On Solving the Sparse Matrix Compression Problem Greedily

Dominik Köppl [*] Vincent Limouzy [†] Andrea Marino [‡] Giulia Punzi [§]
Takeaki Uno [¶]

Abstract

The sparse matrix compression problem asks for a one-dimensional representation of a binary $n \times \ell$ matrix, formed by an integer array of row indices and a shift function for each row, such that access to the matrix can be done in constant time by consulting the representation. It has been shown that the decision problem for finding an integer array of length $\ell + k$ or restricting the shift function up to values of k is NP-complete. In that light, a greedy algorithm has been proposed to shift the i-th row until it forms a solution with its predecessor rows. Despite this greedy algorithm being cherished for its good approximation in practice, we show that it actually exhibits an approximation ratio of $\Theta(\sqrt{\ell + k})$.

1 Introduction

Binary matrices have ubiquitous applications in computer science, such as in databases, data mining, and machine learning. For instance, a binary matrix can represent an incidence matrix of a bipartite graph or a transition matrix of a finite automaton. In practice, these matrices are often sparse, which means that most of the entries are zero. To save space and time, it is desirable to compress these matrices. One way to compress a sparse matrix is to represent it as a one-dimensional array of row indices and a shift function for each row. Such a representation allows for constant-time access to the matrix, and has already been proposed in the 1970s. It was also studied in the context of compilers and databases in the 1980s [1]. The problem of finding such a representation is known as the *sparse matrix compression problem*, which we denote by SMC. The original version of the decision problem of SMC is defined as follows:

Problem 1.1 (SMC, [10, Chapter A4.2, Problem SR13]). Given an $n \times \ell$ matrix $A[1..n][1..\ell]$ with n rows and ℓ columns and entries $A[i][j] \in \{0, 1\}$ for all $i \in [1..n]$, $j \in [1..\ell]$, and an integer $k \in [0..\ell \cdot (n-1)]$, determine whether there exists an integer array $C[1..\ell + k]$ with $C[i] \in [0..n]$ for every $i \in [1..\ell + k]$, and a function $s : [1..n] \to [0..k]$ such that $A[i][j] = 1 \Leftrightarrow C[s(i) + j] = i$ for all $i \in [1..n]$ and $j \in [1..\ell]$. Here, we assume $A[0][j] = 0$ for all j to allow setting $C[i] = 0$ for some i, modeling that this entry is unassigned.

In what follows, we call C the *placement* and s the *shift function* of the representation of A asked by SMC.

Example 1.2. Consider the SMC problem with $k = 2$ for the following matrix A (of size 3×5) defined

[*]University of Yamanashi
[†]University Clermont Auvergne
[‡]University of Florence
[§]University of Pisa
[¶]National Institute of Informatics

as follows (first zero row omitted):

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Intuitively, we wish to find a way to shift each row in a way such that no column contains two '1's, and use this information to compress the matrix into a one-dimensional vector of row indices of length $5 + 2 = 7$.

For instance, the shift function $s(1) = 0, s(2) = 2, s(3) = 1$ allows no column collisions for the '1's. Such shift can be represented as follows:

$$\begin{array}{cccccccc} 0 & 1 & 0 & 0 & 1 & 0 & & \\ \rightarrow & \rightarrow & 0 & 1 & 0 & 1 & 0 & 0 \\ \rightarrow & 0 & 1 & 0 & 0 & 0 & 1 & \end{array}$$

From this, we can obtain the placement $C = [0, 1, 3, 2, 1, 2, 3]$ by setting $C[j]$ to be equal to the only row index which has a 1 in column j after the shift function is applied. Since such C is of the desired length, the problem has a positive answer.

Solving this problem is not an easy task. For instance, a brute-force algorithm checks for all possible shifts whether we can linearize the shifted matrix A with C. However, the number of possible configurations for the shift function s is $(k + 1)^n$, which is prohibitive even for a maximum shift k of 1.

The optimization version of SMC is to find the smallest k such that there exists a solution. We can also understand this problem as finding the smallest upper bound on the maximum shift.

However, the decision problem is already NP-complete, as shown by Even et al. [9]. They showed that the problem is NP-complete by reducing the 3-COLORING problem to SMC. The 3-COLORING problem is to decide whether a given graph can be colored with three colors so that no two adjacent vertices have the same color.

However, here we study a modification of SMC, which we justify by the fact that the original problem does not take into account that the leftmost columns of all rows could be empty (as in the matrix of Example 1.2). So, we want to consider the length of the placement $(b_1, \ldots, b_{\ell+k})$ after trimming its empty borders. It is therefore no longer the case that minimizing the length of the placement and minimizing the maximum length of a shift are equivalent. For instance, a matrix consisting of only the empty row would give a sequence of length ℓ. We name our variant 1TETRIS, based on the fact that we can model the problem as a combinatorial puzzle of 1-dimensional polyominos with gaps, which we call *tiles* in the following. The problem is formally defined as follows.

Problem 1.3 (1TETRIS)**.** Given a set of n binary strings $S_1, \ldots, S_n$ of length ℓ such that $S_i \in \{\circ, i\}^\ell$ and an integer $k \in [0..\ell n]$, determine whether there exists a placement $S[1..k] \in (\{\circ\} \cap [1..n])^k$ such that S_i has a match in S, where $\circ$ always matches (making it possible to match longer strings with shorter ones).

An instance of 1TETRIS with $S_i[1] = S_i[\ell] = i$ is equivalent to SMC if we increment k by ℓ. Vice versa, an instance of SMC is equivalent to 1TETRIS when we

1. trim all columns by their prefixes and suffixes of '0's,
2. decrement k by the length of the longest column after trimming,
3. map all remaining zeros to $\circ$, and
4. map each '1' to its corresponding row index.

Example 1.4. For matrix A of Example 1.2, we have a corresponding instance of 1TETRIS given by $S_1 = 1\circ\circ1, S_2 = 2\circ2$, and $S_3 = 3\circ\circ\circ3$. For $k = 6$, a possible placement is $S = 132123$.

Lemma 1.5. A placement of 1TETRIS without holes is optimal, but not every problem instance admits a placement without holes.

2 Related Work

Ziegler [13] was the first who mentioned Problem 1.1. He gave a heuristic that tries to fit the next row at the leftmost possible available position. He also augmented his heuristic with a strategy that rearranges the order of the rows by prioritizing the row with the largest number of '1's. This strategy is known as Ziegler's algorithm. Despite the fact that Ziegler's algorithm is only a heuristic, it is often used in practice due to its good performance. For instance, Sadayappan and Visvanathan [12] similarly studied practical aspects of Ziegler's algorithm, and Aho et al. [1, Section 3.9.8] recommend using Ziegler's algorithm to represent the state transition of a deterministic finite automaton (DFA) in a compressed form. However, the approximation ratio of Ziegler's algorithm has not been studied yet.

Unfortunately, finding the optimal solution is generally hard. Even et al. [9] gave an NP-hard proof based on 3-coloring, which found an entry in the textbook of Garey and Johnson [10, Chapter A4.2, Problem SR13]. They showed that the problem is NP-complete even if the maximum needed shift is at most two. Problem 1.1 has been adapted to Bloom filters [4, 5, 8], and has also been studied under the name COMPRESSED TRANSITION MATRIX [6, Sect. 4.4.1.3] problem. Finally, Bannai et al. [2] showed that Problem 1.1 is NP-hard even when the width ℓ of the matrix is $\ell \in \Omega(\lg n)$. Their hardness proof can be applied directly to 1TETRIS, proving that 1TETRIS is NP-hard even when all tiles have a width of $\ell \in \Omega(\lg n)$.

Chang and Buehrer [3] considered a different variant in which cyclic rotations of a matrix row are allowed. However, this work is only practical and does not provide any theoretical guarantees. Another variation is to restrict rows to have no holes and to have not only one placement, but a fixed number of placements of the same length, which reduces to a rectangle packing problem [7, Section 2]. Finally, Manea et al. [11] studied a variant of embedding subsequences with gap constraints.

3 Preliminaries

The *leftmost-fit greedy strategy* processes the tiles in the order in which they are given, for instance by an queue.

1. Place the first tile at position 1.
2. Place the next tile at the leftmost position ≥ 1 at which the tile fits.
3. Repeat Step 2 until the last tile of the input queue has been placed.

Ziegler's algorithm runs the leftmost-fit greedy strategy after sorting the input sequence by the number of '1's, starting with the tile that has the larger number of '1's. In this handout, we show the following result on the approximation ratio of the leftmost-fit greedy strategy and Ziegler's algorithm.

Theorem 3.1. Both greedy strategies have an approximation ratio of $\mathcal{O}(\sqrt{m})$ for 1TETRIS if the optimal solution has length m. This ratio is tight in the sense that there is an instance for which both strategies exhibit a ratio of $\Omega(\sqrt{m})$.

4 Approximation Ratio of Greedy Strategies

We proof Thm. 3.1 in two parts for the lower and the upper bounds.

Table 1: X and Y for $k = 4$ with collisions at $i_1 = 1$ and $i_2 = 13$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
X	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
Y	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

4.1 Lower Bound

We consider only two different types of tiles X and Y with frequencies $k-2$ and $k-1$, respectively. We define both tiles X and Y to be periodic with roots 10^{k-2} and 10^{k-1}, respectively. Since $k-1$ and k are co-prime, $X[i] = Y[i] = 1$ if and only if $i = 1 + jk(k-1)$ for some integer $j \geq 0$ and $j \leq \min(|X|, |Y|)$. Let us suppose that X and Y have lengths of at least $1+k(k-1)$. We can then say more generally that, for every $d \in [0..k]$, there exists an $i \in [1..k(k-1)]$ such that $X[i] = Y[i+d] = 1$. Thus, given that we placed Y as the first tile at position 1, the first position we can place X is in the range $[|Y| - 2k..|Y|]$. Similarly, if we placed X as the first tile at position 1, the first position we can place Y is in the range $[|X| - 2k..|X|]$. Suppose that we have placed Y first and second X at the first available position. Further, suppose that we want to place Y. Because Y is in conflict with itself, we need to shift it. However, we need to shift to the last k position of the placed X, otherwise some positions of Y are in conflict with the already placed X. If we continue placing X and Y in alternating order, we end up with a placement of length $\Omega((k-2)(|X|-2k)+(k-1)(|Y|-2k))$. Setting the lengths of X and Y to $k(k+1)+1$, the placement length is $\Omega(k^3)$.

However, an optimal placement has length $\Theta(k^2)$. To see this, first place all $k-2$ many X tiles at positions 1, 2, …, $1+k-2$. This gives a placement of length $1 + k - 2 + |X| - 1 = 1 + k - 2 + k(k+1) + 1 - 1 = k^2 + 2k - 1$. Next, place all Y tiles at subsequent positions $k^2 + 2k$, k^2+2k+1, …, k^2+3k-2. The total length of this placement thus is $k^2 + 3k - 2 + |Y| = \Theta(k^2)$. Since this placement is without holes, it is optimal by Lemma 1.5. In total, the placement of the greedy algorithm is at least $\Omega(k)$ times larger than the optimal solution of length $\Theta(k^2)$. Since k is arbitrary, the greedy algorithm selecting alternatively X and Y tiles has an approximation ratio of at least $\sqrt{m}$, where $m = \Theta(k^2)$ being the length of the optimal solution.

Finally, we can scale the lengths of each X and Y tile individually to let Ziegler's method behave like the leftmost-fit greedy strategy. Since Ziegler's method greedily selects the remaining tile with the highest number of '1's, we make every X and Y tile unique such that the method will select X and Y in an alternating order. To this end, define $X_i = (10^{k-2})^{2i+1}1$ and $Y_i = (10^{k-1})^{2i}1$. This lets Ziegler's precomputation step arrange the tiles in a list $[Y_{k-1}, X_{k-2}, Y_{k-2}, X_{k-3}, \ldots, X_1, Y_1]$. Like before, an optimal solution is to combine all X_i and Y_i, each separately into two placements without holes, each of length $\Theta(k^2)$. Analogously, Ziegler's method works like the alternating algorithm resulting in a string of length $\Omega(k^3)$.

Corollary 4.1. Both greedy strategies have an approximation ratio of $\Omega(\sqrt{m})$ for SMC if the optimal solution has length m.

Proof. If we write each tile as a row in a matrix, then both greedy strategies have a maximal shift of $\Omega(k^3)$. Our solution without holes needs a maximal shift of $\mathcal{O}(k^2)$ to move the last Y. Therefore, the approximation ratio is at least $\Omega(\sqrt{m})$. □

4.2 Upper Bound

Given that the length of the optimal solution is m, assume for a proof by contradiction that one of the greedy strategies generates a placement T longer than $(3m+1)m$. The coarse idea is to split T into two strings X and Y such that we can show that X or Y have at least m many '1"s, which contradicts the optimal length m by the pigeonhole principle. In detail, we split T into a prefix X of $2m\sqrt{m}$ positions and a suffix Y of $(m+1)\sqrt{m}$ positions. We further decompose X into chunks of length $2m$. Let C denote the chunk in X with the minimum number of '1"s, which we define to be x.

We consider tiles that the greedy strategy could not place in X and thus got placed into Y. Let P be one of these tiles with the least number of '1's, which we define as y. Since the optimal solution has length m, all tiles have lengths of at most m. In particular, P is completely contained in Y, which is composed of at least $\sqrt{m}+1$ tiles.

Since P could not be placed in X, P conflicts with C at each position of C. Each conflict can be expressed by the ranks of '1' of the x '1's in C and the y '1's in P. Using a counting argument for the ranks $\in [1..x] \times [1..y]$, the product xy must be at least as large as $2m$, the length of C. In particular, x or y must be at least $\sqrt{m}$, for which we have a short case analysis.

- If x is at least $\sqrt{m}$, then X must have at least m '1's. That is because it has $\sqrt{m}$ chunks and each chunk has at least as many '1's as C by choice of C.
- If y is at least $\sqrt{m}$, then Y must have at least m '1's. That is because at least $\sqrt{m}$ tiles have been placed into Y, and each has at least as many '1's as P by choice of P.

In both cases, the number of '1's in T must exceed m. We thus obtain a contradiction that T and the optimal solution must contain the same number of '1's, so the optimal solution must be longer than m.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986.

[2] Hideo Bannai, Keisuke Goto, Shunsuke Kanda, and Dominik Köppl. NP-completeness for the space-optimality of double-array tries. *arXiv CoRR*, abs/2403.04951, 2024. doi: 10.48550/ARXIV.2403.04951.

[3] Chin-Chen Chang and Daniel J. Buehrer. An improvement to Ziegler's sparse matrix compression algorithm. *J. Syst. Softw.*, 35(1):67–71, 1996. doi: 10.1016/0164-1212(95)00086-0.

[4] Chin-Chen Chang and Tzong-Chen Wu. A letter-oriented perfect hashing scheme based upon sparse table compression. *Softw. Pract. Exp.*, 21(1):35–49, 1991. doi: 10.1002/SPE.4380210104.

[5] Chin-Chen Chang, Huey-Cheue Kowng, and Tzong-Chen Wu. A refinement of a compression-oriented addressing scheme. *BIT Numerical Mathematics*, 33(4):529–535, 1993. doi: 10.1007/BF01990533.

[6] Jan Daciuk, Jakub Piskorski, and Strahil Ristov. Natural language dictionaries implemented as finite automata. In Carlos Martín-Vide, editor, *Scientific Applications Of Language Methods*, volume 2, chapter 4. World Scientific, 2010.

[7] Erik D. Demaine and Martin L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs Comb.*, 23(Supplement-1):195–208, 2007. doi: 10.1007/S00373-007-0713-4.

[8] Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. Fast succinct retrieval and approximate membership using ribbon. In *Proc. SEA*, volume 233 of *LIPIcs*, pages 4:1–4:20, 2022. doi: 10.4230/LIPICS.SEA.2022.4.

[9] S. Even, D.I. Lichtenstein, and Y. Shiloah. Remarks on Ziegler's method for matrix compression. unpublished, 1977.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. A Series of books in the mathematical sciences. Bell Laboratories, 1979.

[11] Florin Manea, Jonas Richardsen, and Markus L. Schmid. Subsequences with generalised gap constraints: Upper and lower complexity bounds. In *Proc. CPM*, volume 296 of *LIPIcs*, pages 22:1–22:17, 2024. doi: 10.4230/LIPICS.CPM.2024.22.

[12] P. Sadayappan and V. Visvanathan. Efficient sparse matrix factorization for circuit simulation on vector supercomputers. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 8 (12):1276–1285, 1989. doi: 10.1109/43.44508.

[13] S. F. Ziegler. Small faster table driven parser. Technical report, Madison Academic Computing Center, University of Wisconsin, 1977. unpublished.