either 1 or 2. An instance of TSP(1,2) can be specified by a graph H, where the edges of H connect those pairs of vertices with distance 1. The problem remains MAX-SNP hard if we bound the outdegree of each vertex in H by some arbitrary but fixed constant [16]. This result holds for both symmetric and asymmetric TSP(1,2); i.e., for both undirected and directed graphs H. We assume that H is directed. We further assume without loss of generality that no vertex in H has outdegree 1, for the outgoing edge from any such vertex must be in any TSP(1,2) solution.

We associate a set $S(H)$ of strings to the vertices and edges of H; $S(H)$ will be the input to Problem 5.1. Each vertex v engenders three symbols: v, v', and $\$_v$. Let $w_0, \ldots, w_{d-1}$ be the vertices on the edges out of v in H, in some arbitrary but fixed cyclic order. For $0 \leq i < d$ and mod-d arithmetic, we say that edge (v, w_i) *cyclicly precedes* edge (v, w_{i+1}). The $d+1$ strings we associate to v and these edges are: $e(v, w_i) = (v' w_{i-1})^4 v' w_i$, for $0 \leq i < d$ and mod-d arithmetic; and $s(v) = v^4 (v')^5 \$_v$.

LEMMA 5.1. *Let H have n_h vertices and m_h edges. A TSP(1,2) solution with k cost-2 edges (thus of cost $n_h - 1 + k$) can be transformed into a table compression schedule of cost $m_h + k + 3n_h + 1$, and vice-versa.*

PROOF (SKETCH). The core idea of the transformation is a canonical form for expressing paths in H. It can be shown that, for all edges (v, w), $e(v, w)$ parses into one phrase when immediately preceded by $e(v, y)$ for the cyclic predecessor (v, y) of (v, w), and into more phrases otherwise; and that $s(v)$ parses into two phrases when immediately preceded by some $e(x, v)$, and into three phrases otherwise. Thus, an edge (v, w_i) is best encoded $s(v)e(v, w_{i+1})e(v, w_{i+2}) \cdots e(v, w_i)s(w_i)$. A path of r vertices in this form parses into $3r + 1 + D$ phrases, where D is the sum of the outdegrees of the vertices. □

THEOREM 5.1. *Problem 5.1 is MAX-SNP hard when $\mathcal{C}$ is LZ77.*

PROOF (SKETCH). Any schedule can be transformed in polynomial time into canonical form at no extra cost. Given a TSP(1,2) solution, we derive the corresponding canonical-form schedule for T; and given a schedule for T, we transform it into canonical form, from which we derive a TSP(1,2) solution. Lemma 5.1 shows linearity of costs. □

5.2 Complexity with Run Length Encoding. In run length encoding (RLE), an input string is parsed into phrases of the form (σ, n), where σ is a character, and n is the number of times σ appears consecutively. For example, $aaaabbbbaaaa$ is parsed into $(a, 4)(b, 4)(a, 4)$.

THEOREM 5.2. *Problem 5.1 can be solved in polynomial time when $\mathcal{C}$ is run length encoding.*

PROOF (SKETCH). Let $x_1, \ldots, x_n$ be the input strings. An SCS is $\mathit{pref}(x_{\pi_1}, x_{\pi_2}) \cdots \mathit{pref}(x_{\pi_{n-1}}, x_{\pi_n}) x_{\pi_n}$ for some permutation π. Assume without loss of generality that each x_i is of the form $\sigma\sigma'$; i.e., two distinct characters. Thus, $\mathit{pref}(x_i, x_j)$ is of length 2 if the last character of x_i equals the first of x_j and 3 otherwise. An SCS therefore gives an optimal RLE parsing, and SCS can be solved in polynomial time for input strings of length two [8]. □

We prove MAX-SNP hardness of row-major table compression with RLE using a transformation analogous to that in Section 5.1. Let H be a graph encoding a TSP(1,2) instance. We transform the vertices and edges of H into an instance of row-major table compression. We associate a column to each vertex and edge of H.

For each vertex v, we generate three symbols: v, v', and v''. Let $w_0, \ldots, w_{d-1}$ be the vertices on the edges out of v in some fixed, arbitrary cyclic order. We associate the following strings to v and its outgoing edges: $s(v) = v'v''v$; and $e(v, w_i) = v'v''w_i$, $0 \leq i < d$. The input table is formed by assigning each such string to a column.

THEOREM 5.3. *Row-major table compression is MAX-SNP hard when $\mathcal{C}$ is run length encoding.*

PROOF (SKETCH). Given a solution to the TSP(1,2) instance with k cost-2 edges, we can transform it into a schedule (in row-major order) of cost $n_h + 2m_h + k + 1$, and vice versa. The canonical form for an edge (v, w) is the column for $s(v)$, followed by the columns for all the $e(v, q)$ in any order except that $e(v, w)$ is last, followed by $s(w)$. □

6 Conclusion

We demonstrate a general framework that links independence among groups of variables to efficient partitioning algorithms. We provide general solutions in ideal cases in which dependencies form equivalence classes or cost functions are sub-additive. The application to table compression suggests that there exist weaker structures that allow partitioning to produce significant cost improvements. Open is the problem of refining the theory to explain these structures.

Based on experimental results, we conjecture that our TSP reordering algorithm is close to optimal; i.e., that no partition-based algorithm will produce significantly better compression rates. It is open if there exists a measurable lower bound for compression optimality, analogous, e.g., to the Held-Karp TSP lower bound.

Finally, while we have shown some MAX-SNP hardness results pertaining to table compression, it is open whether the problem is even approximable to within constant factors.

Acknowledgements

We are indebted to David Johnson for running his implementation of Zhang's algorithm and local 3-opt on our files. We thank David Applegate, Flip Korn, Cecilia LaNave, S. Muthukrishnan, Grazieno Pesole, and Andrea Sgarro for many useful discussions.

References

[1] A. Bateman, E. Birney, R. Durbin, S. R. Eddy, K. L. Howe, and E. L. L. Sonnhammer. The Pfam protein families database. *Nucleic Acids Res.*, 28(1):263–6, 2000.

[2] A. Blum, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *J. ACM*, 41(4):630–47, 1994.

[3] A. L. Buchsbaum, D. F. Caldwell, K. W. Church, G. S. Fowler, and S. Muthukrishnan. Engineering the compression of massive tables: An experimental approach. In *Proc. 11th ACM-SIAM SODA*, pages 175–84, 2000.

[4] Census of population and housing, 1990: Summary tape file 3 on CD-ROM. U.S. Bureau of the Census, Washington, 1992.

[5] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Proc. 3rd ALENEX*, volume 2153 of *LNCS*, pages 32–59. Springer-Verlag, 2001.

[6] G. Cormack. Data compression in a data base system. *C. ACM*, 28(12):1336, 1985.

[7] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[9] S. Grumbach and F. Tahi. A new challenge for compression algorithms: Genetic sequences. *Inf. Proc. & Manag.*, 30(6):875–86, 1994.

[10] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[11] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees. *OR*, 18:1138–62, 1970.

[12] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Math. Prog.*, 1:6–25, 1971.

[13] S. R. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM J. Comp.*, 29(3):893–911, 2000.

[14] C. Lanave, S. Liuni, F. Licciulli, and M. Attimonelli. Update of AMmtDB: A database of multi-aligned Metazoa mitochondrial DNA sequences. *Nucleic Acids Res.*, 28(1):153–4, 2000.

[15] C. Nevill-Manning and I. H. Witten. Protein is incompressible. In *Proc. IEEE DCC '99*, pages 257–66, 1999.

[16] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Op. Res.*, 18(1):1–11, 1993.

[17] W. Zhang. Truncated branch-and-bound: A case study on the asymmetric TSP. In *Spring Symposium on AI and NP-Hard Problems*, pages 160–6. AAAI, 1993.

[18] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Thy.*, IT-23(3):337–43, 1977.