



On Computing Optimal Linear Diagrams

Alexander Dobler  and Martin Nöllenburg 

Algorithms and Complexity Group, TU Wien, Vienna, Austria
{adobler,noellenburg}@ac.tuwien.ac.at

Abstract. Linear diagrams are an effective way to visualize set-based data by representing elements as columns and sets as rows with one or more horizontal line segments, whose vertical overlaps with other rows indicate set intersections and their contained elements. The efficacy of linear diagrams heavily depends on having few line segments. The underlying minimization problem has already been explored heuristically, but its computational complexity has yet to be classified. In this paper, we show that minimizing line segments in linear diagrams is equivalent to a well-studied NP-hard problem, and extend the NP-hardness to a restricted setting. We develop new algorithms for computing linear diagrams with minimum number of line segments that build on a traveling salesperson (TSP) formulation and allow constraints on the element orders, namely, forcing two sets to be drawn as single line segments, giving weights to sets, and allowing hierarchical constraints via PQ-trees. We conduct an experimental evaluation and compare previous algorithms for minimizing line segments with our TSP formulation, showing that a state-of-the-art TSP-solver can solve all considered instances optimally, most of them within few milliseconds.

Keywords: Linear Diagrams · Consecutive Ones · TSP · NP-hardness · Algorithm Benchmarking

1 Introduction

Many real-world datasets represent set systems, and there is a vast landscape of different visualization techniques for set-based data. Two well-known techniques are Euler and Venn Diagrams that draw sets as closed curves and set intersections are represented by intersections of the boundaries of these curves. For a detailed survey of these and other set visualizations we refer to Alsallakh et al. [1].

The set visualization that we study in this paper are linear diagrams. It has been demonstrated that they are simple and effective, and have advantages when compared with other set visualizations [8,23,31]. Linear diagrams represent elements as columns and sets as rows of a matrix or table, where in each row there are one or more horizontal line segments indicating which elements are contained in a specific set. Vertical overlaps of these line segments in different rows show set intersections, and the corresponding elements. Figure 1a shows a linear diagram representing a Simpsons data set introduced by Jacobsen et al. [19]. For example, the set Blue Hair contains the elements Jacquelin Bouvier, Marge, and Milhouse,

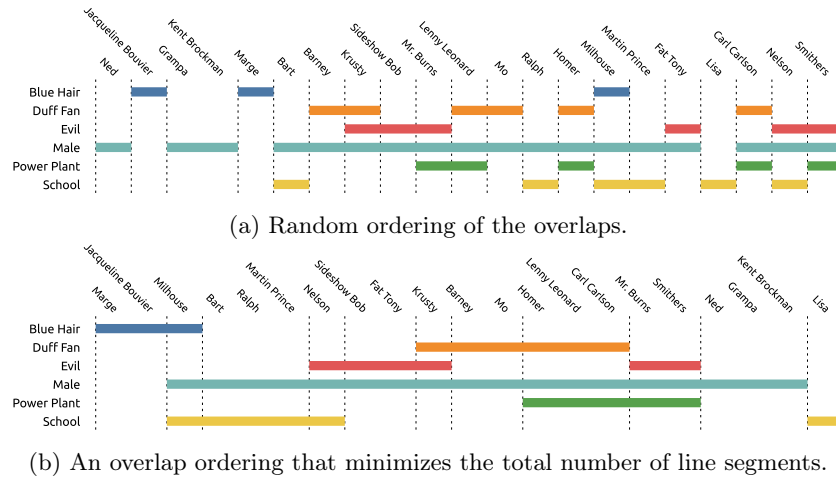


Fig. 1: Linear diagrams representing the Simpsons.

and is drawn with three line segments. Mr. Burns is contained in the sets Evil, Male, and Power Plant, as represented by the corresponding vertical overlap of the line segments in these three rows with the column of Mr. Burns.

Linear diagrams can be drawn in many ways, e.g., by choosing different permutations of the rows/sets and columns/overlaps. It has been shown that there are several quality criteria for linear diagrams, while the most important one is finding an ordering of the elements that minimizes the number of line segments [28]. For example, the linear diagram depicted in Figure 1b shows the same set system as before, but with an ordering of the overlaps that minimizes the number of line segments, here using 8 segments instead of 23.

The underlying computational problem of finding an ordering of the overlaps that minimizes line segments seems hard, as for n overlaps, there are $n!$ different orderings of these overlaps. Finding orderings that minimize line segments is mainly done via heuristics in the literature [7, 18, 28]. The main topic of this paper is computing *optimal linear diagrams* – those which realize the minimum possible number of line segments that have to be drawn.

Related work. Several user studies were performed to compare the efficacy of linear diagrams and other diagram types; they showed that linear diagrams perform equally well or better than other diagram types including Euler and Venn diagrams [8, 23, 29]. Linear diagrams have then been used, e.g., to visualize sets over time [26], and Lamy et al. [21] extended linear diagrams to allow multiple sets per row.

Existing algorithms for minimizing line segments in linear diagrams are of heuristic nature, i.e., they may often find good solutions, but do not provide proven guarantees on the solution quality. Rodgers et al. [28] presented a simple heuristic that first defines a pair-wise similarity between two overlaps based

on the number of sets they have in common. Then, this heuristic iteratively builds an overlap ordering aiming to group similar overlaps next to each other. Chapman et al. [7] compared different heuristics based on simulated annealing, a travelling salesperson (TSP) formulation, and other variants of the heuristic of Rodgers et al. [28]. A GitHub project [18] provides an implementation of linear diagrams in Python. The underlying algorithm tries to minimize the number of line segments by applying multiple runs of an iterative greedy heuristic, each with a different pair-wise similarity measure between overlaps that is augmented by random seeds.

Contribution and structure. We further investigate the computational problem of computing optimal linear diagrams. Section 2 defines general preliminaries and notation for permutations, matrices, and graphs. In Section 3, we describe how the problem of computing optimal linear diagrams can be modelled as a known problem on binary matrices, thus bridging the gap missing in the literature¹. This problem is known to be NP-complete; we further strengthen this NP-completeness result by showing that computing optimal linear diagrams is even NP-complete for set systems where each set contains exactly two elements and each element is contained in exactly three sets. Moreover, we present further literature on matrix problems that are relevant with regard to linear diagrams.

In Section 4, we present a way to compute optimal linear diagrams by reducing the problem to TSP, thus, completing the work of Chapman et al. [7]. They also presented an algorithm based on a TSP formulation, but this algorithm sometimes produces non-optimal overlap orderings. We further expand on this formulation, showing that we can model specific constraints on the overlap orders. Namely, we can force up to two sets to be drawn as single line segments while still minimizing the number of line segments. This is particularly interesting for allowing interactivity in linear diagrams [5]. We also show how to model constraints based on weighted sets and hierarchical ordering constraints represented by PQ-trees, which is of interest for certain set visualization tasks.

In Section 5, we conduct an experimental evaluation of our algorithms from Section 4, and compare them with the state-of-the-art heuristics. We show that a state-of-the-art TSP-solver can solve all considered instances optimally, most of them within few milliseconds. We also verify that the considered heuristics from the literature perform well with regard to the number of line segments, where the average optimality gaps of the heuristics are less than ten percent.

2 Preliminaries

Let A be a matrix with m rows and n columns; we set $n_A = n$ and $m_A = m$. We write $A_{i,j}$ with $1 \leq i \leq m$ and $1 \leq j \leq n$ for the *entry* of A at row i and column j . Furthermore, by r_i^A and c_j^A we denote the i -th row and j -th column of A , respectively. A matrix is a *binary matrix* if all its entries are either 0 or 1.

¹ This observation has been made independently in a paper that was published after the first version of our paper [6].

If it is clear from the context, we might omit explicitly mentioning the matrix A in the above notations.

We denote by $[k]$ the set of elements $\{1, \dots, k\}$. A *permutation* $\pi : [k] \rightarrow X$ is a bijective function from $[k]$ to a set X . Sometimes we write permutations π as sequences of elements, that is, $\pi = (x_1, \dots, x_n)$ is the permutation such that $\pi(i) = x_i$ for $1 \leq i \leq n$. We denote by Π_k the set of all permutations from $[k]$ to $[k]$. For two permutations $\pi_1 = (x_1, \dots, x_n)$ and $\pi_2 = (y_1, \dots, y_m)$, we denote by $\pi_1 \star \pi_2$ their *concatenation* $(x_1, \dots, x_n, y_1, \dots, y_m)$. For two sets Π_1 and Π_2 of permutations, we define $\Pi_1 \star \Pi_2 = \{\pi_1 \star \pi_2 \mid \pi_1 \in \Pi_1, \pi_2 \in \Pi_2\}$.

For a matrix A and a permutation $\pi : [n_A] \rightarrow [n_A]$ we denote by $\pi(A)$ the matrix such that $\pi(A)_{i,j} = A_{i,\pi(j)}$. Equivalently $\pi(r_i^A) = r_{\pi(i)}^A$ for a row r_i^A . By “a permutation of the columns of the matrix A ” we mean a permutation $\pi : [n_A] \rightarrow [n_A]$.

A block of *consecutive ones* in a row r_i^A of a matrix A with n columns is a maximal non-empty sequence $A_{i,p}, A_{i,p+1}, \dots, A_{i,q}$ satisfying

- $A_{i,j} = 1$ for all $p \leq j \leq q$,
- $p = 1$ or $A_{i,p-1} = 0$, and
- $q = n$ or $A_{i,q+1} = 0$.

For a row r_i^A , $\text{cons1}(r_i^A)$ is the number of blocks of consecutive ones in r_i^A . Additionally, $\text{splits}(r_i^A)$ (the number of gaps between the blocks) is defined as $\text{cons1}(r_i^A) - 1$ if r_i^A contains a 1-entry, and 0 otherwise. We define $\text{cons1}(A) = \sum_{i=1}^{m_A} \text{cons1}(r_i^A)$ for a matrix A . Equivalently, $\text{splits}(A) = \sum_{i=1}^{m_A} \text{splits}(r_i^A)$. Let c_i^A and c_j^A be two columns of a binary matrix. By $d_h(c_i^A, c_j^A)$ we denote the Hamming distance between c_i^A and c_j^A , that is, the number of rows with different values.

In this paper we assume graphs G as simple and undirected. By $V(G)$ and $E(G)$ we denote the vertex set and edge set of G , respectively. For a binary matrix A , let $G(A)$ be the complete graph that consists of the vertices $V = \{v_i \mid c_i^A \text{ is a column in } A\}$. If we talk about a vertex v_i with index i in $G(A)$, we mean the vertex v_i that corresponds to column c_i^A .

Sometimes we consider graphs $G(A)$ obtained from a matrix A with a quadratic and symmetric *distance matrix* D of size $|V(G)| \times |V(G)|$, such that $D_{i,j}$ is the *length* of the edge between v_i and v_j . A *tour* T in $G(A)$ is a sequence of vertices $(v_{i_1}, \dots, v_{i_n})$ that contains each vertex of $G(A)$ exactly once. (We do not require adjacency, as $G(A)$ is complete.) The *length* of T in $G(A)$ under a distance matrix D is $D_{i_n, i_1} + \sum_{k=1}^{n-1} D_{i_k, i_{k+1}}$. Finding a tour of minimum length in $G(A)$ under a distance matrix D is known as the *Travelling Salesperson Problem* (TSP) and is NP-complete [27].

3 Complexity of Linear Diagrams

The most important quality aspect supporting the cognitive effectiveness of linear diagrams is the number of line segments [28]. To minimize the number of line segments that have to be drawn, we have to find an appropriate horizontal ordering of the overlaps. There is a one-to-one correspondence between linear diagrams

and binary matrices: Let $(\mathcal{S}, \mathcal{U})$ be a set system with universe $\mathcal{U} = \{u_1, \dots, u_n\}$ and sets $\mathcal{S} = \{S_1, \dots, S_m\}$, hence, for all $i \in [m]$, $S_i \subseteq \mathcal{U}$. The system \mathcal{S} can be represented by a binary matrix A s.t. $A_{i,j} = 1$ if and only if element u_j belongs to set S_i . The rows and columns of A are exactly the rows and columns of the linear diagram, respectively. Line segments in the linear diagram correspond to blocks of consecutive ones in the matrix A . The problem of finding a horizontal ordering of the overlaps that minimizes the number of line segments is equivalent to the problem of finding a permutation $\pi \in \Pi_n$ that minimizes $\text{cons1}(\pi(A))$.

A matrix A is said to have the *consecutive ones property* (C1P) if there is a permutation $\pi \in \Pi_{n_A}$ with $\text{splits}(\pi(A)) = 0$. There are several linear-time algorithms for testing if a matrix has the C1P and for computing the corresponding permutation, the first due to Booth and Lueker [3]. Thus, we can decide in linear time if a linear diagram can be drawn such that each set is represented by exactly one line segment.

Most of the time though, linear diagrams cannot be drawn in this way. In this case we want to minimize the number of required line segments. The corresponding binary matrix problem is known as *consecutive block minimization* in the literature, its decision problem is given below.

CONSECUTIVE BLOCK MINIMIZATION

Instance: A binary matrix A and a non-negative integer k .

Question: Does there exist a permutation $\pi \in \Pi_{n_A}$ such that $\text{cons1}(\pi(A)) \leq k$?

The problem has been shown to be NP-complete [20], even if each row contains exactly two ones [14]. We give here an alternative proof of NP-completeness for binary matrices with two ones per row and three ones per column, thus further strengthening the NP-completeness result.

Theorem 1. CONSECUTIVE BLOCK MINIMIZATION is NP-complete for matrices with two ones per row and three ones per column.

Proof. Membership in NP is evident. For hardness, we give a reduction from HAMILTONIAN PATH on graphs of degree 3, which is NP-complete [12]. HAMILTONIAN PATH asks for a given graph G , if there is a path in G that visits every vertex exactly once. Let G be an instance of HAMILTONIAN PATH such that $E(G) = \{e_1, \dots, e_m\}$ and $V(G) = \{v_1, \dots, v_n\}$ and G has degree 3. We construct an instance (A, k) of CONSECUTIVE BLOCK MINIMIZATION as follows. Let A be the incidence matrix of G , which has $n_A = |V(G)|$ columns and $m_A = |E(G)|$ rows with $A_{i,j} = 1$ if and only if $v_i \in e_j$. Clearly, this matrix has two ones per row, as each edge contains two vertices and 3 ones per column, as G has degree 3. We show that G contains a Hamiltonian path if and only if there exists a permutation π of the columns of A such that $\text{cons1}(\pi(A)) \leq 2 \cdot m - (n - 1)$.

“ \Rightarrow ”: Let $P = (v_{\ell_1}, v_{\ell_2}, \dots, v_{\ell_n})$ be a Hamiltonian path in G . We claim that $\pi = (\ell_1, \ell_2, \dots, \ell_n)$ satisfies $\text{cons1}(\pi(A)) \leq 2 \cdot m - (n - 1)$. Consider the edges $\{v_{\ell_i}, v_{\ell_{i+1}}\}$ for $1 \leq i \leq n - 1$, which exist because P is a path. As v_{ℓ_i} and $v_{\ell_{i+1}}$ are consecutive in P , the columns $c_{\ell_i}^A$ and $c_{\ell_{i+1}}^A$ are consecutive in $\pi(A)$. Thus,

the row in A corresponding to the edge $\{v_{\ell_i}, v_{\ell_{i+1}}\}$ contributes to exactly one block of consecutive ones. The remaining $m - (n - 1)$ rows can contribute to at most two blocks of consecutive ones as they only contain two 1-entries each. Together, there are at most $n - 1 + 2 \cdot (m - (n - 1)) = 2 \cdot m - (n - 1)$ blocks of consecutive ones in $\pi(A)$.

“ \Leftarrow ”: Let $\pi = (\ell_1, \ell_2, \dots, \ell_n)$ be a permutation of the columns of A that satisfies $\text{cons1}(\pi(A)) \leq 2 \cdot m - (n - 1)$. We claim that $P = (v_{\ell_1}, v_{\ell_2}, \dots, v_{\ell_n})$ is a Hamiltonian path in G . There are at least $n - 1$ blocks of consecutive ones of size two in $\pi(A)$ as otherwise $\text{cons1}(\pi(A)) > 2 \cdot m - (n - 1)$. As G is a simple graph, no two rows of A contain ones in the same columns and thus each of these blocks of consecutive ones has to start at a different column. By the pigeonhole principle, for each $1 \leq i \leq n - 1$, there exists such a block of consecutive ones that starts at the i -th column of $\pi(A)$. Hence, $\{v_{\ell_i}, v_{\ell_{i+1}}\}$ is an edge in G for all $1 \leq i \leq n - 1$, and P is a Hamiltonian path. \square

We have been made aware of the fact that this result has been proved independently previously, using the same reduction [16].

CONSECUTIVE BLOCK MINIMIZATION has been further studied from an algorithmic view. Several heuristic methods for finding permutations π with small $\text{cons1}(\pi(A))$ have been given [15, 16, 30]. Haddadi and Layouni [17] transformed CONSECUTIVE BLOCK MINIMIZATION to a travelling salesperson problem, we will go into more details on their results in Section 4.

Further variations of consecutive-ones problems that could be interesting for linear diagrams have been studied, mostly giving hardness results or polynomial algorithms assuming that some underlying parameters of the problems are constant: It has been shown that the problem of finding a permutation π of the columns of a binary matrix A such that for all $i \in [m_A]$, $\text{cons1}(r_i^A) \leq k \in \mathbb{N}$ is NP-complete [13], which translates to the problem of having at most k line segments per set in a linear diagram. Another more involved problem has been studied, called GAPPED CONSECUTIVE ONES, in which we are given a binary matrix A and want to find a permutation π of the columns of A such that for all $i \in [m_A]$, $\text{cons1}(r_i^A) \leq k \in \mathbb{N}$, and the gaps between two consecutive blocks of ones in a row of $\pi(A)$ is at most some maximum gap parameter δ [9, 24, 25]. Here gaps refer to maximal blocks of zeros between two blocks of ones.

Furthermore, there is literature devoted to turning a binary matrix into a binary matrix that has the C1P by deleting rows, deleting columns, or flipping entries (turning 1-entries into 0-entries and/or turning 0-entries into 1-entries). Dom et al. [11] give a summary of results.

CONSECUTIVE BLOCK MINIMIZATION is also related to a variant of database compression. Given a database consisting of m rows and n columns, one wants to permute the rows of the database such that the number of *runs* is minimized. Runs are essentially consecutive elements in a column that have the same value. When compressing the database, such runs can be saved using constant memory by only saving the start, end, and value of the run. The main difference to our application is that databases are normally comprised of a huge number of rows, sometimes in the millions. Thus, mainly heuristics have been studied,

in particular, heuristics that take less than quadratic time in the number of rows. In our context of linear diagrams, algorithms and heuristics consuming quadratic time in the input are no problem. Some algorithms for minimizing runs in databases are for example described in [22].

4 TSP Model

In this section, we describe the procedure of minimizing the number of line segments in a linear diagram by using a TSP model, and give a runtime optimization. We also show how to incorporate further constraints into this model.

4.1 Solving Linear Diagrams with TSP

We now present how to solve the task of minimizing the number of line segments drawn in a linear diagram. Let us start with the key lemma for our model.

Lemma 1 ([17]). *Let A be a binary matrix with n columns and let A' be the binary matrix obtained from A by appending a column of zeros to the right of A . Let $(v_{i_1}, v_{i_2}, \dots, v_{i_{n+1}})$ be a tour of length L in $G(A')$ under distance matrix $D_{i,j} = d_h(c_i^{A'}, c_j^{A'})$. Assume that $v_{i_k} = v_{n+1}$, corresponding to the appended column of zeros, and let $\pi = (i_{k+1}, \dots, i_{n+1}, i_1, \dots, i_{k-1})$. Then $L = 2 \cdot \text{cons1}(\pi(A))$.*

As discussed in Section 3, the task of minimizing line segments in linear diagrams is the same as finding a permutation π of the columns of a binary matrix A that minimizes $\text{cons1}(\pi(A))$. One way to find such a permutation is with a TSP-model as outlined by Lemma 1: Let A be a binary matrix with n columns. We construct the binary matrix A' by appending a column of zeros to the right of A . From the matrix A' , we construct the complete graph $G(A')$, such that vertices correspond to columns in A' . A distance matrix D for $G(A')$ is constructed such that $D_{i,j}$ is the Hamming distance $d_h(c_i^{A'}, c_j^{A'})$. We then compute a TSP tour $(v_{i_1}, v_{i_2}, \dots, v_{i_{n+1}})$ of minimal length in $G(A')$. Assume that v_{i_k} is the vertex corresponding to the column $c_{n+1}^{A'}$. Then, by Lemma 1, $\pi = (i_{k+1}, \dots, i_{n+1}, i_1, \dots, i_{k-1})$ is the permutation with minimal $\text{cons1}(\pi(A))$. The intuition for this is that choosing an edge $\{v_i, v_j\}$ of small length in $G(A')$ is the same as starting or ending few consecutive blocks of ones (corresponding to line segments in a linear diagram) when going from the column c_i to c_j . With this argumentation each block of consecutive ones is started and ended exactly once, and the length of the tour is $2 \cdot \text{cons1}(\pi(A))$. Note that adding the extra column at the end is necessary, as otherwise it could be that some consecutive blocks of ones, those that start at the first column or end at the last column, are “not counted in the tour”.

There is a small runtime optimization that can be applied to decrease the size of the graph $G(A')$. Columns of A that have ones in the same rows, their Hamming distance being zero, can be collapsed into a single column. The above procedure may be applied to compute the desired permutation of columns, and then

the collapsed columns can be expanded again to appear consecutively. Clearly, this does not influence the number consecutive blocks of ones in the resulting matrix. In terms of set systems, this corresponds to collapsing multiple overlaps that contain the same sets into a single representative. In an optimal linear diagram, such overlaps would never be separated.

We tested this method of computing optimal column orderings by applying a state-of-the-art TSP-solver. We will report on experimental results for real-world and previously considered set visualization instances in Section 5. Note that the same procedure has already been applied to instances from consecutive block minimization [30].

4.2 Priorities for Sets

In some contexts certain sets in a linear diagram might be considered more important than others. We would want to compute a linear diagram, in which these sets are drawn with a single line segment, but the other sets should be drawn with as few line segments as possible. It is clear that forcing more than two sets to be drawn as one line segment is not always possible, as there are binary matrices with three rows that do not have the CIP. We can solve the problem on binary matrices as a TSP model due to the following result.

Lemma 2. *Let A be a binary matrix with n columns and exactly p 1-entries and let $C_1, \dots, C_q \subseteq \{c_1, \dots, c_n\}$ be a family of non-empty sets of columns of A satisfying*

$$\exists \pi \in \Pi_n \forall k \in [q] : \text{the columns in } C_k \text{ appear consecutively in } \pi(A).$$

Let A' be the matrix obtained from A by appending a column of zeros. We consider the graph $G(A')$ with distance matrix D s.t.

$$D_{i,j} = d_h(c_i, c_j) + (2p+1) \cdot \sum_{k=1}^q |\mathbb{1}_{C_k}(c_i) - \mathbb{1}_{C_k}(c_j)|,$$

where $\mathbb{1}_{C_k}$ is the indicator function for set C_k . Let $T = (v_{i_1}, v_{i_2}, \dots, v_{i_{n+1}})$ be a tour of minimal length in $G(A')$ under distance matrix D . Let $v_{i_k} = v_{n+1}$, corresponding to the appended column of zeros. Then the permutation $\pi = (i_{k+1}, \dots, i_{n+1}, i_1, \dots, i_k)$ has the following properties

- (1) *For all $k \in [q]$ the columns in C_k appear consecutively in $\pi(A)$.*
- (2) *Of all $\pi' \in \Pi_n$ that satisfy (1), π is the one with minimum $\text{cons1}(\pi(A))$.*

Proof. Let π be the permutation as defined above. We first show by contradiction that π satisfies (1). Assume to the contrary that π does not satisfy (1) and consider any permutation π' of the columns of A that satisfies (1). This permutation exists by assumption. Consider the tour $T' = (v_{n+1}) \star \pi$. The length of T' is at most $2q(2p+1) + 2p$, as there can be at most p consecutive blocks of ones in $\pi'(A)$, each contributing two to the length of T' . The value $2q(2p+1)$

is due to the fact that we “leave” or “enter” vertices corresponding to the set of columns C_k , $1 \leq k \leq q$, exactly twice. To the contrary, the length of T is at least $2(q+1)(2p+1)$. Hence, T cannot be a tour of minimal length, yielding a contradiction. It is clear that π also satisfies (2), as increases in the length of the tour T , also increases the number of consecutive ones of the matrix $\pi(A)$ due to the same reasoning as in Lemma 1. \square

We can directly apply the above lemma to find a permutation of the columns of a matrix A with minimum blocks of consecutive ones among the permutations π that have $\text{cons1}(r_{i_1}^{\pi(A)}) = \text{cons1}(r_{i_2}^{\pi(A)}) = 1$ for $i_1, i_2 \in m_A$: We simply define $C_1 = \{j \in [n_A] \mid A_{i_1, j} = 1\}$ and $C_2 = \{j \in [n_A] \mid A_{i_2, j} = 1\}$, and apply the reduction to TSP as outlined in Lemma 2. Clearly, C_1 and C_2 satisfy the requirements of Lemma 2, as a matrix with two rows always has the C1P. In our experiments we show how adding these constraints affects the runtime and the number of blocks of consecutive ones. Note, however, that the result of Lemma 2 allows us to constrain column orders of a matrix in far more general ways.

4.3 A Weighted Version

The shortcoming of the approach described in Section 4.2 is that we can only restrict two sets to be drawn in one line segment. If we want to involve more sets in this, we can use a model with weighted sets (corresponding to rows in a binary matrix A). Then, if a weight of row r_i^A is bigger than a weight of r_j^A , it is “worse” to have more blocks of consecutive ones for r_i^A than it is for r_j^A . Formally, we are given a binary matrix A and a weight function $f : [m_A] \rightarrow \mathbb{N}$, and we want to find a permutation π of the columns of A that minimizes $\sum_{i=1}^{m_A} f(i) \text{cons1}(r_i^{\pi(A)})$. Solving this problem is straight-forward with a TSP-model: We construct the matrix A' by appending a column of zeros to the right of A . We then create a distance matrix D for $G(A')$ such that $D_{i,j} = \sum_{k=1}^{m_A} f(k) |A'_{k,i} - A'_{k,j}|$. This distance matrix corresponds to weighted Hamming distances. Then, we simply find the tour T of minimal total distance in $G(A')$ under D , and obtain the desired permutation π from T as in Lemma 1. We conduct experiments for this weighted version of consecutive block minimization in Section 5.

4.4 Hierarchical Constraints

We now to present an algorithm that allows for more general constraints on the allowed column orders of a binary matrix, restricting column orders by *PQ-trees*. We adopt the definition of PQ-trees of Burkard et al. [4], as our algorithm directly applies their results. A PQ-tree T over the set $[n]$ is a rooted, ordered tree whose *leaves* are pairwise distinct elements of $[n]$ and whose *internal* nodes are distinguished as either *P*-nodes or *Q*-nodes. The set $\text{LEAF}(T)$ denotes the leaves of T .

Every PQ-tree T represents a set $\Pi(T)$ of permutations of $\text{LEAF}(T)$ as follows. If T consists of a single leaf $i \in [n]$, then $\Pi(T) = \{(i)\}$. Otherwise, the root $r(T)$

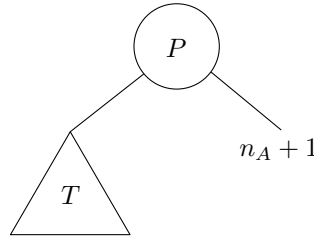


Fig. 2: Construction of PQ-tree in Theorem 2.

of T is a P -node or a Q -node. Let v_1, \dots, v_m denote the children of $r(T)$, ordered from left to right, and let T_i denote the maximal subtrees rooted at v_i , $1 \leq i \leq m$. If $r(T)$ is a P -node, then

$$\Pi(T) = \bigcup_{\psi \in \Pi_m} \Pi(T_{\psi(1)}) \star \Pi(T_{\psi(2)}) \star \dots \star \Pi(T_{\psi(m)}),$$

and if $r(T)$ is a Q -node, then

$$\Pi(T) = \Pi(T_1) \star \Pi(T_2) \star \dots \star \Pi(T_m) \cup \Pi(T_m) \star \Pi(T_{m-1}) \star \dots \star \Pi(T_1).$$

Informally, children of P -nodes can be permuted arbitrarily, while children of Q -nodes can only be reversed.

For applications of PQ-trees we refer to Booth and Lueker [3]. They can be used to model allowed column orders of a binary matrix or, equivalently, the orders of overlaps in a linear diagram; for example, if overlaps illustrated by a linear diagram have some hierarchical relations between them and should not be permuted arbitrarily, then we might represent this by a PQ-tree accordingly. If the maximum degree of the PQ-tree T that represents column orders of a binary matrix A has small maximum degree, then the permutation $\pi \in \Pi(T)$ that minimizes $\text{cons1}(\pi(A))$ can be computed efficiently.

Theorem 2. *Let A be a binary matrix and let T be a PQ-tree of maximum degree d such that $\Pi(T) \subseteq \Pi_{n_A}$. The permutation $\pi \in \Pi(T)$ that minimizes $\text{cons1}(\pi(A))$ can be found in time $\mathcal{O}(\max(m_A \cdot n_A^2, 2^d \cdot n_A^3))$.*

Proof. We apply a result of Burkard et al. [4] that states that for a PQ-tree T with maximum degree d , and an $n \times n$ distance matrix D , the shortest TSP tour for the matrix D contained in $\Pi(T)$ can be computed in $\mathcal{O}(2^d \cdot n^3)$ overall time.

Let A be a binary matrix and let T be a PQ-tree of maximum degree d such that $\Pi(T) \subseteq \Pi_{n_A}$. Let A' be the binary matrix obtained from A by appending a column of zeros to the right of A . We construct a PQ-tree T' such that $\Pi(T') \subseteq \Pi_{n_A+1}$. The PQ-tree T' consists of a P -node that has two children: The leaf $n_A + 1$ and the tree T rooted at $r(T)$ (see Figure 2). Notice that the maximum degree of T' is at most $d + 1$. Let D be the distance matrix corresponding to edge weights $D_{i,j} = d_h(c_i^{A'}, c_j^{A'})$ in $G(A')$. Due to the result of Burkard et al. we

can find in time $\mathcal{O}(2^d \cdot n_A^3)$ a tour of minimum length in $G(A')$ that is contained in $\Pi(T')$. By Lemma 1 and the construction of T' , we can obtain from this tour a permutation $\pi \in \Pi(T)$ that minimizes $\text{cons1}(\pi(A))$. We need $m_A \cdot n_A^2$ time to construct the distance matrix D , thus we need to account for the possibility that $m_A \cdot n_A^2 > 2^d \cdot n_A^3$, taking the maximum of both. \square

5 Experiments

In this section, we present an experimental evaluation of the algorithms proposed in Section 4, comparing them with state-of-the-art heuristics.

5.1 Setup and Test Data

Setup. All experiments were performed on a desktop machine with an Intel i7-8700K processor. The implementations of algorithms were done in Python 3.7. To solve our TSP models, we used the Concorde TSP solver² with the QSopt linear programming solver³. The code is available online [10].

Test data. We consider binary matrices from two different sources. The first set of instances, referred to as T_1 , is taken from Chapman et al. [7] and is available online⁴. These instances consist of 440 binary matrices with 5 sets and 10 overlaps up to 50 sets and 70 overlaps. Chapman et al. [7] provide results of their algorithms for minimizing line segments for these instances.

The second set of instances, referred to as T_2 , comes from a work by Jacobsen et al. [19] and is available online⁵. The set systems represented by these instances are taken out from a large real-world dataset coming from the Kaggle “What’s Cooking” competition [2]. The sizes of these instances range from 20 overlaps and 6 sets to 160 overlaps and 20 sets. Overall, there are a total of 4060 instances.

5.2 Computing Optimal Linear Diagrams

The first set of experiments considers the task of computing optimal linear diagrams, or equivalently, finding column orderings of the instances that minimize the number of blocks of consecutive ones.

Algorithms. We include comparisons of the following algorithms.

- TSPConcorde: This algorithm from Section 4.1 uses our TSP model and the Concorde TSP solver to solve the problem optimally. The reported runtimes include generating input files for the Concorde solver and reading its output.

² <https://www.math.uwaterloo.ca/tsp/concorde.html>

³ <http://www.math.uwaterloo.ca/~bico/qsopt/>

⁴ <https://doi.org/10.17869/emu.2021.2748170>

⁵ <https://osf.io/nvd8e/>

- HeuristicRodgers: This algorithm is a python implementation of a greedy algorithm by Rodgers et al. [28]. A pairwise similarity measure between overlaps is defined, and then an overlap order is computed iteratively, trying to place similar overlaps next to each other. Rodgers et al. provide an online demo that implements this algorithm⁶.
- Supervenn: This algorithm is from a recent GitHub project [18]. For a set of 10000 seeds it defines a pairwise similarity measure between overlaps and then applies a heuristic to compute an overlap order.
- BestChapman: Chapman et al. [7] compare several heuristic methods to compute overlap orderings of linear diagrams that minimize the drawn line segments. They report the number of line segments of overlap orders computed by their algorithms for test set T_1 . As they do not provide the code for all algorithms, and the explanation of the remaining algorithms is incomplete, we had to restrict the evaluation of their approaches to test set T_1 . For an instance of T_1 , we assume that the algorithm BestChapman is any algorithm of Chapman et al. that computes an overlap ordering with the least amount of blocks of consecutive ones. They do not provide the runtimes of their algorithms in their abstract [7], so we cannot either.

Comparison. TSPConcorde by design computes optimal column orderings. Hence, we report the relative and absolute optimality gaps for the other algorithms. That is, let $\text{blocks}(\mathcal{A}, I)$ be the number of blocks of consecutive ones of a column ordering computed by algorithm \mathcal{A} for instance I . Then the relative optimality gap in percent is $100 \cdot \left(\frac{\text{blocks}(\mathcal{A}, I)}{\text{blocks}(\text{TSPConcorde}, I)} - 1 \right)$ and the absolute optimality gap is $\text{blocks}(\mathcal{A}, I) - \text{blocks}(\text{TSPConcorde}, I)$. For a set of instances, we report these value averaged. For TSPConcorde we provide the average number of consecutive blocks of ones per row, as the optimality gap is always zero. We also provide the mean runtime for the same set of instances. Results are broken down by the number of columns, as the factorial of the number of columns determines the size of the possible search space for an algorithm.

Test set T_1 . Table 1 shows the results for test set T_1 . The simple heuristic of Rodgers et al. [28] has the smallest runtimes, while also performing worst with regard to optimality gaps. The runtimes of Supervenn are rather high, while the optimality gaps are lower when compared to HeuristicRodgers, resulting from the 10000 runs of a heuristic, each skewed with a different seed value. While the problem of consecutive block minimization is NP-complete, TSPConcorde solved all instances optimally. The average runtime for the largest class of instances from T_1 is still less than a second. It is worth mentioning that optimality gaps of mostly under 10% indicate that the heuristics are quite good.

The heuristics of Chapman et al. [7] solved 340 of the 440 instances optimally. For the remaining instances, the maximum difference between the optimal number of consecutive blocks and their best solution is 3. This yields the fairly small optimality gaps for BestChapman, while we expect that these values would increase for larger instances, a pattern that just starts to appear in Table 1.

⁶ <http://www.eulerdiagrams.com/linear/generator/>

Table 1: Results for test set T_1 . White columns depict the mean relative/absolute optimality gaps (except TSPConcorde); gray columns depict the mean runtimes. For the algorithms of Chapman we do not know the runtimes.

#columns	TSPConcorde		HeuristicRodgers		Supervenn		Best Chapman
	blocks / row	t [ms]	gap [rel./abs.]	t [ms]	gap [rel./abs.]	t [ms]	gap [rel./abs.]
10	1.7	7	3.3/0.9	0	1.6/0.8	853	0.0/0.0
20	2.8	13	6.0/3.0	1	3.3/1.8	1360	0.0/0.0
30	4.0	22	6.0/5.2	1	3.4/3.1	1861	0.2/0.3
50	6.0	69	6.9/9.3	4	3.7/5.6	2969	0.4/0.5
70	7.9	340	8.1/13.3	7	4.7/8.0	4192	0.5/0.8

Table 2: Results for test set T_2 . White columns depict the mean relative/absolute optimality gaps (except TSPConcorde); gray columns depict the mean runtimes.

#columns	TSPConcorde		HeuristicRodgers		Supervenn	
	blocks / row	t [ms]	gap [rel./abs.]	t [ms]	gap [rel./abs.]	t [ms]
20-50	1.7	17	8.4/2.0	1	7.7/1.8	1949
55-80	1.8	23	10.0/2.5	2	8.4/2.2	3642
85-110	1.9	36	10.9/2.8	4	9.2/2.5	5408
115-140	2.0	82	11.1/3.1	6	9.8/2.8	7782
145-160	2.0	71	10.7/3.0	9	9.8/2.9	10133

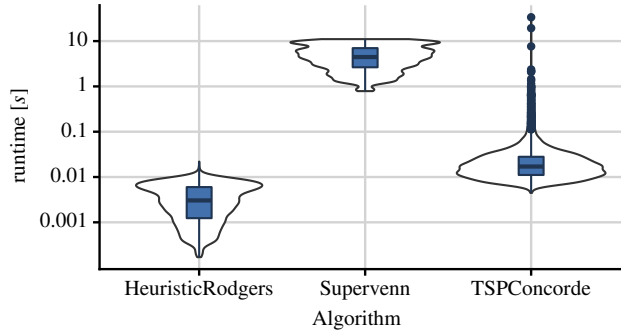


Fig. 3: Violin and box plot showing runtimes for all instances from T_1 and T_2 .

Test set T_2 . Table 2 shows results for test set T_2 . TSPConcorde is able to solve all instances optimally, the mean runtime still being well below 100ms, even for instances with up to 160 columns. For Supervenn and HeuristicRodgers we see similar results as in the previous test set. While Supervenn has slightly

better optimality gaps, HeuristicRodgers takes only a thousandth of the time of Supervenn. Again, optimality gaps increase with increasing number of columns to about 10 percent compared to the optimal solutions.

Runtimes. Figure 3 shows a boxplot and violin plot of the runtimes of the three algorithms HeuristicRodgers, Supervenn, and TSPConcorde for the combined test set $T_1 \cup T_2$. The y -axis is scaled logarithmically. It again reflects that Supervenn takes much longer than HeuristicRodgers, while the runtimes for both algorithms do not contain outliers as their runtime is rather “deterministic”, in the sense that their runtime is accurately represented as a polynomial function of the number of columns of an instance. On the contrary, the runtimes of TSPConcorde contain a multitude of outliers, while most runtimes are still below 100ms. Only two instances take more than 10 seconds to solve.

5.3 Constraints

Next, we present experiments on how constraints on the column order affect the runtime and the number of blocks of TSPConcorde. Namely, we implemented the constraints from Sections 4.2 and 4.3 that either specify that two sets/rows have to be represented as a single line segment/consecutive blocks of ones, or give specific weights to sets. The evaluation for both constraints works as follows.

- Two sets as single line segment: We pick uniformly at random for each instance in our test set $T_1 \cup T_2$ two sets that have to be drawn as a single line segments, and then apply the reduction to TSP described in Section 4.2, and solve the resulting TSP-instance with the Concorde TSP-solver. We identify this approach by TSPConcordeFS for “fixed sets”.
- Weighted sets: For each matrix A in the test set $T_1 \cup T_2$ we specify a weight function $f : [m_A] \rightarrow \mathbb{N}$ that assigns to each set a unique integer weight in $[m_A]$ uniformly at random. Then, we apply the reduction to TSP as described in Section 4.3 and solve the resulting TSP-instance with the Concorde TSP-solver. We identify this approach by TSPConcordeW for “weighted”.

Table 3 shows runtimes and optimality gaps for test set T_1 . We observe that adding constraints does not influence runtimes of the TSP solver significantly. Furthermore, by adding constraints we may not be able to reach the optimal number of line segments anymore and see a maximum optimality gap of 10%. The results for test set T_2 are similar and are given in Table 4.

Figure 4 shows a box and violin plot of runtimes for TSPConcorde and the constrained versions thereof, further suggesting that adding constraints does not significantly influence runtimes.

6 Conclusion

We have studied the algorithmic complexity of computing optimal linear diagrams and observed that it is equivalent to a related problem on binary matrices.

Table 3: Results for test set T_1 and constrained versions. White columns depict mean relative/absolute optimality gaps (except TSPConcorde); gray columns depict mean runtimes.

#columns	TSPConcorde		TSPConcordeFS		TSPConcordeW	
	blocks / row	t [ms]	gap [rel./abs.]	t [ms]	gap [rel./abs.]	t [ms]
10	1.7	7	4.1/1.7	8	2.6/0.9	7
20	2.8	13	4.8/3.7	17	4.5/3.0	11
30	4.0	22	6.2/6.7	37	6.5/5.9	25
50	6.0	69	6.4/10.2	104	8.7/12.1	61
70	7.9	340	7.0/14.8	210	9.4/16.7	84

Table 4: Results for test set T_2 and constrained versions. White columns depict the mean relative and absolute optimality gaps (except TSPConcorde); gray columns depict the mean runtimes.

#columns	TSPConcorde		TSPConcordeFS		TSPConcordeW	
	blocks / row	t [ms]	gap [rel./abs.]	t [ms]	gap [rel./abs.]	t [ms]
20-50	1.7	17	3.6/0.9	28	4.0/0.9	16
55-80	1.8	23	3.1/0.8	36	4.8/1.2	27
85-110	1.9	36	3.0/0.8	57	5.2/1.4	30
115-140	2.0	82	3.2/0.9	79	6.0/1.7	40
145-160	2.0	71	3.1/0.8	99	6.2/1.7	38

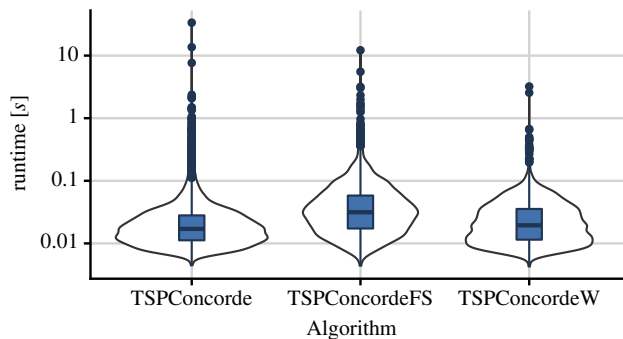


Fig. 4: Runtimes of constrained algorithms for all instances from T_1 and T_2 .

Despite its NP-completeness, even in a restricted setting, we have formulated a TSP model for solving the problem optimally. In an experimental study, we have seen that a state-of-the-art TSP solver can in fact solve a large set of instances obtained from our model optimally, most of them within few milliseconds. Hence

it is feasible to strive for optimal linear diagrams in most practical settings and thus reduce the number of line segments by up to 10% compared to the best heuristics, which, otherwise, are faster by one to two orders of magnitude.

References

1. Alsallakh, B., Micallef, L., Aigner, W., Hauser, H., Miksch, S., Rodgers, P.: The state-of-the-art of set visualization. *Computer Graphics Forum* **35**(1), 234–260 (2016). doi:10.1111/cgf.12722
2. Amburg, I., Veldt, N., Benson, A.: Clustering in graphs and hypergraphs with categorical edge labels. In: *The Web Conference (WWW’20)*. pp. 706–717. ACM (2020). doi:10.1145/3366423.3380152
3. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.* **13**(3), 335–379 (1976). doi:10.1016/S0022-0000(76)80045-1
4. Burkard, R.E., Deineko, V.G., Woeginger, G.J.: The travelling salesman and the PQ-Tree. *Math. Oper. Res.* **23**(3), 613–623 (1998). doi:10.1287/moor.23.3.613
5. Chapman, P.: Interactivity in Linear Diagrams. In: Basu, A., Stapleton, G., Linker, S., Legg, C., Manalo, E., Viana, P. (eds.) *Diagrammatic Representation and Inference (Diagrams’21)*. LNCS, vol. 12909, pp. 449–465. Springer (2021). doi:10.1007/978-3-030-86062-2_47
6. Chapman, P., Sim, K., Chen, H.H.: Minimising line segments in linear diagrams is NP-hard. *J. Comput. Lang.* **71**, 101136 (2022). doi:10.1016/j.cola.2022.101136
7. Chapman, P., Sim, K., Chen, H.: Drawing algorithms for linear diagrams. In: *Talk Abstracts of Diagrams 2021*. pp. 1–3 (2021), <http://www.diagrams-conference.org/2021/wp-content/uploads/2021/08/Peter-Chapman-6-chapman.pdf>
8. Chapman, P., Stapleton, G., Rodgers, P., Micallef, L., Blake, A.: Visualizing sets: An empirical comparison of diagram types. In: Dwyer, T., Purchase, H., De-laney, A. (eds.) *Diagrammatic Representation and Inference (Diagrams’14)*. LNCS, vol. 8578, pp. 146–160. Springer (2014). doi:10.1007/978-3-662-44043-8_18
9. Chauve, C., Mañuch, J., Patterson, M.: On the gapped consecutive-ones property. *Electron. Notes Discret. Math.* **34**, 121–125 (Aug 2009). doi:10.1016/j.endm.2009.07.020
10. Dobler, A.: On computing optimal linear diagrams: Code (2022). doi:10.5281/zenodo.6637911
11. Dom, M., Guo, J., Niedermeier, R.: Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. *J. Comput. Syst. Sci.* **76**(3), 204–221 (2010). doi:10.1016/j.jcss.2009.07.001
12. Garey, M.R., Johnson, D.S., Tarjan, R.E.: The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.* **5**(4), 704–714 (1976). doi:10.1137/0205049
13. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. *J. Comput. Biol.* **2**(1), 139–152 (1995). doi:10.1089/cmb.1995.2.139
14. Haddadi, S.: A note on the NP-hardness of the consecutive block minimization problem. *Int. Trans. Op. Res.* **9**(6), 775–777 (2002). doi:10.1111/1475-3995.00387
15. Haddadi, S.: Exponential neighborhood search for consecutive block minimization. *Int. Trans. Op. Res.* (2021). doi:10.1111/itor.13065
16. Haddadi, S.: Iterated local search for consecutive block minimization. *Comput. Oper. Res.* **131**, 105273 (2021). doi:10.1016/j.cor.2021.105273

17. Haddadi, S., Layouni, Z.: Consecutive block minimization is 1.5-approximable. *Inf. Process. Lett.* **108**(3), 132–135 (2008). doi:10.1016/j.ipl.2008.04.009
18. Indukaev, F.: Supervenn python package (v0.3.2) (2021). doi:10.5281/zenodo.4424381
19. Jacobsen, B., Wallinger, M., Kobourov, S., Nöllenburg, M.: MetroSets: Visualizing sets as metro maps. *IEEE Trans. Vis. Comput. Graph.* **27**(2), 1257–1267 (2021). doi:10.1109/TVCG.2020.3030475
20. Kou, L.T.: Polynomial complete consecutive information retrieval problems. *SIAM J. Comput.* **6**(1), 67–75 (1977). doi:10.1137/0206004
21. Lamy, J.B., Berthelot, H., Capron, C., Favre, M.: Rainbow boxes: A new technique for overlapping set visualization and two applications in the biomedical domain. *J. Vis. Lang. Comput.* **43**, 71–82 (2017). doi:10.1016/j.jvlc.2017.09.003
22. Lemire, D., Kaser, O., Gutarra, E.: Reordering rows for better compression: Beyond the lexicographic order. *ACM Trans. Database Syst.* **37**(3), 20:1–20:29 (2012). doi:10.1145/2338626.2338633
23. Luz, S., Masoodian, M.: A comparison of linear and mosaic diagrams for set visualization. *Inf. Vis.* **18**(3), 297–310 (2019). doi:10.1177/1473871618754343
24. Mañuch, J., Patterson, M.: The complexity of the gapped consecutive-ones property problem for matrices of bounded maximum degree. *J. Comput. Biol.* **18**(9), 1243–1253 (2011). doi:10.1089/cmb.2011.0128
25. Mañuch, J., Patterson, M., Chauve, C.: Hardness results on the gapped consecutive-ones property problem. *Discret. Appl. Math.* **160**(18), 2760–2768 (2012). doi:10.1016/j.dam.2012.03.019
26. Masoodian, M., Koivunen, L.: Temporal visualization of sets and their relationships using time-sets. In: *Information Visualisation (IV'18)*. pp. 85–90. IEEE (2018). doi:10.1109/iV.2018.00025
27. Papadimitriou, C.H.: The Euclidean travelling salesman problem is NP-complete. *Theor. Comput. Sci.* **4**(3), 237–244 (1977). doi:10.1016/0304-3975(77)90012-3
28. Rodgers, P., Stapleton, G., Chapman, P.: Visualizing sets with linear diagrams. *ACM Trans. Comput. Hum. Interact.* **22**(6), 1–39 (2015). doi:10.1145/2810012
29. Sato, Y., Mineshima, K.: The efficacy of diagrams in syllogistic reasoning: A case of linear diagrams. In: Cox, P., Plimmer, B., Rodgers, P. (eds.) *Diagrammatic Representation and Inference (Diagrams'12)*. LNCS, vol. 7352, pp. 352–355. Springer (2012). doi:10.1007/978-3-642-31223-6_49
30. Soares, L.C.R., Reinsma, J.A., Nascimento, L.H.L., Carvalho, M.A.M.: Heuristic methods to consecutive block minimization. *Comput. Oper. Res.* **120**, 104948 (2020). doi:10.1016/j.cor.2020.104948
31. Stapleton, G., Chapman, P., Rodgers, P., Touloumis, A., Blake, A., Delaney, A.: The efficacy of Euler diagrams and linear diagrams for visualizing set cardinality using proportions and numbers. *PLOS ONE* **14**(3), e0211234 (2019). doi:10.1371/journal.pone.0211234