COMPUTAÇÃO COMPLETA EM TEMPO SUBQUADRÁTICO DE UM NOVO TIPO DE BICLUSTER GENÉRICO EM MATRIZES DENSAS E ESPARSAS

BERNARDO DE ALMEIDA ABREU

COMPUTAÇÃO COMPLETA EM TEMPO SUBQUADRÁTICO DE UM NOVO TIPO DE BICLUSTER GENÉRICO EM MATRIZES DENSAS E ESPARSAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Loïc Pascal Gilles Cerf

Belo Horizonte

Abril de 2021

BERNARDO DE ALMEIDA ABREU

COMPLETE COMPUTATION IN SUBQUADRATIC TIME OF A NEW GENERIC TYPE OF BICLUSTER IN DENSE AND SPARSE MATRICES

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Loïc Pascal Gilles Cerf

Belo Horizonte

April 2021

Abreu, Bernardo de Almeida

A162c Complete computation in subquadratic time of a new generic type of bicluster in dense and sparse matrices / Bernardo de Almeida Abreu. — Belo Horizonte, 2021
xviii, 93 f. : il. ; 29cm

Orientador: Loïc Pascal Gilles Cerf.

Dissertação (mestrado) — Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 88-93

1.Computação – Teses. 2. Mineração de dados Teses. 3. Programação dinâmica – Teses. 4. Problemas de enumeração combinatória – Teses. 5. Biclustering – Teses. I. Cerf, Loïc Pascal Gilles. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV. Título.

CDU 519.6*73(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz - CRB 6ª Região nº 819.

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Complete Computation in Subquadratic Time of a New Generic Type of Bicluster in Dense and Sparse Matrices

BERNARDO DE ALMEIDA ABREU

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LOIC PASCAL GILLES CERF - Orientador
Departamento Ciência da Computação - UFMG

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

PROF. RENATO VIMIEIRO
Departamento de Ciência da Computação - UFMG

PROF. JOÃO PAULO ATAIDE MARTINS
Departamento de Química - UFMG

Belo Horizonte, 16 de Abril de 2021.

To family and friends.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. Additionally it was also financed in part by the Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) through the project APQ-04224-16 of Multilateral Collaboration FAPEMIG/CNRS, and by the Fundação de Desenvolvimento da Pesquisa (FUNDEP).

Resumo

Biclustering é uma técnica definida como a clusterização simultânea de linhas e colunas de uma matriz de dados. Dada uma matriz real m-por-n, esse trabalho define um novo tipo de bicluster: em qualquer uma de suas colunas, os valores das linhas do bicluster devem ser estritamente maiores do que os valores em todas as linhas ausentes do mesmo. As linhas que formam um bicluster não podem ser um subconjunto ou um superconjunto das linhas contidas em outro bicluster de maior qualidade. A qualidade de um bicluster é um valor real dado por qualquer função computável, tornando a definição genérica. "*Muscly patterns*" são os biclusters que respeitam a definição dada.

`Biceps` é proposto para listar exaustivamente os *muscly patterns* em tempo subquadrático, enumerando os biclusters possíveis em um DAG e selecionando aqueles de maior qualidade. O algoritmo é dividido em três fases: Durante a primeira fase, o DAG é construído tal que seus vértices representem biclusters e suas arestas representem uma relação de inclusão entre as linhas dos biclusters conectados. Durante a segunda fase, programação dinâmica é utilizada para descobrir biclusters melhores que predecessores ou sucessores que têm necessariamente um subconjunto ou superconjunto de linhas. Apesar disso, as arestas do grafo não representam todas as possíveis relações de inclusão, logo é necessária uma terceira fase para encontrar somente os *muscly patterns* entre os biclusters obtidos pela segunda fase.

Os biclusters são listados em tempo $O(m^2n + mn^2)$, mais o tempo para se computar $O(mn)$ qualidades. Após algumas adaptações, `Biceps` permanece subquadrático se sua complexidade é expressada em função de $m_{\text{non-min}}n$, onde $m_{\text{non-min}}$ é o número máximo de valores não-mínimos em uma coluna (matrizes esparsas). Experimentos em três datasets do mundo real demonstram a efetividade da proposta em diferentes aplicações. Também mostram uma boa eficiência prática: 2 min e 5.27 GB de RAM são suficientes para listar todos os biclusters em uma matriz densa de 801 × 20,531; 3.5s e 192 MB de RAM para uma matriz esparsa de 631,532 × 174,559 com 2,575,425 valores não-nulos.

Palavras-chave: Biclustering, Programação Dinâmica, Enumeração Exaustiva, Biclusters com colunas-exclusivas, Complexidade subquadrática.

Abstract

Biclustering is a technique that is defined as a simultaneous clustering of rows and columns of a data matrix. Given a m-by-n real matrix, this work defines a new type of bicluster: in any of its columns, the values on the rows of the bicluster must be all strictly greater than those on the rows absent from it. The rows that form a bicluster can't be a subset or superset of the rows contained in a different bicluster of greater quality. The quality of a bicluster is a real value assigned from any computable function, making the bicluster definition generic. "Muscly patterns" are biclusters that comply with this definition.

`Biceps` is proposed to exhaustively list the muscly patterns in subquadratic time, by enumerating possible biclusters in a DAG and selecting those with greater qualities. The algorithm is divided in three phases: During the first phase, a DAG is built where the vertices represent biclusters and edges represent an inclusion relationship between the rows of connected biclusters. During the second phase, dynamic programming is employed to discover biclusters that are better than any of its predecessor or successors, which necessarily have a subset or superset of rows. Nonetheless, the edges in the graph do not represent all possible inclusion relationships, thus a third phase is required to discover only the muscly patterns among the bicluster obtained by the second phase.

The biclusters are listed within $O(m^2n + mn^2)$ time, plus the time to compute $O(mn)$ qualities. After some adaptations, the proposed algorithm, `Biceps`, remains subquadratic if its complexity is expressed in function of $m_{\text{non-min}}n$, where $m_{\text{non-min}}$ is the maximal number of non-minimal values in a column, i. e., for sparse matrices. Experiments on three real-world datasets demonstrate the effectiveness of the proposal in different application contexts. They also show its good theoretical efficiency is practical as well: two minutes and 5.27 GB of RAM are enough to list the desired biclusters in a dense 801-by-20,531 matrix; 3.5s and 192 MB of RAM for a sparse 631,532-by-174,559 matrix with 2,575,425 nonzero values.

Keywords: Biclustering, Dynamic Programming, Exhaustive Enumeration, Exclusive-columns Biclusters, Subquadratic Complexity.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Cluster analysis, or clustering, is a fundamental tool in machine learning and data mining, and has been applied to many different problems and fields over the years. The task of clustering consists in partitioning a dataset into groups, called clusters, such that the data points in each cluster are more similar to each other than to the rest of the data [Mirkin, 1996]. The goal of clustering may vary depending on the application, sometimes it is used to better understand the data, while in other applications it may represent a first step in solving a more complex problem.

One class of clustering algorithms is that of biclustering. The goal of biclustering is to perform a clustering of rows and of columns of a matrix simultaneously, instead of clustering each dimension separately, thus discovering submatrices. Traditional clustering techniques can be applied to either the rows or the columns of a matrix separately, thus deriving a global model. Biclustering, contrastingly, perform clustering on both dimensions, thus deriving a local model, as the clustering of a set of rows is performed on a subspace defined by a subset of columns, or vice-versa [Madeira and Oliveira, 2004].

Therefore, biclustering approaches are useful when applied to situations where the hidden patterns in the data only exist in certain combinations of rows and columns. One field in which biclustering is much used is gene expression data analysis, in which a dataset is a matrix of genes by conditions [Cheng and Church, 2000]. Biclustering supports the identification of some interesting cellular processes which are only active for some subset of conditions, with the participation of only a subset of genes. Nevertheless, biclustering is not limited to gene expression data analysis. There are multiple applications, and the methods used to discover biclusters and evaluate their quality depend on the application.

This work proposes a new type of bicluster. Its definition is simple and relevant in

various application contexts: a bicluster in the input $m \times n$ matrix is output if and only if 1) it has between $k \in \mathbb{N}$ and $m-k$ rows, 2) at least one column, 3) any of its columns is such that the values in the rows of the bicluster are all strictly greater than those out, and 4) any other bicluster satisfying the previous conditions and involving a subset or a superset of rows is worse. To deem a bicluster "worse" than another one, grades are compared. A user-defined function returns the grade of a bicluster. Consequently, the proposed bicluster definition is generic, allowing for quality functions that are suitable to various specific problems.

To more concretely present and motivate the proposed definition, Figure 1.1 shows a toy matrix. Here, every real number in the matrix is an integer that represents how many times a twitter hashtag (a row, e. g., r_3) was sent from a borough of New York City (a column, e. g., c_1) during the data collection. The values of the rows are $r = (\texttt{\#santa}, \texttt{\#rockefellercenter}, \texttt{\#christmas}, \texttt{\#longisland})$, and the values of the columns are $c = (\texttt{Manhattan}, \texttt{Bronx}, \texttt{Staten Island}, \texttt{Brooklyn}, \texttt{Queens})$. For instance, at the intersection of $r_3 = \texttt{\#christmas}$ and $c_1 = \texttt{Manhattan}$, the number 55 would mean that the hashtag `#christmas` was sent 55 times from devices geolocalized in `Manhattan`.

	c_1	c_2	c_3	c_4	c_5
r_1	45	38	10	9	6
r_2	35	6	0	27	11
r_3	55	32	4	13	11
r_4	5	20	0	23	29

Figure 1.1: Toy matrix

In the informal definition given two paragraphs above, the first two conditions force any output bicluster to correspond to a nonempty submatrix that splits the rows of the input matrix into two large-enough clusters, with at least $k \in \mathbb{N}$ rows each: the rows *in* the submatrix and those *out* of it. In Figure 1.2, the highlighted bicluster has two rows and three columns. Assuming $k = 1$, it satisfies the first two conditions, because $1 \leq 2 \leq 4 - 1$ and $3 \geq 1$.

Given the rows of a bicluster, any of its columns is characterized by the third point of the definition: if the bicluster has $i \in \{k, \ldots, m-k\}$ rows, then the i highest values in the column must be in these rows and the i^{th} value must be different from the $(i+1)^{\text{th}}$. In the example, $i = |\{r_1, r_3\}| = 2$ and three columns of the matrix have their two highest values in the first and third row: c_1, c_2 and c_3. They are all the boroughs where the hashtags r_1 and r_3 were sent more than the remaining hashtags, r_2 and r_4. More precisely, in any of those three boroughs, the least-sent hashtag among

	c_1	c_2	c_3	c_4	c_5
r_1	45	38	10	9	6
r_2	35	6	0	27	11
r_3	55	32	4	13	11
r_4	5	20	0	23	29

(a) Original toy matrix

c_1	c_2	c_3	c_4	c_5
$r_3, 55$	$r_1, 38$	$r_1, 10$	$r_2, 27$	$r_4, 29$
$r_1, 45$	$r_3, 32$	$r_3, 4$	$r_4, 23$	$r_2, 11$
$r_2, 35$	$r_4, 20$	$r_2, 0$	$r_3, 13$	$r_3, 11$
$r_4, 5$	$r_2, 6$	$r_4, 0$	$r_1, 9$	$r_1, 6$

(b) Sorted toy matrix

Figure 1.2: Highlighted bicluster in the toy matrix

r_1 and r_3 was still sent more than the most-sent hashtag among r_2 and r_4. Because r_1 is, like r_3, a Christmas-related hashtag (`#santa`), and the other hashtags are not, c_1, c_2 and c_3 are probably boroughs where Christmas-related events took place during the data collection. This work proves that Conditions 2 and 3 define a search space that contains at most $(m - 2k + 1)n$ candidate biclusters.

The last condition discards from that search space every candidate bicluster whose grade does not exceed that of another candidate with a subset or a superset of rows. As mentioned earlier, a user-defined function assigns the grades. It may be any computable function that, given a bicluster, returns a real number, and it may be tailored to the particularities of the application. In the example, the locations of the boroughs in a bicluster may influence its grade, perhaps to favor the biclusters with nearby boroughs. The grade may depend on the matrix too. To discover biclusters that stand out, it makes sense, given a bicluster with i rows, to take into account the differences between the $(i + 1)^{\text{th}}$ and the i^{th} highest values in every column of the matrix that is involved in the bicluster. In fact, the analyst may simply define the grade as the sum of those differences. If so, the bicluster highlighted in Figure 1.2a is graded $(45 - 35) + (32 - 20) + (4 - 0) = 26$ and deemed worse than the bicluster $(\{r_1, r_2, r_3\}, \{c_1\})$, which is another candidate (assuming $k = 1$) with a higher grade: $35 - 5 = 30$. Since $\{r_1, r_3\} \subset \{r_1, r_2, r_3\}$, the last condition in the definition would therefore discard the highlighted bicluster. Because r_2 is `#rockefellercenter`, the location of the eponymous complex explains why r_2 is mostly sent from $c_1 =$ `Manhattan` and the Christmas tree in the center explains its association with $r_1 =$ `#santa` and $r_3 =$ `#christmas`, which are not location-specific. The biclusters $(\{r_1, r_3\}, \{c_1, c_2, c_3\})$ and $(\{r_1, r_2, r_3\}, \{c_1\})$ intersect. They partly carry the same information. This work proves that intersecting biclusters always have nested sets of rows. Condition 4 therefore eliminates redundancy of information in the output collection of biclusters.

After formally defining the proposed biclusters, it is possible to define an algorithm that exhaustively lists such biclusters. The proposed algorithm, called `Biceps`, is

complete, as it discovers all the biclusters satisfying the formal definition. `Biceps` has a subquadratic running time complexity. The algorithm is divided into three phases: During the first phase, a direct acyclic graph (DAG) is built, where the vertices represent biclusters contained in the search space defined by Conditions 1, 2 and 3, and edges represent an inclusion relationship between the rows of the biclusters. During the second phase, dynamic programming is employed to discover every bicluster that is better than any of its predecessors or successors, which necessarily represent biclusters with a proper subset or superset of rows. Nonetheless, the edges in the graph do not represent all possible inclusion relationships. A third phase is required to discover only the biclusters defined by Condition 4.

The proposed definition of a new type of bicluster presents a search space that is polynomial, contrasting with the bicluster definitions, and its algorithms, existing in the literature, which typically present decision variations which are NP-Complete or NP-Hard. Given that the search space is polynomial, the discovery of biclusters is fast, which is reflected in the proposed algorithm. This type of bicluster also deals with rows that exhibit the top values for certain columns, and stand out in comparison to the other rows in this column given a generic quality function. This type of bicluster, which focuses on top rows that stand out in comparison to rows outside of the bicluster has not yet been explored in the literature, and the possibilities presented by a generic quality function allow this new type of bicluster to be used in numerous applications.

This thesis is organized as follows: Chapter 2 presents some biclustering concepts, a classification of the different types of biclusters, as well as validation techniques and some existing algorithms. It focuses on algorithms with similar approaches to the new type of bicluster proposed here. Chapter 3 proves the properties mentioned in this introduction and formalizes the problem. Chapter 4 mathematically specifies the algorithm's three steps, and proves they exactly solve the stated problem. It shows as well that `Biceps` runs in subquadratic time. Furthermore, it explains adaptations for sparse matrices and details technical improvements to the algorithm. Chapter 5 demonstrates `Biceps`' effectiveness and efficiency on three real-world datasets dealing with twitter usage, chemistry and gene expression analysis. Finally, Chapter 6 concludes the work and suggests future work.

Chapter 2

Related Work

This Chapter presents the concept of biclustering as well as traditional classifications of types and structures of biclusters, and classifications of the algorithms employed to discover them. To the best of our knowledge, this thesis proposes a new type of bicluster. Thus there are no works in the revised literature which cover exactly that type of bicluster. Nevertheless, there are works with similar classifications to the one proposed. Section 2.1 presents a general definition of the term biclustering, as well as some of the applications of biclustering. Section 2.2 presents the traditional classifications of biclustering structures, regarding how biclusters are organized in the matrix. Section 2.3 presents the traditional types of biclusters, as defined by Madeira and Oliveira [2004], which are related to the underlying characteristics of the bicluster. Section 2.4 presents some algorithms that employ an exhaustive enumeration approach to discover biclusters. Section 2.5 presents some strategies to validate a biclustering. Finally, Section 2.6 presents a summary of the chapter, classifying the new type of bicluster proposed based on the classifications and algorithms presented.

2.1 Biclustering and Its Applications

The term biclustering is defined by Mirkin [1996] as a simultaneous clustering of sets of rows and sets of columns of a data matrix. More formally, given an m by n matrix, where $X = \{x_1, \ldots, x_m\}$ is its set of rows and $Y = \{y_1, \ldots, y_n\}$ its set of columns, a bicluster is a pair (I, J) such that $I \subseteq X$ and $J \subseteq Y$. It is typically required that the submatrix formed by the pair (I, J) is either over-expressed (i. e., includes values above the average of the matrix), or presents some sort of homogeneity. However, in general, the term biclustering may refer to an arbitrarily-defined collection of patterns, each associating a subset of rows with a subset of columns in any data matrix [Madeira

and Oliveira, 2004; Busygin et al., 2008].

The problem of listing biclusters was first introduced in Hartigan [1972]. It has since been applied to multiple areas. One field in which biclustering is much used is the analysis of biological data, such as gene expression data. Cheng and Church [2000] first proposed that application. Since then, multiple algorithms have been developed for that purpose [Madeira and Oliveira, 2004; Tanay et al., 2005; Padilha and Campello, 2017], i. e., to discover subsets of genes (the rows of the bicluster) that are only expressed under some specific conditions (its columns). The traditional clustering techniques rely on similarities computed over all conditions in the matrix, thus they are unable to discover genes that are only co-expressed in some of the conditions.

In the text mining literature, biclustering is also known as co-clustering. It is used to identify subsets of words (the rows of the bicluster) associated with subsets of documents (its columns) [Dhillon, 2001]. The values in the data matrix are weights for each word in a document. A graph partitioning technique proposed by Dhillon [2001] and Rege et al. [2006], involves modeling a matrix of words by documents as a bipartite graph, whose edges are weighted with the frequency of terms in a given document. The graph is partitioned so as to minimize the weight of the crossing edges between the partitions. A more common technique minimizes the loss of mutual information between two random variables that represent the clusters of rows and columns [Dhillon et al., 2003; Sim et al., 2009; Gao et al., 2006; Greco et al., 2007].

Other applications of biclustering deal with multimedia data processing and retrieval. For instance, biclustering techniques were developed for real-time rendering [Sun et al., 2011] and video document retrieval [Goyal et al., 2010]. Other uses include collaborative filtering [Wang et al., 2002; Hofmann and Puzieha, 1999; Ungar and Foster, 1998] to find subsets of customers with a similar behavior toward a subset of products and dimensionality reduction [Benczúr et al., 2007].

2.2 Bicluster Structures

Regular clustering techniques result in clusters with a full coverage of the data matrix and no overlap, i. e., when clustering the rows of a matrix, any row will belong to one and only one cluster. In contrast, biclustering allows for more flexible structures [Madeira and Oliveira, 2004]. Biclusters can overlap. That overlap, when analyzing gene expression, allows the discovery of a gene present in multiple biclusters that is not co-active under all conditions for all of these biclusters [Cheng and Church, 2000]. Indeed, genes may appear in more than one bicluster, associated with different

conditions in each bicluster. Furthermore, the biclustering needs not be exhaustive, i. e., some row or column may belong to no bicluster.

Some biclustering algorithms only try to find one bicluster [Ben-Dor et al., 2003; Murali and Kasif, 2003], as illustrated in Figure 2.1a. Madeira and Oliveira [2004] categorize the structures (depicted in Figure 2.1) of biclustering with several biclusters as follows:

(b) Exclusive row and column biclusters: Each row and column belongs to exactly one bicluster. As in traditional clustering, the biclusters are exhaustive and do not overlap. For example, Busygin et al. [2002] and Segal et al. [2001] propose algorithms to discovering bicluster.

(c) Nonoverlapping biclusters with checkerboard structure: The Cartesian product of a clustering of the rows with a clustering of the columns. As a consequence, any cell belongs to one and only one bicluster. For example, Busygin et al. [2002] and Kluger [2003] compute such biclusterings.

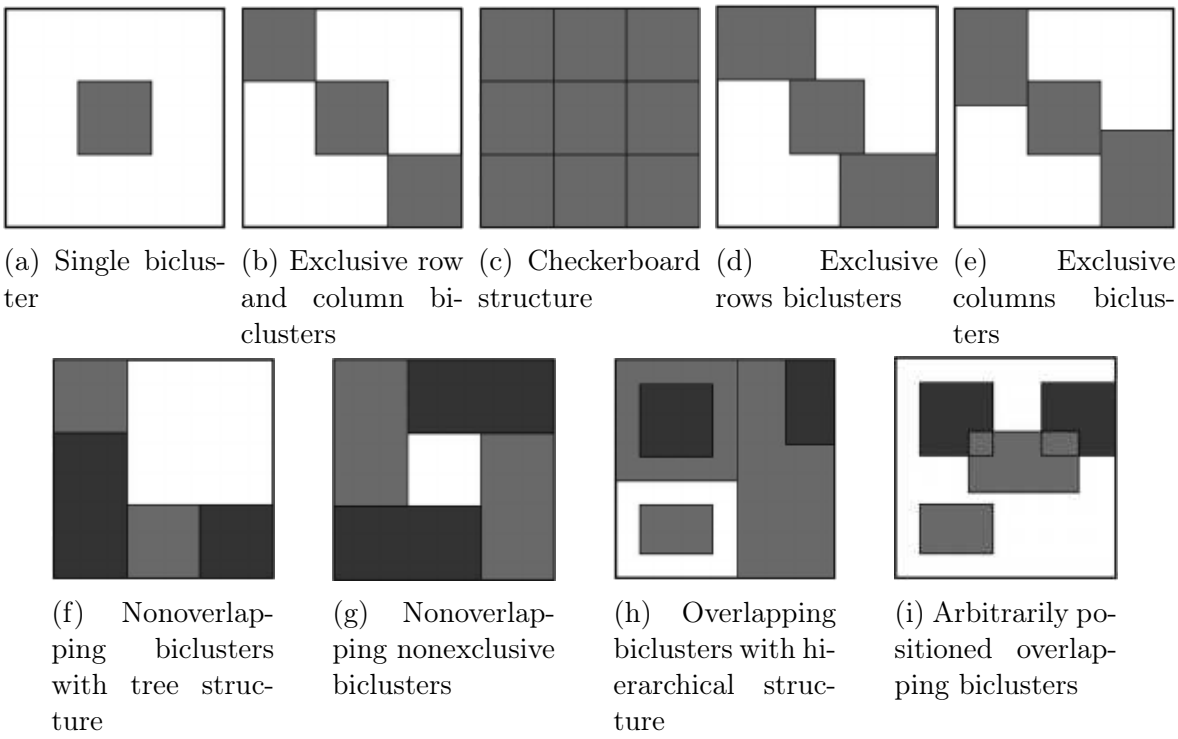
(d) Exclusive-rows biclusters: Each row belongs to at most one bicluster, whereas columns may belong to multiple biclusters. For example, Sheng et al. [2003] and Chun Tang et al. [2001] focus on that structure.

(e) Exclusive-columns biclusters: Each column belongs to at most one bicluster, whereas rows may belong to multiple biclusters. That may be seen as exclusive-rows biclustering of the transposed matrix.

(f) Nonoverlapping biclusters with tree structure: Biclusters are pairs of clusters of rows and clusters of columns. Any two clusters of rows or columns are disjoint, or one includes the other. The first biclustering algorithm proposed by Hartigan [1972] lists a collection of biclusters satisfying that property.

(g) Nonoverlapping nonexclusive biclusters: Rows and columns may belong to more than one bicluster. Unlike the checkerboard structure, the biclusters do not need to be exhaustive. For example, Wang et al. [2002] find nonoverlapping nonexclusive biclusters.

(h) Overlapping biclusters with hierarchical structure: Two biclusters are disjoint or nested. The checkerboard structure and the tree structure are specializations of this structure.



(a) Single bicluster

(b) Exclusive row and column biclusters

(c) Checkerboard structure

(d) Exclusive rows biclusters

(e) Exclusive columns biclusters

(f) Nonoverlapping biclusters with tree structure

(g) Nonoverlapping nonexclusive biclusters

(h) Overlapping biclusters with hierarchical structure

(i) Arbitrarily positioned overlapping biclusters

Figure 2.1: Bicluster structures [Madeira and Oliveira, 2004]

(i) Arbitrarily positioned overlapping biclusters: The most general structure, allowing overlaps and nonexhaustive biclusters. There are numerous contributions that consider that structure [Ben-Dor et al., 2003; Cheng and Church, 2000; Getz et al., 2000; Lazzeroni and Owen, 2002; Liu and Wang, 2003; Murali and Kasif, 2003; Tanay et al., 2002; Yang et al., 2002, 2003].

In structures (b), (c), (d) and (e), biclusters are exhaustive, i. e., every row and every column belong to at least one bicluster. However, that constraint could be removed. In fact, the biclustering structure defined in this thesis is columns-exclusive without the exhaustive constraint. For that reason, Sheng et al. [2003] is related. The author consider the rows-exclusive structure without the constraint, hence the same structure after transposition of the matrix. They use Gibbs sampling to build a probabilistic model of the data. The algorithm finds one bicluster at a time, and masks its rows so that future iterations of the algorithm disconsiders them. Interrelated Two-Way Clustering (ITWC), by Chun Tang et al. [2001], is related too. Similarly to the previously discussed algorithm, it finds one bicluster at a time and masks its rows.

2.3 Bicluster Types

Any bicluster relates to a submatrix. Constraints on such a submatrix, i. e. on every bicluster, depend on the type of problem that is tackled. Madeira and Oliveira [2004] categorizes the types of biclusters that are usually searched in four major classes.

Biclusters with constant values: All values in a submatrix are equal. In real world matrices, noise usually prevents the existence of large such biclusters. That is why quality functions are introduced to score a bicluster. The closer the values in the related submatrix, the higher the score. One such score function is the variance based score function, used by Hartigan [1972].

Biclusters with constant values on rows or columns: All values in a given row (respectively column) of the submatrix are equal. Different rows (respectively columns) may have different values. Formally, a bicluster with constant rows is a submatrix such that any of its values $a_{i,j}$ is $\mu + \alpha_i$ where μ is the typical value of the bicluster and α_i is an adjustment for row i. Alternatively, $a_{i,j}$ can be defined multiplicatively as $\mu \times \alpha_i$. A bicluster with constant columns is analogously defined: $a_{i,j} = \mu + \beta_j$ (or $a_{i,j} = \mu \times \beta_j$). As for bicluster with constant values, scores are computed to allow the discovery of bicluster that are close to satisfying the definition. Nevertheless, this time, computing the variance is not enough. New strategies are necessary, such as first normalizing the data to transform the biclusters into constant biclusters [Getz et al., 2000], or modeling every bicluster in a probabilistic way [Sheng et al., 2003].

Bicluster with coherent values: Biclusters are viewed as either based on an additive or a multiplicative model. In the additive model, each $a_{i,j}$ is $\mu+\alpha_i+\beta_j$, while in the multiplicative one $a_{i,j}$ is $\mu \times \alpha_i \times \beta_j$. In both approaches, μ is the typical value in the bicluster, α_i is an adjustment for row i and β_j is an adjustment for column j. Cheng and Church [2000] introduces a score called *mean squared residue* as a quality function to measure the coherence of rows and columns in this type of biclusters.

Bicluster with coherent evolutions: Unlike the previous types, which refer to the numeric values in the submatrix, biclusters tackle the problem of finding coherent evolutions regardless of the values in the submatrix. For example, the order-preserving submatrix (OPSM) algorithm by Ben-Dor et al. [2003] and OP-Clusters (Order preserving clusters) by Liu and Wang [2003] discover this type of bicluster.

2.3.1 Order-Preserving Submatrices (OPSMs)

Biclusters with coherent evolutions aim at grouping biclusters not based on how similar the values in the related submatrix are, but based in some underlying behavior. One such type of bicluster are order-preserving submatrices. In an order-preserving submatrix, there exists a permutation of the columns of the submatrix that orders the values in any of its rows.

Ben-Dor et al. [2003] first proposed the discovery of order-preserving submatrices. Formally, given a submatrix with a set of T columns and $\pi = (t_1, t_2, \ldots, t_{|T|})$ a linear ordering of T, the pair (T, π) is a complete OPSM model and a row i supports (T, π) if and only if $a_{i,t_1} < a_{i,t_2} < \ldots < a_{i,t_{|T|}}$. The goal of the algorithm is to find the complete models with the supports that are the most statistically significant. They are not necessarily the complete models with maximum support. Indeed, the number of rows that support a model is expected to decrease as its number of columns increases. The decision version of that problem is proven to be NP-Hard, which asks if there is an order preserving submatrix of size k-by-s in the matrix. That is why a greedy search of the biclusters is proposed.

The OPSM model has drawbacks. First, the values in all rows must strictly follow a same ordering. That constraint may be too strong for some applications. Then, only one cluster can be found at a time. The resulting quality of the discovered biclusters is sensitive to the initial selection of the partial models, and also to some given parameters. Furthermore, it favors biclusters with large supports, which can obstruct the discovery of significant biclusters with small supports. Since 2003, many algorithms have been developed to list (approximately) order-preserving submatrices and deal with the presented drawbacks. Cheung et al. [2007], Xue et al. [2014, 2015, 2016, 2018], Veroneze [2016], Liu et al. [2017] and Trapp et al. [2018] have proposed complete algorithms.

2.3.2 Order Preserving Clusters (OP-Clusters)

OP-Clusters, proposed by Liu and Wang [2003], generalizes OPSM. In any row of the input matrix, columns that have similar values, within a user-specified threshold, are grouped and effectively treated as a single element in the row: their relative ordering is irrelevant. The elements in the group are therefore sorted in by their column identifiers, so as to maintain consistency. In this way, a row can more easily support an OP-Cluster than an OPSM. After sorting the rows and grouping similar values, the proposed algorithm, which takes inspiration in sequential pattern mining, treats the rows as sequences of columns where sequential patterns are to be found. Figure 2.2a shows an

example matrix. Its columns are labeled $\{a, b, c, d\}$. Under *seq*, the sorted sequences for each row are shown. Furthermore, elements that are grouped due to similar values are put between "()".

The sequences are stored in a special prefix tree structure called OPC-tree. It is iteratively grown to finally store all the subsequences. As such, the developed OPC-tree contains all potential OP-Clusters. Each node represents one column in the row sequence and each edge connects an element to the next one in the sequence. Figure 2.2b shows the tree as it is initially built (by inserting each entire sequence of the matrix), before an iterative algorithm is applied to grow the tree. The numbers after ":" in the leaves correspond to the supporting rows of each sequence leading to that leaf.

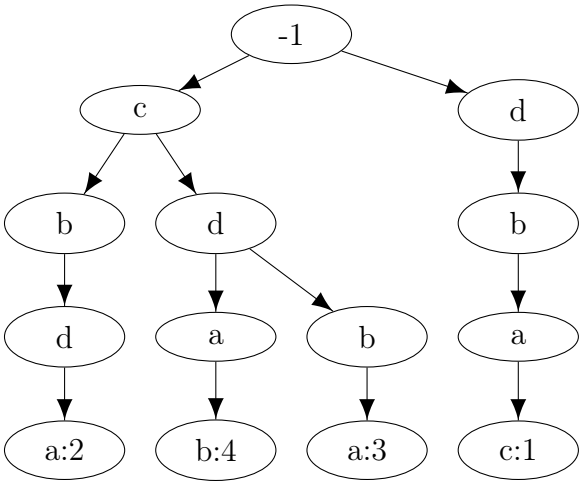
The algorithm contains two user-specified parameters, the minimum number of rows and columns. These parameters allow to prune branches of the tree during its construction, that would only lead to biclusters violating at least one of the minimum sizes. The decision version of OP-Cluster being NP-hard, as it is a generalization of OPSM, enumerating all possible subsequences is usually prohibitive and pruning is essential. After the tree is grown, the OP-Clusters present in it are discovered. The columns in an OP-Cluster are found along the paths leading from the root to any node with depth no less than the minimum number of columns and row support no less than the minimum number of rows.

2.4 Bicluster Algorithms Based on Exhaustive Enumeration

The complexity of the biclustering problem depends on its formulation, but a great part of the existing variants are NP-complete [Madeira and Oliveira, 2004; Cheng and Church, 2000; Ben-Dor et al., 2003; Dhillon, 2001; Dhillon et al., 2003]. One simple formulation of the biclustering problem is, given a binary matrix, finding the maximum edge biclique in the corresponding bipartite graph. The maximum edge biclique problem, where the number of edges is maximum, is equivalent to finding a maximum size bicluster in a matrix, such that the number of rows and columns in the submatrix that is the bicluster is maximum. The decision variant of this problem is proven to be NP-Complete [Peeters, 2003]. As a consequence, many algorithms use heuristics in order to find the proposed biclusters. Nevertheless, some algorithms exhaustively enumerate biclusters but constrain the size of the biclusters in order to make this strategy feasible in practice [Madeira and Oliveira, 2004].

row	a	b	c	d	seq
1	4392	284	4108	228	$db(ac)$
2	401	281	120	298	$c(bd)a$
3	401	292	109	238	$cdba$
4	280	318	37	215	$cdab$

(a) An example matrix



(b) The OPC-Tree during the initial step of the algorithm

Figure 2.2: An OP-Cluster example

The OP-Cluster algorithm by Liu and Wang [2003], presented in Section 2.3.2, exhaustively enumerates biclusters too. Use of user-specified parameters define the minimum number of rows and columns in a valid bicluster. They allow to prune subtrees of the OPC-tree that only contain biclusters that are too small. As previously mentioned, there are also some exhaustive algorithms proposed to mine OPSMs. Cheung et al. [2007] propose an algorithm to find all maximal size-constrained OPSMS, constrained by size, based on the apriori principles [Agrawal and Srikant, 1994]. An OPSM is a maximal OPSM if it is not a proper subcluster of any other OPSM, and it is size-constrained because the number of rows and columns must be larger than two user-defined thresholds. Several rules to prune inadequate patterns are proposed to list all OPSMs efficiently. Xue et al. [2014, 2015] propose approaches based on finding all common subsequences and then applying the apriori principle, Xue et al. [2016] also propose an approach based on the apriori principle, while Xue et al. [2018] propose and approach base on sequential pattern mining. Liu et al. [2017] converts the OPSM mining problem into a frequent sequential pattern mining problem, and then uses dynamic programming to find all common subsquences, storing all common subsequences into a suffix tree. Then, all OPSMs that meet the threshold values of row and column in the

suffix tree are presented. Trapp et al. [2018] propose two algorithms to mine OPSMs, formulating the problem as either a minimization or maximization problem, and then iteratively solving the mathematical programming formulations to global optimality.

The SAMBA (Statistical-Algorithmic Method for Bicluster Analysis) algorithm proposed by Tanay et al. [2002] employs an exhaustive enumeration strategy for gene expression analysis. A bicluster is a submatrix where the values change significantly for a gene (rows) at a condition (columns) with respect to its normal values. That algorithm models the matrix as a bipartite graph, where edges correspond to a significant expression change. They are assigned weights according to a probabilistic model. In this way, heavy subgraphs, i. e., subgraphs with a large sum of weights, correspond to biclusters with a high likelihood. The goal of the algorithm is to find the heaviest subgraphs. In order to avoid an exponential runtime, rows with degree greater than a user-defined parameter are ignored, thus limiting the sizes of the discovered biclusters.

Wang et al. [2002] proposed pClusters, which also exhaustively enumerates biclusters. The algorithm uses a prefix tree to efficiently enumerate all subsets of rows and columns that are valid biclusters. The algorithm aims to find coherent biclusters, in which the score of a bicluster is less or equal than a threshold. The user specifies that threshold and minimum numbers of rows and columns in a bicluster. They allow to avoid the enumeration of invalid clusters, what decreases the runtime of the algorithm. Nevertheless, the worst-case runtime and number of biclusters discovered are exponential in the number of columns.

Serin and Vingron [2011] proposed DeBi, a biclustering algorithm based on Maximal Frequent Itemset mining [Gouda and Zaki, 2001] that identifies coherent biclusters. In that work, a bicluster is a submatrix of genes and samples where the genes are homogeneously highly or lowly expressed all over the samples in the submatrix. Furthermore, each gene in the bicluster shows statistical difference in expression between the samples in the bicluster and the ones not in the bicluster. The data matrix is binarized according to either up or down regulation in the genes, i. e., genes that are fold up regulated in a certain sample are 1, otherwise they are 0. They can also be binarized in the inverse way. The support of a gene set is the fraction of samples for which all genes in the set are 1. A bicluster is the pair consisting of the gene set and the samples that support it. Maximal itemsets, discovered by MAFIA [Burdick et al., 2001] are mined in the matrix after its binarization. In addition to the minimal frequency, a number of columns of the original matrix, a minimal number of rows allows to further prune the pattern space.

2.5 Biclustering Validation

The validation of a biclustering is challenging given the different objectives of the algorithms [Zhao et al., 2012]. The validation of traditional clustering is based on distance measures or on indexes. They aim to measure qualities such as the compactness, the connectedness, the variance or the robustness of the clustering. However, the various types of biclustering make the validation more difficult.

2.5.1 Supervised Validation

When the true biclusters in a dataset are known, the quality of the biclustering can be measured by evaluating how well the found biclusters match the true biclusters. Consequently, an index based validation, such as the one used in traditional clustering, may be employed.

The Jaccard index [Jaccard, 1912], also known as Jaccard similarity coefficient, can be employed when validating traditional clustering. Equation 2.1 defines it for two clusters A and B.

$$JacInd(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2.1}$$

The Jaccard index must be adapted to be used with biclustering. Prelić et al. [2006] modify it. They introduce a match score that Liu and Wang [2007] improved, as shown in Equation 2.2, where B_1 is the set of true biclusters in the matrix and B_2 is the set of detected biclusters. When all biclusters are detected, $S(B_1, B_2) = 1$. Every true bicluster is associated with the most similar bicluster discovered, and then the average score is obtained. This represents how well each of the true biclusters is discovered by the algorithm.

$$S(B_1, B_2) = \frac{1}{|B_1|} \sum_{(X_1,Y_1)\in B_1} \max_{(X_2,Y_2)\in B_2} \frac{|X_1 \cap X_2| + |Y_1 \cap Y_2|}{|X_1 \cup X_2| + |Y_1 \cup Y_2|} \tag{2.2}$$

Another supervised type of strategy for evaluating a biclustering uses domain knowledge, such as prior knowledge about the biological conditions in the biclustering of gene expression data. For example, known classifications of samples or genes support the computation of p-values for validation in Cheng et al. [2008]. The p-value used is the probability of including genes of a given category in a cluster by chance. Thus, the validation can be done by measuring the percentage of clusters of genes that is very unlikely to be obtained randomly. Furthermore, some traditional statistic, such

as sensitivity and specificity, can be used for the comparison of the biclustering results when the true biclusters are known [Gu and Liu, 2008].

2.5.2 Unsupervised Validation

If the true biclusters are not known, there is a need for unsupervised functions to evaluate the quality of a biclustering. One such function is the *mean squared residual error*, proposed by Cheng and Church [2000]. It measures the deviation of the values of the submatrix associated with a bicluster from its mean value. That strategy therefore applies to the validation of constant value biclusters. Teng and Chan [2008] measure the correlation between the rows or the columns of the submatrix to validate a biclustering with coherent values. Ayadi et al. [2009] proposed a quality index based on the Spearman's Rank correlation to measure how similar the order of the rows or the columns. That validation suits biclusters with coherent evolutions.

A different type of strategy consists in validating through statistical tests. A biclustering is compared to a random partitioning of the data matrix. Sheng et al. [2003] randomizes the data according to a uniformly random graph model in order to perform a statistical test. The idea is that biclusters appearing in randomized data are irrelevant.

2.6 Summary

To the best of our knowledge, the type of bicluster proposed in this thesis has not yet been explored in the literature. Nevertheless, the classifications traditional to biclustering can be employed to also classify this new type. The biclustering proposed presents a structure of exclusive-columns. Thus, a column can only be present in a bicluster where the rows with the highest values in that column are also present in the bicluster. Because there can be no biclusters with a superset or subset of rows of another candidate bicluster, a column can be a part of at most one bicluster.

None of the types of biclusters presented in Section 2.3 accurately classify the new type of bicluster proposed. However, the type "Coherent evolutions" presents a similar idea to the new bicluster. OPSMs, which are of that type, exhibit an underlying behavior in which the order of the columns must be the same for all rows in the bicluster. OP-Clusters generalize OPSMs such as to allow the biclusters to have some of its columns not follow a strict ordering. These algorithms are presented in this chapter because, although the matrix in them should first be transposed to better compare them to the new bicluster, their idea is very similar to the one presented in

the new proposed bicluster. But there is still one major difference. The new type of bicluster contains all rows with values in the columns of the bicluster which are strictly greater than the values in rows outside of the bicluster. Thus, the ordering of the rows inside the bicluster is ignored. Meanwhile, in the OPSMs and OP-Clusters only the ordering of the values inside the biclusters are considered.

Section 2.4 explored some algorithms that employ an exhaustive enumeration strategy to discover biclusters. That strategy requires that the search space be limited to make it feasible, usually by constraining the size of the biclusters discovered. The algorithm which will be proposed to discover the new type of bicluster do not present such limitations, as the search space of the proposed bicluster does not need to be reduced.

Finally, Section 2.5 presents some of the strategies used to validate biclusterings, as this is a challenging problem. In particular, the experiments performed in this thesis require that an unsupervised validation strategy be employed, as there are no ground truth to apply a supervised validation.

Chapter 3

A New Type of Bicluster Evaluated Through a Generic Function

Given a finite set R of row identifiers and a finite set C of column identifiers, this work deals with the discovery of relevant biclusters in $\mathbf{M} \in \mathbb{R}^{R \times C}$, a real matrix. All along this work, $m = |R|$ and $n = |C|$, i. e., m and n respectively are the numbers of rows and columns.

Dense and sparse matrices are considered. "Sparse" here means that the number of non-minimal (typically nonzero for nonnegative matrices) values in every column is much smaller than m. The largest such number, $\max_{c \in C} |\{r \in R \mid \mathbf{M}_{r,c} \neq \min_{r' \in R} \mathbf{M}_{r',c}\}|$, is denoted $m_{\text{non-min}}$. Count matrices are common specific sparse matrices. Their values are nonnegative integers. In every column, their distribution typically follows a power law. As a consequence, in any column, there may be many small values that are equal. m_{distinct} denotes the maximal number of distinct values in a column, $\max_{c \in C} |\{\mathbf{M}_{r,c} \mid r \in R\}|$. Clearly, $m_{\text{distinct}} \leq m_{\text{non-min}} + 1 \leq m$.

3.1 Pattern and Support

As usual in the literature, a *bicluster* in $\mathbf{M}$ is a pair of sets: a subset of R and a subset of C. Nevertheless, in this work, given the subset of rows, one single subset of columns can be associated with it to form a valid bicluster. That is why, from now on, the term *pattern* refers to any subset of R. The associated subset of C is called the *support* of the pattern. A column is in the support if and only if the real values in the rows of the pattern are all strictly greater than those in the remaining rows.

Definition 1 (Support)**.** Given a matrix $\mathbf{M} \in \mathbb{R}^{R \times C}$, the support of a pattern $P \subseteq R$, denoted $supp(P)$, is $\{c \in C \mid \forall r \in P, \forall r' \in R \setminus P, \mathbf{M}_{r,c} > \mathbf{M}_{r',c}\}$.

Example 3.1.1. In Figure 1.2, $P = \{r_1, r_3\}$ is a pattern, i. e., a subset of row identifiers. In that matrix $\mathbf{M}$, $\min\{\mathbf{M}_{r_1,c}, \mathbf{M}_{r_3,c}\} > \max\{\mathbf{M}_{r_2,c}, \mathbf{M}_{r_4,c}\}$ if and only if $c \in \{c_1, c_2, c_3\}$, i. e., $supp(P) = \{c_1, c_2, c_3\}$. Indeed, $45 > 35$ (in column c_1), $32 > 20$ (c_2), $4 > 0$ (c_3), but $9 \leq 27$ (c_4) and $6 \leq 29$ (c_5). Figure 1.2a highlights the bicluster $(P, supp(P))$ in the matrix. Figure 1.2b highlights it too, but in a transformation of the matrix. That transformation pairs every value in any row of the matrix with the row identifier and sorts the pairs in every column by decreasing order of value. In this way, highlighting any bicluster $(P, supp(P))$ in the transformed matrix is highlighting its cells that are at the intersection of the $|P|$ first rows and of the columns of $supp(P)$.

The definition of the support entails that intersecting supports necessarily relate to nested patterns. The following lemma formalizes the logically equivalent contrapositive statement.

Lemma 1. $\forall P_1 \subseteq R, \forall P_2 \subseteq R, (P_1 \not\subseteq P_2 \wedge P_2 \not\subseteq P_1) \Rightarrow supp(P_1) \cap supp(P_2) = \emptyset$.

Proof. $P_1 \not\subseteq P_2$ and $P_2 \not\subseteq P_1$ mean that $\exists r_1 \in P_1 \mid r_1 \in R \setminus P_2$ and that $\exists r_2 \in P_2 \mid r_2 \in R \setminus P_1$. Assuming, by contradiction, $supp(P_1) \cap supp(P_2) \neq \emptyset$ is assuming $\exists c \in supp(P_1) \cap supp(P_2)$. By definition of $supp(P_1)$, the inequality $\mathbf{M}_{r_1,c} > \mathbf{M}_{r_2,c}$ would hold. However, by definition of $supp(P_2)$, the logically incompatible inequality $\mathbf{M}_{r_2,c} > \mathbf{M}_{r_1,c}$ would hold as well. □

A corollary of Lemma 1 is that a given column supports at most one pattern of a given size, as formalized and proven below.

Corollary 1. $\forall i \in \mathbb{N}, \forall c \in C, |\{P \subseteq R \mid |P| = i \wedge c \in supp(P)\}| \leq 1$.

Proof. $\forall i \in \mathbb{N}, \forall c \in C$, let $\mathcal{S}_{i,c} = \{P \subseteq R \mid |P| = i \wedge c \in supp(P)\}$, the set whose size is to be bounded. $\forall P_1 \in \mathcal{S}_{i,c}, \forall P_2 \in \mathcal{S}_{i,c}, c \in supp(P_1) \cap supp(P_2)$. Lemma 1 applies: $P_1 \subseteq P_2 \vee P_2 \subseteq P_1$. Moreover $|P_1| = |P_2| = i$. As a consequence, $P_1 = P_2$ and $|\mathcal{S}_{i,c}| \leq 1$. □

In conclusion, a pattern is a subset of rows, and the support of a pattern is the single subset of columns which can be associated to the pattern, such that the values of the rows in the pattern are strictly greater than the values of the rows in the complement of the pattern, for the columns in the support. Additionally, a column can only belong to the support of a single pattern.

3.2 Adequately-sized Patterns

To truly occur in the matrix $\mathbf{M}$, a bicluster should have at least one row and one column. In fact, to split the rows of R into two significant clusters (the rows *in* the pattern and those *out* of it), a pattern should contain at least k and at most $m - k$ rows, for some user-defined $k \in \{1, \ldots, \lfloor \frac{m}{2} \rfloor\}$. The significance of a cluster depends on the application and is therefore determined by the user through the choice of k.

Definition 2 (Adequately-sized pattern)**.** Given a matrix $\mathbf{M} \in \mathbb{R}^{R \times C}$ and an integer $k \in \{1, \ldots, \lfloor \frac{m}{2} \rfloor\}$, a pattern $P \subseteq R$ is adequately-sized if and only if $k \leq |P| \leq m - k$ and $supp(P) \neq \emptyset$.

Example 3.2.1. In Figure 1.2, k must be either 1 or 2, because the matrix has four rows and $\lfloor \frac{4}{2} \rfloor = 2$. The pattern $\{r_1, r_3\}$ having two rows and a nonempty support (see Example 3.1.1), it is adequately-sized with either value of k. Indeed, $1 \leq 2 \leq 4 - 1$ (if $k = 1$) and $2 \leq 2 \leq 4 - 2$ (if $k = 2$).

From now on, $\mathcal{A}$ denotes the set of all adequately-sized patterns. There are at most $(m - 2k + 1)n$ such patterns, as formalized and proven below.

Theorem 2. $|\mathcal{A}| \leq (m - 2k + 1)n$.

Proof. By Definition 2, $P \in \mathcal{A} \Leftrightarrow k \leq |P| \leq m - k \wedge supp(P) \neq \emptyset$. By Definition 1, $supp(P) \neq \emptyset$ means $\exists c \in C \mid c \in supp(P)$. Doing the union over all columns, $\mathcal{A} = \cup_{c \in C} \{P \subseteq R \mid k \leq |P| \leq m - k \wedge c \in supp(P)\}$. Doing the union over all valid sizes, $\mathcal{A} = \cup_{c \in C} \cup_{i=k}^{m-k} \{P \subseteq R \mid |P| = i \wedge c \in supp(P)\}$. As a consequence, $|\mathcal{A}| \leq \sum_{c \in C} \sum_{i=k}^{m-k} |\{P \subseteq R \mid |P| = i \wedge c \in supp(P)\}|$ and, by Corollary 1, $|\mathcal{A}| \leq \sum_{c \in C} \sum_{i=k}^{m-k} 1 = \sum_{c \in C} (m - 2k + 1) = (m - 2k + 1)n$. □

Additionally, the cardinality of the set of adequately-sized pattern, $|\mathcal{A}|$, is upper-bounded by $(m_{\text{distinct}} - 1)n$:

Theorem 3. $|\mathcal{A}| \leq (m_{\text{distinct}} - 1)n$.

Proof. $\mathcal{A} = \cup_{c \in C} \{P \subseteq R \mid k \leq |P| \leq m - k \wedge c \in supp(P)\}$, as shown in the proof of Theorem 2. Since $k \geq 1$, $R \notin \mathcal{A}$ and $\mathcal{A} \subseteq \cup_{c \in C} \{P \subset R \mid c \in supp(P)\}$. As a consequence, $|\mathcal{A}| \leq \sum_{c \in C} |\{P \subset R \mid c \in supp(P)\}|$. By Definition 1, $\{P \subset R \mid c \in supp(P)\} = \{P \subset R \mid \forall r \in P, \forall r' \in R \setminus P, \mathbf{M}_{r,c} > \mathbf{M}_{r',c}\}$, whose cardinality equals $|\{\mathbf{M}_{r,c} \mid r \in R \wedge \mathbf{M}_{r,c} \neq \min_{r' \in R} \mathbf{M}_{r',c}\}| \leq m_{\text{distinct}} - 1$. That is why $|\mathcal{A}| \leq \sum_{c \in C} (m_{\text{distinct}} - 1) = (m_{\text{distinct}} - 1)n$. □

In conclusion, an adequately-sized pattern is a pattern with non-null support and a size defined by $k \leq |P| \leq m - k$. The size of the set of adequately-sized patterns is upper-bounded by both $(m - 2k + 1)n$ and $(m_{\text{distinct}} - 1)n$. If a matrix contains many repeated values, m_{distinct} should be a lot smaller than m, such that $m_{\text{distinct}} - 1) < (m - 2k + 1)$, and thus the size of the set of adequately-size patterns is upper-bounded by $(m_{\text{distinct}} - 1)n$. Otherwise, it is upper-bounded by $(m - 2k + 1)n$.

3.3 Bicluster Quality

This work assumes the existence of a computable function, $q : \mathcal{A} \to \mathbb{R}$. It numerically grades the quality of any adequately-sized pattern. There is no other constraint on the definition of q. It may be tailored to the particularities of the application. It may depend on the matrix $\mathbf{M}$ and/or on other data.

3.3.1 Example of a Natural Quality Function

A natural quality function, given Definition 1, is shown in Equation 3.1, where $\|.\|$ is any norm on $\mathbb{R}^n$.

$$q(P) = \left\| \left(\max \left(\min_{r \in P} \mathbf{M}_{r,c} - \max_{r \in R \setminus P} \mathbf{M}_{r,c}, 0 \right) \right)_{c \in C} \right\| \tag{3.1}$$

This function quantifies to what extent a bicluster stands out in the matrix $\mathbf{M}$. For instance, if $\|.\|$ is the ℓ_1 norm, then $q(P) = \sum_{c \in supp(P)} \min_{r \in P} \mathbf{M}_{r,c} - \max_{r \in R \setminus P} \mathbf{M}_{r,c}$. It sums over the columns of the support the difference between the "worst" row in the pattern (i. e., with the smallest element, in the column) and the "best" row out of the pattern (i. e., with the greatest element, in the column).

Example 3.3.1. Evaluating that quality function on the pattern $P = \{r_1, r_3\}$ in Figure 1.2 is summing the differences $\min\{\mathbf{M}_{r_1,c}, \mathbf{M}_{r_3,c}\} - \max\{\mathbf{M}_{r_2,c}, \mathbf{M}_{r_4,c}\}$ for all $c \in supp(P) = \{c_1, c_2, c_3\}$ (see Example 3.1.1). Those differences are $45 - 35$ (for $c = c_1$), $32 - 20$ (for $c = c_2$) and $4 - 0$ (for $c = c_3$). The pattern P is therefore graded $10 + 12 + 4 = 26$.

3.3.2 Example of an Alternative Quality Function

Another example of a quality function is shown in Equation 3.2, where I_p is the regularized incomplete beta function of parameter $p = \binom{m}{|P|}^{-1}$.

$$q(P) = -I_p(|supp(P)| - 1, n - |supp(P)|) \tag{3.2}$$

Assuming $m_{\text{distinct}} = m$ (for a simpler example), it quantifies how surprising the discovery of an adequately-sized pattern with $|P|$ rows and $|supp(P)|$ columns in its support. It considers, as the null hypothesis, that all $\binom{m}{|P|}$ patterns of size $|P|$ have the same probability, $p = \binom{m}{|P|}^{-1}$, of having a specific column in their supports (any column supports exactly one pattern of size $|P|$, thanks to the simplifying hypothesis) and that the columns are uncorrelated. In this way, the size of the support follows the binomial distribution and $\sum_{\ell=|supp(P)|}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} = I_p(|supp(P)|, n-|supp(P)|+1)$ is the probability that at least $|supp(P)|$ columns support a pattern with $|P|$ rows. q actually ignores one column of the support. That is because any column supports m patterns, of sizes 1, 2, ..., m: q minimally grades them all, -1, unless additional columns support them.

Example 3.3.2. In Figure 1.2, the pattern $P = \{r_1, r_3\}$ having two rows, $p = \binom{4}{2}^{-1} = \frac{1}{6}$. Moreover, $|supp(P)| = 3$ (see Example 3.1.1). As a consequence, P is graded $-I_{\frac{1}{6}}(3-1, 5-3) = -I_{\frac{1}{6}}(2,2) \approx -0.074$.

3.4 Problem Statement

This work proposes to completely list *muscly patterns*. A *muscly pattern* is an adequately-sized pattern with a strictly greater quality than any adequately-sized pattern that is a proper subset or superset of it.

Definition 3 (Muscly pattern)**.** Given $\mathbf{M} \in \mathbb{R}^{R \times C}$, $k \in \{1, \ldots, \lfloor \frac{m}{2} \rfloor\}$ and a function $q : \mathcal{A} \to \mathbb{R}$, a pattern $P \subseteq R$ is a muscly pattern if and only if $P \in \mathcal{A}$ and $\forall P' \in \mathcal{A}$, $(P' \subset P \vee P \subset P') \Rightarrow q(P') < q(P)$.

Example 3.4.1. In Figure 1.2, assuming $k = 1$ and q defined as in Equation 3.1 (with the ℓ_1 norm), the pattern $P = \{r_1, r_3\}$ is not muscly, because it admits a proper and adequately-sized superpattern with a quality greater or equal to $q(P) = 26$ (see Example 3.3.1). Indeed, $P' = \{r_1, r_2, r_3\}$, whose support is $\{c_1\}$, is graded $q(P') = 35 - 5 = 30 \geq 26$. The pattern P' is actually muscly. Its only proper superpattern, $\{r_1, r_2, r_3, r_4\}$, is not adequately-sized and its subpatterns that are both proper and

adequately-sized, $\{r_1\}$, $\{r_2\}$, $\{r_3\}$ and $P = \{r_1, r_3\}$, are graded below 30: $q(\{r_1\}) = 12$, $q(\{r_2\}) = 4$, $q(\{r_3\}) = 10$ and $q(P) = 26$.

From now on, $\mathcal{M}$ denotes the set of all muscly patterns. Their definition entails that two distinct muscly patterns cannot be nested, as formalized below.

Lemma 4. $\forall\{P_1, P_2\} \subseteq \mathcal{M}, P_1 \not\subset P_2 \wedge P_2 \not\subset P_1$.

Proof. Assume, by contradiction, that $\exists\{P_1, P_2\} \subseteq \mathcal{M} \mid P_1 \subset P_2 \vee P_2 \subset P_1$. The inequality $q(P_1) > q(P_2)$ would hold because P_1 is a muscly pattern and $P_2 \in \mathcal{M} \subseteq \mathcal{A}$. However, the logically incompatible inequality $q(P_2) > q(P_1)$ would hold as well because P_2 is a muscly pattern and $P_1 \in \mathcal{M} \subseteq \mathcal{A}$. □

There cannot be more muscly patterns than columns in the matrix.

Theorem 5. $|\mathcal{M}| \leq n$.

Proof. By Definition 2, $\forall P \in \mathcal{M} \subseteq \mathcal{A}$, $supp(P) \neq \emptyset$. Consequently, $\forall P \in \mathcal{M}$, $1 \leq |supp(P)|$. Summing those inequalities: $|\mathcal{M}| \leq \sum_{P\in\mathcal{M}} |supp(P)|$ (1). By Lemmas 4 and 1 (in that order), given two distinct muscly patterns P_1 and P_2, $supp(P_1) \cap supp(P_2) = \emptyset$ and $|supp(P_1)| + |supp(P_2)| = |supp(P_1) \cup supp(P_2)|$. That entails $\sum_{P\in\mathcal{M}} |supp(P)| = |\cup_{P\in\mathcal{M}} supp(P)|$ (2). Any support being a subset of C (Definition 1), $\cup_{P\in\mathcal{M}} supp(P) \subseteq C$ and $|\cup_{P\in\mathcal{M}} supp(P)| \leq n$ (3). Putting together (1), (2) and (3), $|\mathcal{M}| \leq n$. □

In conclusion, we must derive an algorithm to discover the set of muscly patterns in a matrix. This algorithm must receive as additional input a parameter k, and a quality function to evaluate the biclusters.

Chapter 4

Biceps: An Algorithm to Discover Muscly Patterns

In this chapter, Biceps[1], the algorithm proposed to completely list the muscly patterns, is described. Section 4.1 describes the general working of the algorithm, which is divided into three steps. Section 4.2 describes in detail each step of the algorithm. Section 4.3 presents improvements made to the implementation of the algorithm. Finally, Section 4.4 concludes the chapter.

4.1 Filtering the Muscly Patterns in Three Steps

Biceps computes the muscly patterns in three steps, in Algorithm 1. To prove Biceps is sound, this section formally presents the functions they perform.

Algorithm 1 Biceps

Input: real matrix $\mathbf{M} \in \mathbb{R}^{R \times C}$, number of rows $k \in \{1, \ldots, \lfloor \frac{m}{2} \rfloor\}$, function $q : \mathcal{A} \to \mathbb{R}$
Output: set $\mathcal{M}$ of all muscly patterns (Definition 3)

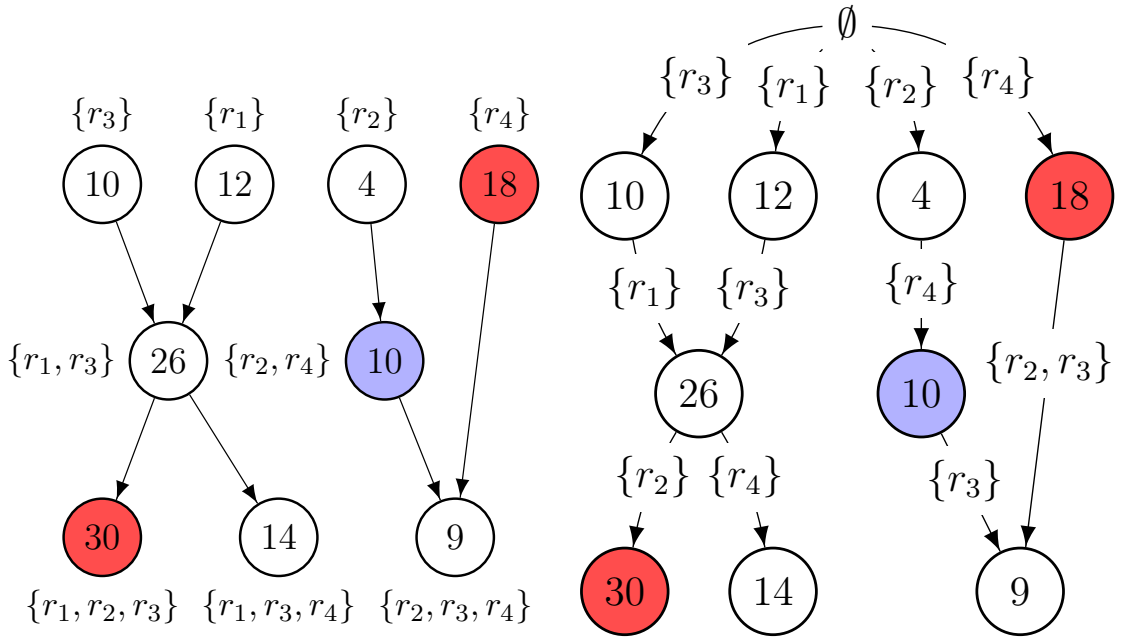
1: $(\mathcal{A}, E) \leftarrow$ BUILD_DAG$(\mathbf{M}, k)$
2: $\mathcal{P} \leftarrow$ FIND_BEST_ON_PATHS$(\mathcal{A}, E, k, q)$
3: FIND_BEST_AMONG_COMPARABLE$(\mathcal{P}, \mathcal{A}, E, k, q)$

BUILD_DAG$(\mathbf{M}, k)$ returns a directed graph. $\mathcal{A}$, the set of all adequately-sized patterns (Definition 2), is the vertex set of the graph. $E \subseteq \mathcal{A} \times \mathcal{A}$ is its edges set. The definition of E will be given later. One single property of it matters to prove Biceps is correct and complete: $\forall (U, V) \in E$, $U \subset V$. Denoting E^+ the transitive

[1] Biceps stands for Biclusters In Columns Exhibiting Prefix-Support, where the prefix refers to the pattern, the top rows in a sorted column of its support.

closure of the edge set E, i. e., the reachability relation, the transitivity of $\subset$ entails that $\forall (U, V) \in E^+$, $U \subset V$. As a consequence, $\forall (U, V) \in E^+$, $U \neq V$, i. e., $(\mathcal{A}, E)$ is a directed acyclic graph (DAG).

Example 4.1.1. Assuming $k = 1$, the vertices of the DAG in Figure 4.1a correspond to all the adequately-sized patterns in the matrix in Figure 1.1. The vertices on a same horizontal level represent patterns having a same number of rows. As explained above, any path in the DAG stands for a proper inclusion between the patterns the two ends of the path represent. The converse statement does not hold. For instance, although $\{r_4\} \subset \{r_2, r_4\}$, no path connects the vertices representing the adequately-sized patterns $\{r_4\}$ and $\{r_2, r_4\}$.



(a) DAG. (b) BUILD_DAG's encoding of the DAG.

Figure 4.1: DAG built from the matrix in Figure 1.1 and $k = 1$, grades (assigned by q in Example 3.3.1, with the ℓ_1 norm) inside the vertices, set $\mathcal{P}$ of candidate patterns (all colored vertices), including all the muscly patterns (red vertices).

FIND_BEST_ON_PATHS$(\mathcal{A}, E, k, q)$ selects every adequately-sized pattern the function q grades strictly better than any of its successors or predecessors:

$$\mathcal{P} = \left\{ P \in \mathcal{A} \mid \forall P' \in \mathcal{A}, \left((P, P') \in E^+ \vee (P', P) \in E^+ \right) \Rightarrow q(P) > q(P') \right\} \quad .$$

Example 4.1.2. In Figure 4.1a, the number inside any vertex is the grade (as defined in Equation 3.1, with the ℓ_1 norm) q gives to the related adequately-sized pattern. The

colored vertices stand for the subset $\mathcal{P}$ of patterns FIND_BEST_ON_PATHS returns, given the shown DAG. For instance, $\{r_2, r_4\} \in \mathcal{P}$. Indeed, the grades 4 and 9 received by its (here single) predecessor ($\{r_2\}$) and its (single as well) successor ($\{r_2, r_3, r_4\}$), respectively, are strictly below $q(\{r_2, r_4\}) = 10$. FIND_BEST_ON_PATHS discards the pattern $\{r_1, r_3\}$, graded 26 (see Example 3.3.1), because $\{r_1, r_2, r_3\}$ succeeds it and is graded $30 \geq 26$ (see Example 3.4.1).

Finally, FIND_BEST_AMONG_COMPARABLE$(\mathcal{P}, \mathcal{A}, E, k, q)$ outputs every pattern of $\mathcal{P}$ that the function q grades strictly better than any of its proper and adequately-sized subsets or supersets:

$$\mathcal{O} = \{P \in \mathcal{P} \mid \forall P' \in \mathcal{A}, (P' \subset P \vee P \subset P') \Rightarrow q(P) > q(P')\} \quad .$$

Example 4.1.3. The two red vertices in Figure 4.1a represent the patterns `Biceps` outputs. FIND_BEST_AMONG_COMPARABLE selects them in $\mathcal{P}$, i. e., among the three patterns corresponding to the colored vertices (see Example 4.1.2). For instance, `Biceps` outputs $\{r_1, r_2, r_3\}$, which is muscly (see Example 3.4.1). Here, FIND_BEST_AMONG_COMPARABLE only discards $\{r_2, r_4\}$, for not being graded better than a proper and adequately-sized (hence in the vertex set) subpattern: $q(\{r_2, r_4\}) \leq q(\{r_4\})$ (see Example 3.4.1).

`Biceps`, as formalized above, is correct and complete.

Theorem 6. $\mathcal{M} = \mathcal{O}$.

Proof. As proven after introducing the directed graph that BUILD_DAG returns, $\forall (U, V) \in E^+$, $U \subset V$. As a consequence, for any adequately-sized pattern $P \in \mathcal{A}$, if $P' \in \mathcal{A}$ satisfies $(P \subset P' \vee P' \subset P) \Rightarrow q(P) > q(P')$ then P' satisfies as well $((P, P') \in E^+ \vee (P', P) \in E^+) \Rightarrow q(P) > q(P')$, i. e., by definitions of $\mathcal{M}$ and $\mathcal{P}$, $\mathcal{M} \subseteq \mathcal{P}$ and, by definition of the output $\mathcal{O}$, $\mathcal{M} = \mathcal{O}$. □

Thus, `Biceps` completely lists the muscly patterns, as long as BUILD_DAG returns an edge set E that is such that $\forall (U, V) \in E$, $U \subset V$. An obvious solution is $E = \emptyset$. Processing a DAG with no edge, FIND_BEST_ON_PATHS would return $\mathcal{A}$ and FIND_BEST_AMONG_COMPARABLE would check whether each and every adequately-size pattern is muscly. Following Definition 3, that last step would pair every adequately-sized pattern with every other and test whether they are nested. For a dense matrix, such a test takes $O(m-k)$ time, if the patterns are ordered. Assuming the evaluations of q on all the adequately-sized patterns are precomputed, the last step would therefore require $O(|\mathcal{A}|^2(m-k))$ time, i. e., by Theorem 2, $O((m-k)^3 n^2)$ time.

	dense $\mathbf{M}$	sparse $\mathbf{M}$
step 1	$O((m-k)^2 n)$	$O\left(m_{\text{non-min}}(\log(m_{\text{non-min}}) + m_{\text{distinct}})n\right)$
step 2	$O(mn)$	$O\left(m_{\text{non-min}} \log(m_{\text{non-min}})n\right)$
step 3	$O\left((m-k)n^2\right)$	$O\left(m_{\text{non-min}} \log(m_{\text{non-min}})n^2\right)$
`Biceps`	$O((m-k)^2 n$ $+(m-k)n^2)$	$O(m_{\text{non-min}} m_{\text{distinct}} n$ $+m_{\text{non-min}} \log(m_{\text{non-min}})n^2)$

Table 4.1: Time complexities.

	dense $\mathbf{M}$	sparse $\mathbf{M}$
step 1	$O((m-k)^2 n)$	$O(m_{\text{non-min}} m_{\text{distinct}} n)$
step 2	$O(mn)$	$O(m_{\text{non-min}} n)$
step 3	$O(mn)$	$O(m_{\text{non-min}} n)$
`Biceps`	$O((m-k)^2 n)$	$O(m_{\text{non-min}} m_{\text{distinct}} n)$

Table 4.2: Space complexities.

For a sparse matrix, testing whether two patterns are nested would take $O(m_{\text{non-min}})$ time and the last step $O(|\mathcal{A}|^2 m_{\text{non-min}})$, i. e., by Theorem 3, $O(m_{\text{non-min}} m_{\text{distinct}}^2 n^2)$.

The next section details `Biceps`' three steps. They comply with their formal specifications in this section. Nevertheless, a different definition of E, dynamic programming and appropriate data structures make the whole algorithm run within $O((m-k)^2 n + (m-k)n^2)$ time, in the worst case for a dense matrix and assuming that, given an adequately-sized pattern and its support, evaluating q takes $O(m+n)$ time. Adaptations for a sparse matrix enable a $O(m_{\text{non-min}} m_{\text{distinct}} n + m_{\text{non-min}} \log(m_{\text{non-min}})n^2)$ time complexity, assuming that evaluating q takes $O(m_{\text{non-min}} + \log(m_{\text{non-min}})n)$ time. Those time requirements are significantly lower than those of the naive algorithm in the previous paragraph, whereas the memory requirements are equal in big O notation: in the worst case, $O((m-k)^2 n)$ for a dense matrix and $O(m_{\text{non-min}} m_{\text{distinct}} n)$ for a sparse matrix. Tables 4.1 and 4.2 show the distributions of the execution time and memory consumption across Algorithm 1's three steps, for both dense and sparse matrices. The next section proves every complexity those tables report.

4.2 Filtering the Muscly Patterns in Subquadratic Time

4.2.1 First Step: The BUILD_DAG Function

4.2.1.1 BUILD_DAG for Dense Matrices

Algorithm 2 details BUILD_DAG. Given the real matrix $\mathbf{M} \in \mathbb{R}^{R \times C}$ and the number of rows $k \in \{1, \dots, \lfloor \frac{m}{2} \rfloor\}$, it returns the graph $(\mathcal{A}, E)$, but materialized differently. Its vertex set, in bijection with $\mathcal{A}$, is partitioned by size of the patterns the vertices represent. Formally, it is $(\mathcal{V}_k, \dots, \mathcal{V}_{m-k})$ where $\forall i \in \{k, \dots, m-k\}$, $\mathcal{V}_i = \{v \text{ representing } V \in \mathcal{A} \mid |V| = i\}$.

The returned functions ds and dp redundantly encode the edge set E. They respectively output the direct successors and the direct predecessors of any vertex u. More precisely, the output $ds(u)$ (respectively, $dp(u)$) is a set of pairs of the type (Δ, v) where v is a direct successor (respectively, direct predecessor) of u and $\Delta \subset R$. The set of rows Δ can be seen as a label on the edge (respectively, the reverse edge) between u and v. That is why, from now on, $\overset{\Delta}{\to} v$ more meaningfully denotes the pair (Δ, v) in a set that ds or dp outputs. BUILD_DAG returns as well $ds(\emptyset)$, although $\emptyset$ is not a vertex. Any $\overset{\Delta}{\to} v$ in $ds(\emptyset)$ satisfies $v \in \cup_{i=k}^{m-k} \mathcal{V}_i$ and $\Delta \subset R$. Finally, BUILD_DAG returns the support $supp(v)$ of the adequately-sized pattern associated with every vertex v.

From a technical point of view, every $\mathcal{V}_i$ is an array. Given the position in $\mathcal{V}_i$ of a vertex u representing a pattern of size i, $\mathcal{V}_i$ provides a constant-time access to $ds(u)$, $dp(u)$, $supp(u)$ and, during `Biceps`' next two steps, to the quality $q(u)$. In fact, v in the notation $\overset{\Delta}{\to} v$ is the position of the vertex in $\mathcal{V}_i$. On the other hand, Δ is a set of rows. More precisely, it is an array of integers identifying rows. $supp(u)$ is an array of integers too, but they identify columns. Lines 1 and 2 in Algorithm 2 initialize those structures.

By Definition 2, $V \subseteq R$ is adequately-sized if and only if $k \leq |V| \leq m - k$ and $supp(V) \neq \emptyset$. As a consequence, enumerating all the patterns having between k and $m - k$ rows and being supported by at least one column is enumerating all the adequately-sized patterns. Line 3 reads, one by one, the columns of the input matrix $\mathbf{M} \in \mathbb{R}^{R \times C}$. To list the adequately-sized patterns the column $c \in C$ supports, line 4 sorts, in $O(m \log m)$ time, the set of rows R in decreasing order of $\mathbf{M}_{r,c}$:

$$\forall c \in C, \forall (r, r') \in R \times R, r \succeq_c r' \Leftrightarrow \mathbf{M}_{r,c} \geq \mathbf{M}_{r',c} \tag{4.1}$$

Example 4.2.1. In any column c of the matrix in Figure 1.2b, the row identifiers from

Algorithm 2 BUILD_DAG

Input: real matrix $\mathbf{M} \in \mathbb{R}^{R \times C}$, number of rows $k \in \{1, \ldots, \lfloor \frac{m}{2} \rfloor\}$
Output: vertices $(\mathcal{V}_k, \ldots, \mathcal{V}_{m-k})$ grouped by number of rows of the patterns they represent: $\forall i \in \{k, \ldots, m-k\}$, $\mathcal{V}_i = \{v \text{ representing } V \in \mathcal{A} \mid |V| = i\}$, direct successors of every $u \in \cup_{i=k}^{m-k} \mathcal{V}_i$: $ds(u) = \{\overset{\Delta}{\Rightarrow} v \mid (U, V) \in E \wedge \Delta = V \setminus U$, where u and v represent U and $V\}$, direct predecessors of every $v \in \cup_{i=k}^{m-k} \mathcal{V}_i$: $dp(v) = \{\overset{\Delta}{\Rightarrow} u \mid \overset{\Delta}{\Rightarrow} v \in ds(u)\}$, "sources" $ds(\emptyset)$, support $supp(v)$ of the pattern represented by every $v \in \cup_{i=k}^{m-k} \mathcal{V}_i$

1: $(\mathcal{V}_k, \ldots, \mathcal{V}_{m-k}) \leftarrow (\emptyset, \ldots, \emptyset)$ ▷ $m_{\text{non-min}}$ instead of $m-k$ if $\mathbf{M}$ is sparse
2: $(ds, dp, supp) \leftarrow (\emptyset, \emptyset, \emptyset)$ ▷ with $O(1)$ access to the head of any edge in ds or dp
3: **for all** $c \in C$ **do**
4: $(r_1, \ldots, r_m) \leftarrow$ sort R w.r.t. $\succeq_c$ in Equation (4.1) ▷ fewer rows if $\mathbf{M}$ is sparse
5: $u \leftarrow \emptyset$
6: $V \leftarrow \emptyset$ ▷ a bitset of size m (if $\mathbf{M}$ is dense) or a sorted array (if $\mathbf{M}$ is sparse)
7: **for** $i = k \rightarrow m - k$ **do**
8: **if** $M_{r_i,c} \neq M_{r_{i+1},c}$ **then**
9: $\Delta \leftarrow \{r_{|V|+1}, \ldots, r_i\}$
10: $V \leftarrow V \cup \Delta$
11: $v \leftarrow$ get or insert vertex in $\mathcal{V}_i$ with key V ▷ through index
12: $ds(u) \leftarrow ds(u) \cup \{\overset{\Delta}{\Rightarrow} v\}$
13: $dp(v) \leftarrow dp(v) \cup \{\overset{\Delta}{\Rightarrow} u\}$
14: $supp(v) \leftarrow supp(v) \cup \{c\}$
15: $u \leftarrow v$
16: **end if**
17: **end for**
18: **end for**
19: **return** $(\mathcal{V}_k, \ldots, \mathcal{V}_{m-k})$, ds, dp, $supp$

top to bottom are sorted w.r.t. $\succeq_c$. To ease the understanding, the figure pairs them with the related values in the column, hence in decreasing order.

By Definition 1, c supports $V \subseteq R$ if and only if $\forall r \in V, \forall r' \in R \setminus V$, $r \succ_c r'$, i. e., the pattern V is a prefix of the sequence of rows as ordered by $\succeq_c$ and, in the column c, the element on the last row in the prefix must be different from the element on the row right after that prefix. Line 8 in Algorithm 2 makes the latter verification for all prefixes of adequate sizes. Line 7 enumerates those sizes, in increasing order. In this way, the bitsets of the rows in the adequately-sized patterns supported by a column can be incrementally built: whenever an adequately-sized pattern supported by $c \in C$ is identified, line 10 only sets the bits corresponding to its rows that were not in the previous such pattern (line 9) or, for the first adequately-sized pattern that c supports, all its rows (given line 6).

Every adequately-sized pattern $V \in \mathcal{A}$ must correspond to one and only one (not

$|supp(V)|$) vertex of $\mathcal{V}_{|V|}$. To discover, in line 11, whether a previously considered column supports V, and create a new vertex (with no direct successor/predecessor and an empty support) otherwise, a bitwise trie indexes the vertices representing the adequately-sized patterns of size $i \in \{k, \ldots, m-k\}$ that were found until now. That data structure enables a worst-case $O(m)$ time complexity to retrieve the vertex identifier of a previously built vertex or, if not found, to insert in the trie the related bitset and associate it with a new vertex identifier. For each of the n columns, line 11 is executed at most $m-2k+1$ times. That maximum is reached if the test in line 8 always passes. Consequently, the overall time spent at line 11 is $O(mn(m-2k+1)) = O((m-k)^2 n)$, in the worst case. Following the same reasoning, all $m-2k+1$ tries take $O((m-k)^2 n)$ space, in the worst case, every inserted bitset being of size m.

Every edge in the DAG that Algorithm 2 builds links two vertices that represent adequately-sized patterns that are consecutively found to be supported by some column. Those patterns contain the rows in larger and larger prefixes of the sorted list obtained in line 4. The property $\forall (U, V) \in E,\ U \subset V$, required for the algorithm to be correct (see Section 4.1), is therefore satisfied and any edge whose head is in $\mathcal{V}_i$ always has its tail in some $\mathcal{V}_j$ with $j < i$:

Lemma 7. $\forall i \in \{k, \ldots, m-k\},\ \forall v \in \mathcal{V}_i$ representing the pattern $V \in \mathcal{A}$, $\{u \in \cup_{j=k}^{m-k} \mathcal{V}_j \mid u \text{ represents } U \in \mathcal{A} \wedge (U, V) \in E\} \subseteq \cup_{j=k}^{i-1} \mathcal{V}_j$.

Proof. $\forall i \in \{k, \ldots, m-k\}$, $\forall v \in \mathcal{V}_i$ representing $V \in \mathcal{A}$, $|V| = i$, by definition of $\mathcal{V}_i$. Moreover, $\forall (U, V) \in E,\ U \subset V$. As a consequence, $|U| < |V| = i$, i. e., the vertex u representing U is in $\cup_{j=k}^{i-1} \mathcal{V}_j$, by definition of $\mathcal{V}_j$. □

In the end, $(U, V) \in \mathcal{A} \times \mathcal{A}$ is in the edge set E if and only if $U \subset V$ and $\exists c \in C \mid \forall u \in U, \forall (v, v') \in (V \setminus U)^2, \forall w \in R \setminus V, \mathbf{M}_{u,c} > \mathbf{M}_{v,c} = \mathbf{M}_{v',c} > \mathbf{M}_{w,c}$. Line 12 adds an edge, from u to v, to $ds(u)$. Line 13 adds the reverse edge, from v to u, to $dp(v)$. A same set of row identifiers labels an edge and the corresponding reverse edge: the difference $V \setminus U$ between the patterns U and V the two vertices u and v represent. The row identifiers in $V \setminus U$ are in a contiguous subsequence (line 9) of the sequence ordered in line 4. After BUILD_DAG starts processing a column, the first vertex v that is retrieved or created becomes directly accessible from $\overset{\Delta}{\to} v$, that line 12 adds to $ds(\emptyset)$. In that situation, given line 6, the label Δ, defined in line 9, identifies every row in the pattern corresponding to the vertex v. More generally, following edges in ds, starting with one in $ds(\emptyset)$, and taking the union of their labels provides the pattern associated with the vertex that is reached.

Example 4.2.2. Given the matrix in Figure 1.1 and $k = 1$, BUILD_DAG returns the graph in Figure 4.1b. In that figure, the ith horizontal level, from top to bottom and starting at the level right below $\emptyset$, contains the vertices in $\mathcal{V}_i$. The reverse edges, in dp, are not explicitly depicted. They are the depicted edges, all those in ds, but reversed. In particular, a same set of row identifiers labels an edge and the corresponding reverse edge: the difference between the nested patterns the two directly connected vertices stand for. To see it, having Figures 4.1a and 4.1b side by side helps. Indeed, Figure 4.1a shows the same graph, with the vertices and the edges in the same positions, but with every vertex labeled with the whole pattern it represents. By looking at the matrix in Figure 1.1 (or at the sorted matrix in Figure 1.2b), the reader may additionally verify that the DAG is the union of five chains. They link the vertices standing for the adequately-sized patterns of increasing size that each of the five columns supports, e. g., $\{r_3\} \rightarrow \{r_1, r_3\} \rightarrow \{r_1, r_2, r_3\}$ for c_1 and $\{r_4\} \rightarrow \{r_2, r_3, r_4\}$ for c_5.

For every column $c \in C$ that is processed, at most $m - k$ row identifiers end up labeling edges in ds: the first ones in the $\succeq_c$ order. As many identifiers label edges in dp. In the end, the returned DAG takes $O((m - k)n)$ space. The edges leaving any vertex $u \in \cup_{i=k}^{m-k} \mathcal{V}_i$ are simply appended to $ds(u)$, without verifying whether they are already present. The same is done for the reverse edges. Checking the presence of an edge in $ds(\emptyset)$ is checking the existence of the vertex at its head. As Line 11 does that verification anyway, duplication of edges in $ds(\emptyset)$ is avoided at no additional cost. In summary, the time BUILD_DAG takes to populate ds and dp is essentially the time to write the edge labels, $O((m - k)n)$, and the edge set is actually an edge multiset.

In the worst case, the supports of all adequately-sized patterns take as much space as the labels of the edges in ds: $O((m - k)n)$. Indeed, whenever the test in line 8 passes, the set $\Delta \subset R$ that labels the edge inserted in ds contains at least one row identifier (line 9), whereas line 14 stores one and only one column identifier. Over BUILD_DAG's whole execution, line 14 populates the supports of all adequately-sized patterns in $O((m - k)n)$ time.

Given the explanations in the previous paragraphs, line 11 dominates BUILD_DAG's time requirements: its time complexity is $O((m - k)^2 n)$, in the worst-case. Also, the tries indexing the vertices dominate the space requirements of BUILD_DAG: its worst-case space complexity is $O((m - k)^2 n)$. Once the execution of BUILD_DAG is over, the tries become useless and are deleted.

4.2.1.2 build_DAG for Sparse Matrices

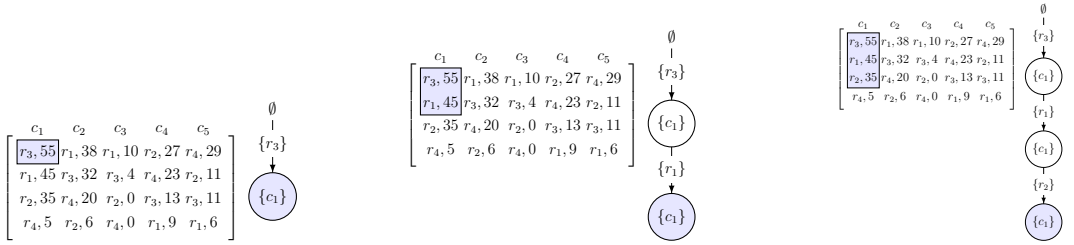
Any column $c \in C$ of a sparse matrix is input as a mapping from a subset $R' \subseteq R$ to $\mathbb{R}$. `Biceps` assumes every value $\mathbf{M}_{r,c}$ with $r \in R \setminus R'$ to equal $\min_{r' \in R'} \mathbf{M}_{r',c}$, the minimal and most common value in the column, typically 0 if $\mathbf{M}$ is nonnegative. In this way, by definition of $m_{\text{non-min}}$, inputting $m_{\text{non-min}} + 1$ pairs of $R \times \mathbb{R}$ is enough to specify any column of $\mathbf{M}$. Each of the n executions of line 4 in Algorithm 2 sorts at most as many rows, in $O(m_{\text{non-min}} \log m_{\text{non-min}})$ time. Given $c \in C$, the definition of $\succeq_c$ (Equation 4.1), used in that line, is extended: $\forall (r, r') \in R \times R$, $r \succ_c r'$ if $\mathbf{M}_{r,c} = \mathbf{M}_{r',c}$ and the integer identifying r is lesser than that of r'. In this way, any edge label, defined in line 9, is ordered by increasing row identifier.

That is important because, for sparse matrices, the pattern V initialized in line 6 is an array (rather than a bitset, for dense matrices) that must be ordered by row identifier: line 11, which verifies whether a previously-considered column supports V, requires such a canonical ordering. Whenever executed, line 10 takes $O(m_{\text{non-min}})$ time to construct the subsequent pattern, by merging V with the edge label while preserving the order. The aforementioned verification has a $O(m_{\text{non-min}})$ amortized cost too. Indeed, for sparse matrices, a hash map (rather than a trie, for dense matrices) indexes the vertices standing for the adequately-sized patterns of size $i \in \{k, \dots, m_{\text{non-min}}\}$ and hashing an array of size $i = O(m_{\text{non-min}})$ takes $O(m_{\text{non-min}})$ time.

The loop in line 7 terminates right before reaching the row that line 4 sorted last[2], i. e., a row mapped to the minimal value in the column. Thanks to the test in line 8, lines 9–10 are executed $O(m_{\text{distinct}} n)$ times, in the worst case. As explained in the previous paragraph, executing once lines 10 and 11 takes $O(m_{\text{non-min}})$ time. Only line 4 requires more, $O(m_{\text{non-min}} \log m_{\text{non-min}})$ time, but it is only executed n times. As a consequence, build_DAG's time complexity is $O(m_{\text{non-min}}(\log(m_{\text{non-min}}) + m_{\text{distinct}})n)$ for sparse matrices.

As for dense matrices, the indices for the vertices dominate the space complexity. For sparse matrices, it is $O(m_{\text{non-min}} m_{\text{distinct}} n)$ in the worst case, where each of the $O(m_{\text{distinct}} n)$ executions of line 11 associates a different array of size $O(m_{\text{non-min}})$ with a new vertex. The indices are deleted once the execution of build_DAG is over. The returned DAG takes $O(m_{\text{non-min}} n)$ space, mostly occupied by the edge labels. By Theorem 3, it has at most $(m_{\text{distinct}} - 1)n$ vertices. $(m_{\text{distinct}} - 1)n$ upper bounds the number of edges in ds (including $ds(\emptyset)$) too. Indeed, only line 12 inserts (one by one)

[2] All along this chapter, the sections about sparse matrices assume the discussed matrix is indeed sparse, i. e., $m_{\text{non-min}} \ll m$. That enables simpler explanations. For instance, the case $m_{\text{non-min}} > m - k$ is not discussed here.

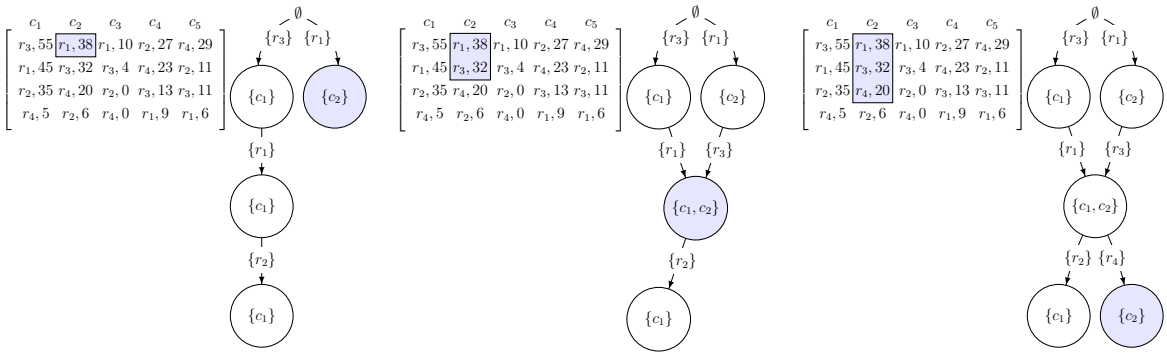


(a) $i = 1$ in line 7: new pattern $\{r_3\}$.

(b) $i = 2$ in line 7: new pattern $\{r_1, r_3\}$.

(c) $i = 3$ in line 7: new pattern $\{r_1, r_2, r_3\}$.

Figure 4.2: Step by step execution of Algorithm 2. Line 3's 1st iteration: enumerating the patterns c_1 supports.



(a) $i = 1$ in line 7: new pattern $\{r_1\}$.

(b) $i = 2$ in line 7: pattern $\{r_1, r_3\}$ retrieved.

(c) $i = 3$ in line 7: new pattern $\{r_1, r_3, r_4\}$.

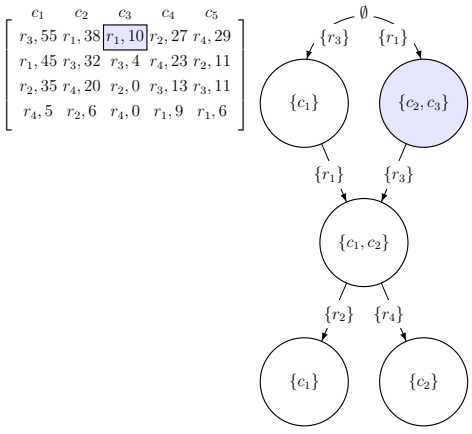
Figure 4.3: Step by step execution of Algorithm 2. Line 3's 2nd iteration: enumerating the patterns c_2 supports.

edges in ds, it is executed as many times as line 11 and, in the scenario maximizing the number of vertices, line 11 always creates a new vertex.

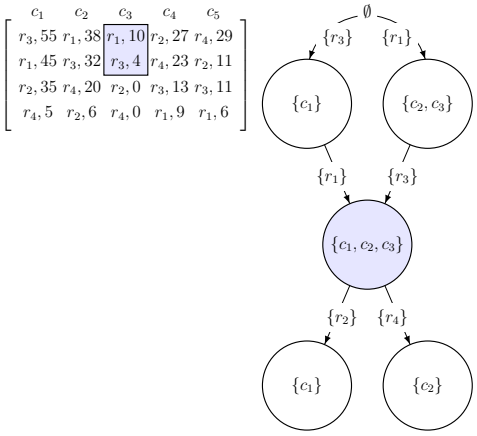
4.2.1.3 BUILD_DAG: Example

Figures 4.2–4.6 illustrate the step-by-step execution of Algorithm 2 on the toy matrix in Figure 1.1. Each Figure represents BUILD_DAG's outer loop in line 3, which enumerates each column in the matrix. Each subfigure represents an iteration of BUILD_DAG's inner loop (starting in line 7), where, for each column, a vertex of the DAG is added or retrieved. The related pattern is highlighted in the matrix, and the vertex itself is highlighted in the DAG.

When a vertex is added, such as in Figure 4.2a, the corresponding support is shown inside it. When an existing vertex is retrieved, such as in Figure 4.3b, the related support is updated: the identifiers of the processed column complements it.

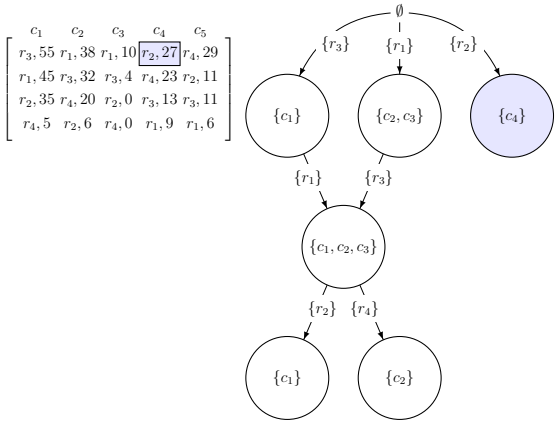


(a) $i = 1$ in line 7: pattern $\{r_1\}$ retrieved.

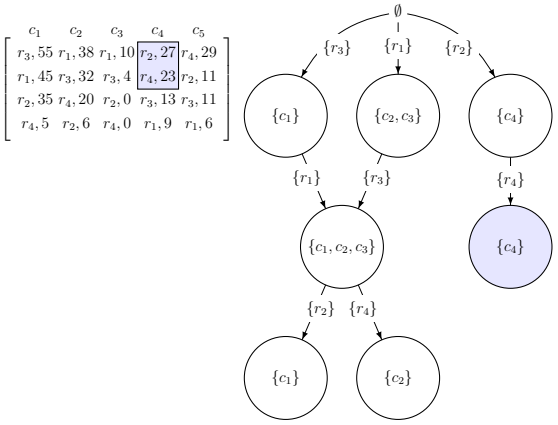


(b) $i = 2$ in line 7: pattern $\{r_1, r_3\}$ retrieved.

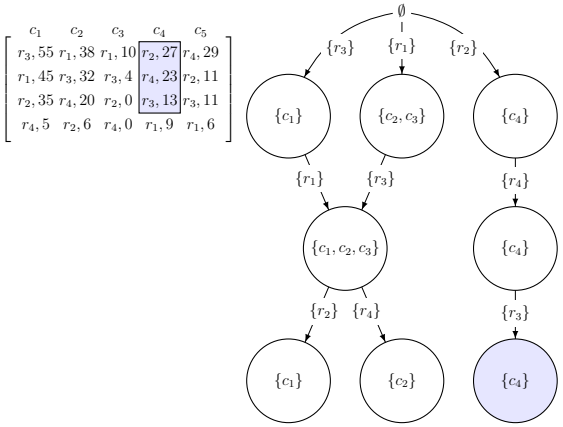
Figure 4.4: Step by step execution of Algorithm 2. Line 3's 3rd iteration: enumerating the patterns c_3 supports.



(a) $i = 1$ in line 7: new pattern $\{r_2\}$.

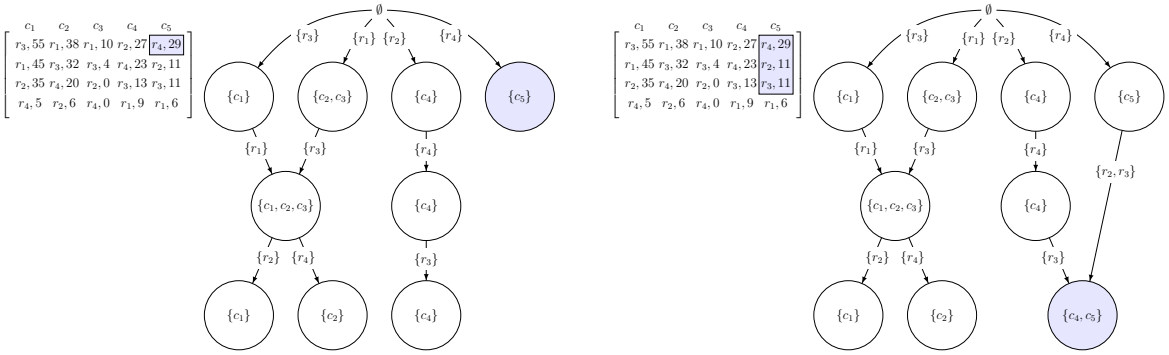


(b) $i = 2$ in line 7: new pattern $\{r_2, r_4\}$.



(c) $i = 3$ in line 7: new pattern $\{r_2, r_3, r_4\}$.

Figure 4.5: Step by step execution of Algorithm 2. Line 3's 4th iteration: enumerating the patterns c_4 supports.



(a) $i = 1$ in line 7: new pattern $\{r_4\}$.

(b) $i = 3$ in line 7: pattern $\{r_2, r_3, r_4\}$ retrieved.

Figure 4.6: Step by step execution of Algorithm 2. Line 3's 5th iteration: enumerating the patterns c_5 supports.

4.2.2 Second Step: The FIND_BEST_ON_PATHS Function

4.2.2.1 FIND_BEST_ON_PATHS for Dense Matrices

Algorithm 3 details FIND_BEST_ON_PATHS. Besides the DAG returned by Algorithm 2 and the user-defined number of rows k, it takes as input the precomputed qualities of all the adequately-sized patterns, which correspond to the vertices of the DAG. Before the execution of Algorithm 3, $q(v)$ is therefore evaluated for each $v \in \cup_{i=k}^{m-k} \mathcal{V}_i$. The support, $supp(v)$, is readily available when $q(v)$ is computed: line 14 in Algorithm 2 filled it. If each evaluation of q takes more than $O(m+n)$ time, the evaluations on all $O((m-k)n)$ adequately-sized patterns (Theorem 2) have an impact of `Biceps`' time complexity in big O notation.

Some quality functions enable an incremental computation of their evaluations along the execution of BUILD_DAG. They can take advantage of line 4's orderings of the rows too. It is the case of the function in Equation 3.1: right after Algorithm 2's line 11, which retrieves or creates a vertex relating to a pattern with $i \in \{k, \ldots, m-k\}$ rows that the column $c \in C$ supports, $\mathbf{M}_{r_i,c} - \mathbf{M}_{r_{i+1},c}$ is added in constant time to an initially-null variable. It becomes, after the enumeration of all the columns in the support, the ℓ_1 norm of the vector Equation 3.1 defines. Since there are at most n columns in the support, evaluating that function q takes $O(n)$ time. The function in Equation 3.2 only depends on the number of rows of the adequately-sized pattern and on the number of columns of its support. Between Algorithms 2 and 3, those two numbers are readily available and evaluating Equation 3.2's function requires $O(m+n)$ time.

In Algorithm 3, lines 1–12 identify every vertex v such that $q(v)$ is strictly greater

Algorithm 3 FIND_BEST_ON_PATHS

Input: DAG returned by Algorithm 2, i. e., $(\mathcal{V}_k, \ldots, \mathcal{V}_{m-k})$, ds and dp, number of rows $k \in \{1, \ldots, \lfloor \frac{m}{2} \rfloor\}$, precomputed $q(v)$ for all $v \in \cup_{i=k}^{m-k} \mathcal{V}_i$

Output: candidates: $\left\{P \in \mathcal{A} \mid \forall P' \in \mathcal{A}, \left((P, P') \in E^+ \vee (P', P) \in E^+\right) \Rightarrow q(P) > q(P')\right\}$, sinks: $\left\{\overset{R \setminus U}{\longrightarrow} u \mid u \text{ represents } U \in \mathcal{A} \wedge ds(u) = \emptyset\right\}$ or $\left\{\overset{U}{\rightarrow} u \mid u \text{ represents } U \in \mathcal{A} \wedge ds(u) = \emptyset\right\}$, for a dense $\mathbf{M}$ or a sparse $\mathbf{M}$, respectively

$\mathcal{P}' \leftarrow \emptyset$ ▷ a bitset of size $\sum_{i=k}^{m-k} |\mathcal{V}_i|$
$q' \leftarrow \{v \mapsto -\infty \mid v \in \cup_{i=k}^{m-k} \mathcal{V}_i\}$ ▷ an array of size $\sum_{i=k}^{m-k} |\mathcal{V}_i|$
for $i = k \rightarrow m - k$ **do**
  for all $v \in \mathcal{V}_i$ **do**
    if $q(v) > q'(v)$ **then**
      $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{v\}$
    end if
    for all $\overset{\Delta}{\Rightarrow} w \in ds(v)$ **do**
      $q'(w) \leftarrow \max(\{q'(w), q'(v), q(v)\})$
    end for
  end for
end for
$q' \leftarrow \{v \mapsto -\infty \mid v \in \cup_{i=k}^{m-k} \mathcal{V}_i\}$
for $i = m - k \rightarrow k$ **do**
  for all $v \in \mathcal{V}_i$ **do**
    if $q(v) \leq q'(v)$ **then**
      $\mathcal{P}' \leftarrow \mathcal{P}' \setminus \{v\}$
    end if
    for all $\overset{\Delta}{\Rightarrow} w \in dp(v)$ **do**
      $q'(w) \leftarrow \max(\{q'(w), q'(v), q(v)\})$
    end for
  end for
end for
$(\mathcal{P}, dp(R)) \leftarrow (\emptyset, \emptyset)$
CANDIDATES_AND_SINKS$(\emptyset, \emptyset, ds, \mathcal{P}', \mathcal{P}, dp(R))$
return $(\mathcal{P}, dp(R))$

Algorithm 4 CANDIDATES_AND_SINKS

Input: current pattern $U \subset R$ (a bitset if $\mathbf{M}$ is dense, an array if $\mathbf{M}$ is sparse), related vertex u, explored vertices $\mathcal{E} \subseteq \cup_{i=k}^{m-k} \mathcal{V}_i$ (a bitset of size $\sum_{i=k}^{m-k} |\mathcal{V}_i|$), ds returned by Algorithm 2, vertices representing the candidate muscly patterns $\mathcal{P}' \subseteq \cup_{i=k}^{m-k} \mathcal{V}_i$ (a bitset of size $\sum_{i=k}^{m-k} |\mathcal{V}_i|$), candidate patterns $\mathcal{P} \subseteq \mathcal{A}$ found until now, sinks $dp(R)$ found until now

```
1: if u ∈ P' then
2:     P ← P ∪ {U}                          ▷ P ← P ∪ {sort(U)}, if M is sparse
3: end if
4: if ds(u) = ∅ then
5:     Δ ← R \ U                            ▷ Δ ← sort(U), if M is sparse
6:     dp(R) ← dp(R) ∪ {Δ→u}
7: end if
8: for all Δ→v ∈ ds(u) do
9:     if v ∉ E then
10:        E ← E ∪ {v}
11:        U ← U ∪ Δ
12:        CANDIDATES_AND_SINKS(U, v, V, ds, P', P, dp(R))
13:        U ← U \ Δ
14:    end if
15: end for
```

than the quality associated with any predecessor of v. Lines 13–23 then keep such a vertex v if and only if $q(v)$ is as well strictly greater than the quality associated with any successor of v. Those two sub-steps are achieved enumerating the edges in ds, then the reverse edges in dp. Dynamic programming makes it possible. Line 2 defines an array q' of size $\sum_{i=k}^{m-k} |\mathcal{V}_i|$. It initially associates every vertex with the value $-\infty$. Lines 3, 4 and 8 enumerate every edge $\overset{\Delta}{\to} w$ leaving some vertex v taken in $\mathcal{V}_k$, then in $\mathcal{V}_{k+1}$, and so on until $\mathcal{V}_{m-k}$, and line 9 updates $q'(w)$ to $\max(q'(w), q'(v), q(v))$. In this way, as formalized below, when any vertex v is enumerated, $q'(v)$ is the greatest quality among the predecessors of v or $-\infty$ if v is a source of the DAG.

Theorem 8. In line 5 of Algorithm 3:

$$q'(v) = \max\{q(u) \in \mathbb{R} \mid u \text{ and } v \text{ represents } U \text{ and } V \wedge (U, V) \in E^+\} \ .$$

Proof. By Lemma 7, $\{(U, V) \in E \mid v \in \mathcal{V}_k \text{ represents } V\} = \emptyset$ (1). As a consequence, for any vertex $v \in \mathcal{V}_k$, line 8 enumerates no edge heading to v and line 9 never redefines $q'(v)$. It is $-\infty$ in line 5, unaltered since the initialization of q' in line 2. (1) also entails that $\{(U, V) \in E^+ \mid v \in \mathcal{V}_k \text{ represents } V\} = \emptyset$ and $\forall v \in \mathcal{V}_k$, $\max\{q(u) \in \mathbb{R} \mid u \text{ and } v \text{ represents } U \text{ and } V \wedge (U, V) \in E^+\}$ is $\max \emptyset = -\infty$.

For any vertex line 4 enumerates at the iteration $i = k$ of the loop in line 3, Theorem 8 therefore holds. Assume the induction hypothesis that for a particular $\ell \in \{k, \ldots, m-k-1\}$, it holds up to $i = \ell$, i. e., $\forall v \in \cup_{i=k}^{\ell} \mathcal{V}_i$, $q'(v) = \max\{q(u) \in \mathbb{R} \mid u \text{ and } v \text{ represent } U \text{ and } V \wedge (U,V) \in E^+\}$. By Lemma 7, $\forall v \in \mathcal{V}_{\ell+1}$, $\{u \in \cup_{j=k}^{m-k} \mathcal{V}_j \mid u \text{ and } v \text{ represent } U \text{ and } V \wedge (U,V) \in E\}$ is included in $\cup_{j=k}^{\ell} \mathcal{V}_j$ (2) and, by the enumeration order, line 8 enumerates all the edges heading to v before the iteration $i = \ell + 1$, where v is considered. When line 8 enumerates the last such edge, line 9 ultimately defines $q'(v)$ as $\max\{\max(q'(u), q(u)) \mid u \text{ and } v \text{ represent } U \text{ and } V \wedge (U,V) \in E\}$. Moreover, (2) means that the vertex u satisfies the induction hypothesis, i. e., $q'(u) = \max\{q(w) \in \mathbb{R} \mid w \text{ and } u \text{ represent } W \text{ and } U \wedge (W,U) \in E^+\}$. Consequently, $q'(v) = \max\{q(u) \in \mathbb{R} \mid u \text{ and } v \text{ represent } U \text{ and } V \wedge (U,V) \in E^+\}$ for any $v \in \mathcal{V}_{\ell+1}$ and, by induction, for any $v \in \cup_{i=k}^{m-k} \mathcal{V}_i$. □

The same rationale applied to the DAG with the edges reversed proves that $q'(v) = \max\{q(u) \in \mathbb{R} \mid v \text{ and } u \text{ represent } V \text{ and } U \wedge (V,U) \in E^+\}$ in line 16. That is why, for each vertex v, lines 5 and 16 respectively test whether the quality associated with v is strictly greater than that of any of its predecessors and whether it is lesser or equal to that of any of its successors. If the first test passes, line 6 adds v to the bitset $\mathcal{P}'$, which is initially empty (line 1). If the second test passes, line 17 removes v from $\mathcal{P}'$. In this way, lines 1–12 compute every vertex that represent an adequately-sized pattern V satisfying $\forall U \in \mathcal{A}, (U,V) \in E^+ \Rightarrow q(V) > q(U)$ and lines 13–23 additionally enforce $\forall U \in \mathcal{A}$, $(V,U) \in E^+ \Rightarrow q(V) > q(U)$. After line 23, $\mathcal{P}'$ therefore is the set of vertices standing for the adequately-sized patterns Section 4.1 promised at the output of FIND_BEST_ON_PATHS.

To get the row identifiers in the patterns the vertices in $\mathcal{P}'$ represent, line 25 calls Algorithm 4 with $\mathcal{P}'$ in argument. That algorithm traverses the DAG depth-first. Lines 9 and 10, together with the fact that all vertices can be reached from some edge in $ds(\emptyset)$, guarantee that every vertex is visited once and only once. An initially-empty bitset of size m materializes the pattern that the reached vertex represents: line 11 sets the bits corresponding to the row identifiers labelling the edge to follow and, back from the recursive call in line 12, line 13 unsets them. Line 2 stores the patterns relating to the vertices in $\mathcal{P}'$ (line 1) in an initially-empty structure (line 24 in Algorithm 3). FIND_BEST_ON_PATHS returns it (line 26).

FIND_BEST_ON_PATHS returns as well the sinks of the DAG. They are denoted $dp(R)$ because they are stored as reverse edges that leave what would be a vertex representing the pattern R. Whenever Algorithm 4's traversal of the DAG reaches a sink (line 4), line 5 lists $R \setminus U$, the row identifiers not in the pattern U the sink

represents. They label the reverse edge line 6 stores. Completed with $dp(R)$, dp allows to follow reverse edges, starting with one in $dp(R)$, while keeping track of the patterns the reached vertices stand for: following any reverse edge $\overset{\Delta}{\to}u$ from the vertex representing the pattern $V \in \mathcal{A}$, the reached vertex u stands for $V \setminus \Delta$.

The patterns in $\mathcal{P}$ have disjoint supports, as formalized and proven below.

Lemma 9. $\forall\{P_1, P_2\} \subseteq \mathcal{P}$, $supp(P_1) \cap supp(P_2) = \emptyset$.

Proof. Assume, by contradiction, that $\mathcal{P}$ contains two distinct patterns P_1 and P_2 such that $supp(P_1) \cap supp(P_2) \neq \emptyset$, i. e., $\exists c \in C \mid c \in supp(P_1) \cap supp(P_2)$. When Algorithm 2's line 3 enumerates c, it would connect the vertices representing P_1 and P_2, i. e., $(P_1, P_2) \in E^+ \vee (P_2, P_1) \in E^+$. Since $P_1 \in \mathcal{P}$, $q(P_1) > q(P_2)$. However, since $P_2 \in \mathcal{P}$, the logically-incompatible inequality $q(P_2) > q(P_1)$ would hold too. □

There are at most n patterns in $\mathcal{P}$.

Theorem 10. $|\mathcal{P}| \leq n$.

Proof. By definitions of $\mathcal{P}$ and $\mathcal{A}$, $\forall P \in \mathcal{P} \subseteq \mathcal{A}$, $supp(P) \neq \emptyset$. As a consequence, $\forall P \in \mathcal{P}$, $1 \leq |supp(P)|$. Summing, $|\mathcal{P}| \leq \sum_{P\in\mathcal{P}} |supp(P)|$ (1). By Lemma 9, $\sum_{P\in\mathcal{P}} |supp(P)| = |\cup_{P\in\mathcal{P}} supp(P)|$ (2). By Definition 1, any support is a subset of C. Consequently, $\cup_{P\in\mathcal{P}} supp(P) \subseteq C$ and $|\cup_{P\in\mathcal{P}} supp(P)| \leq n$ (3). Putting together (1), (2) and (3), $|\mathcal{P}| \leq n$. □

In the worst case, it takes $O((m-k)n)$ time to initialize $\mathcal{P}'$ in line 1 of Algorithm 3 and q' in lines 2 and 13, because there are at most $(m-2k+1)n$ vertices (Theorem 2). For the same reason, comparing the quality associated with every vertex to the highest quality among the predecessors/successors (lines 5 and 16) and updating $\mathcal{P}'$ (lines 6 and 17) take $O((m-k)n)$ time overall. So does propagating one single quality per edge (line 8–10) and reverse edge (line 19–21), because there are $O((m-k)n)$ edges (see Section 4.2.1.1).

In the worst case, Algorithm 4's traversal of the DAG takes $O((m-k)n)$ time. Nevertheless, Algorithm 4 writes $\mathcal{P}$ and $dp(R)$ too. $O(mn)$ time and space are required to write $\mathcal{P}$, because $|\mathcal{P}| \leq n$ (Theorem 10) and m bits represent a pattern in $\mathcal{P}$. Writing $dp(R)$ is equally time-consuming in big O notation, because $|dp(R)| \leq n$ (in each of the n iterations of the loop in line 3 of Algorithm 2, only the last iteration of the inner loop can create a sink) and line 5 reads all m bits representing a sink to list the identifiers of the $O(m-k)$ unset bits. As a consequence, FIND_BEST_ON_PATHS runs in $O(mn)$ time and space, if the matrix is dense.

4.2.2.2 FIND_BEST_ON_PATHS for Sparse Matrices

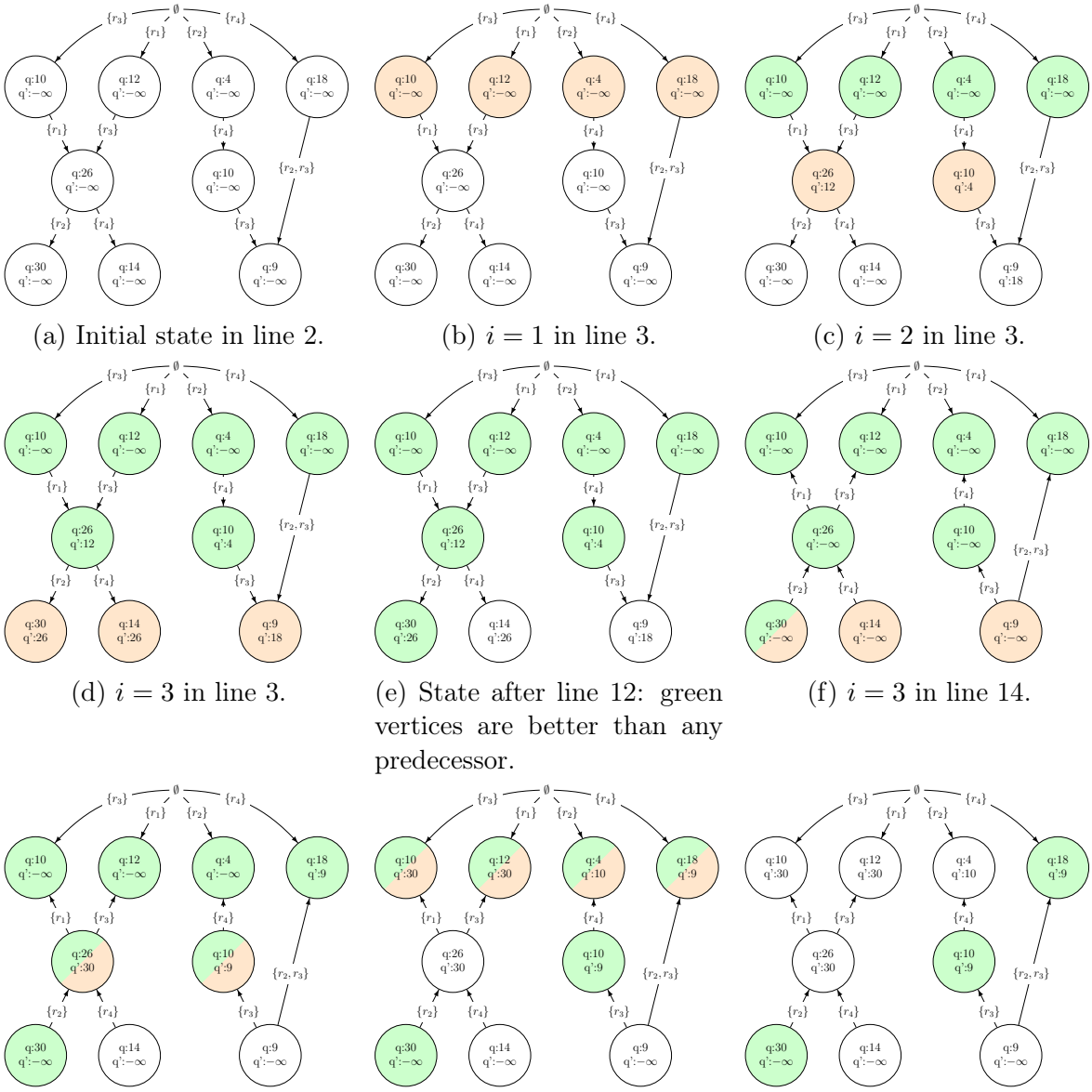
For sparse matrices, an array of row identifiers (rather than a bitset for dense matrices) represents the pattern denoted U in Algorithm 4. As a consequence, any element of $\mathcal{P}$ is such an array, with at most $m_{\text{non-min}}$ row identifiers. Line 5 is modified: it defines the label on the reverse edge inserted in $dp(R)$ as U itself, rather than $R \setminus U$ for dense matrices. In this way, writing $\mathcal{P}$ and $dp(R)$ takes $O(m_{\text{non-min}}n)$ time and space, in the worst case. Nevertheless, every pattern inserted in $\mathcal{P}$ or in $dp(R)$ is sorted right before the insertion. That takes $O(m_{\text{non-min}} \log(m_{\text{non-min}})n)$ time over all those $O(n)$ patterns. In compensation, line 11 needs not produce an ordered array: it simply appends the edge label and line 13 erases that suffix. Those lines therefore take $O(m_{\text{non-min}}n)$ time over Algorithm 4's whole execution.

In the worst case, Algorithm 4's traversal of the DAG requires $O(m_{\text{non-min}}n)$ time, because, edge labels included, the DAG occupies $O(m_{\text{non-min}}n)$ space (see Section 4.2.1.2). In big O notation, that is more than the time to execute all the lines in Algorithm 3 that precede line 25's call to CANDIDATES_AND_SINKS. Indeed, they entirely ignore the edge labels and $(m_{\text{distinct}} - 1)n \leq m_{\text{non-min}}n$ upper bounds both the number of vertices and the number of edges, as the end of Section 4.2.1.2 explains. In the end, for a sparse matrix, FIND_BEST_ON_PATHS requires $O(m_{\text{non-min}} \log(m_{\text{non-min}})n)$ time and $O(m_{\text{non-min}}n)$ space.

4.2.2.3 FIND_BEST_ON_PATHS: Example

Figure 4.7 illustrate the step-by-step execution of Algorithm 3 given the DAG generated for the toy matrix in Figure 1.1 and the quality function in Equation 3.1. The supports associated with each vertex are not reported. They are not used or modified. Instead, inside every vertex v, Figure 4.7 gives $q(v)$ and $q'(v)$. Figure 4.7a depicts the graph as it stands right after line 2. Figures 4.7b–4.7e each correspond to an iteration of the *for* loop in line 3. After testing $q(v) > q'(v)$ for each vertex v in an horizontal level of the DAG (colored in orange), and, if the test passes, adding v to $\mathcal{P}'$ (v is colored in green), the value of q' for each of its successors increase to $max(q'(v), q(v))$ unless it was already higher.

Figures 4.7f–4.7h analogously depict every iteration of Algorithm 3's second loop, which tests the vertices from bottom to top and possibly removes them from $\mathcal{P}'$ (green vertices becoming white). Since q' is then updated for predecessors, those figures show the reverse edges, in dp, rather than the edges, in ds. In the end, the green vertices in Figure 4.7i represent the candidate muscly patterns that Algorithm 4 materializes.



(a) Initial state in line 2.

(b) $i = 1$ in line 3.

(c) $i = 2$ in line 3.

(d) $i = 3$ in line 3.

(e) State after line 12: green vertices are better than any predecessor.

(f) $i = 3$ in line 14.

(g) $i = 2$ in line 14.

(h) $i = 1$ in line 14.

(i) Final state, after line 23: green vertices are better than any predecessor or successor.

Figure 4.7: Step by step execution of Algorithm 3. Every vertex v colored orange has its quality $q(v)$ compared to $q'(v)$. Vertices colored green are in the set $\mathcal{P}'$. They represent candidate muscly patterns.

4.2.3 Third Step: The FIND_BEST_AMONG_COMPARABLE Function

4.2.3.1 FIND_BEST_AMONG_COMPARABLE for Dense Matrices

Algorithm 5 details FIND_BEST_AMONG_COMPARABLE. Every iteration of the loop in line 1 enumerates a candidate pattern $P \in \mathcal{P}$ that Algorithm 3 returned. Line 3 tests whether there exists a proper subpattern that is adequately-sized and that the function q grades at least as well as P. If there is, P is removed from the the set of candidate patterns. Then, every iteration of the loop in line 7 enumerates a candidate pattern $P \in \mathcal{P}$ that was not removed in the previous loop. Line 9 tests whether there exists a proper superpattern that is adequately-sized and that the function q grades at least as well as P. If not, P is muscly and line 10 outputs it. If line 2 in Algorithm 4 is modified so that it stores the vertex u along the pattern P it represents, $supp(u)$ and $q(u)$ can be output too.

Algorithm 5 FIND_BEST_AMONG_COMPARABLE

Input: $\mathcal{P} \subseteq \mathcal{A}$ returned by Algorithm 3, DAG returned by Algorithm 2, *i. e.*, $(\mathcal{V}_k, \ldots, \mathcal{V}_{m-k})$, ds and dp, $dp(R)$ returned by Algorithm 3, precomputed $q(v)$ for all $v \in \cup_{i=1}^{m-k} \mathcal{V}_i$

Output: $\{P \in \mathcal{P} \mid \forall P' \in \mathcal{A}, (P' \subset P \vee P \subset P') \Rightarrow q(P) > q(P')\}$

```
 1: for all P ∈ P do
 2:     E ← ∅                                   ▷ a bitset of size Σ_{i=k}^{|P|-1} |V_i|
 3:     if BETTER_PATTERN?(P, ∅, |P|, E, ds, q) then
 4:         P ← P \ P
 5:     end if
 6: end for
 7: for all P ∈ P do
 8:     E ← ∅                                   ▷ a bitset of size Σ_{i=|P|+1}^{m-k} |V_i|
 9:     if ¬BETTER_PATTERN?(R \ P, R, m − |P|, E, dp, q) then
10:         output P
11:     end if
12: end for
```

Algorithm 6 is responsible for searching either a proper subpattern or a proper superpattern that is adequately-sized and that the function q grades at least as well as the tested pattern. Algorithm 6 follows edges (respectively, reverse edges) in ds (respectively, dp) depth-first, starting from those in $ds(\emptyset)$ (respectively, $dp(R)$). Line 1 enumerates every edge (respectively, reverse edge) leaving the current vertex. Such an edge, $\stackrel{\Delta}{\rightarrow} v$, is followed only if the vertex v has not been visited yet (test $v \notin \mathcal{E}$ in line 2) and if it represents a proper subpattern (respectively, superpattern) of the

Algorithm 6 BETTER_PATTERN?

Input: tested pattern $P \in \mathcal{P}$ (a bitset if $\mathbf{M}$ is dense, a sorted array if $\mathbf{M}$ is sparse) or its complement $R \setminus P$ if $\mathbf{M}$ is dense and $d = dp$, vertex u representing a comparable pattern $P' \in \mathcal{A}$, $\ell = |P \Delta P'|$ (where Δ is the symmetric difference), explored vertices $\mathcal{E}$ (a bitset of size $\sum_{i=k}^{m-k} |\mathcal{V}_i|$), $d \in \{ds, dp\}$ returned by Algorithm 2, precomputed $q(v)$ for all $v \in \cup_{i=k}^{m-k} \mathcal{V}_i$

Output: **true** or **false**

```
1: for all Δ⇒v ∈ d(u) do
2:     if |Δ| < ℓ ∧ v ∉ E then
3:         E ← E ∪ {v}
4:         if Δ ⊆ P ∧ (q(v) ≥ q(P) ∨ BETTER_PATTERN?(P, v, ℓ − |Δ|, E, d, q)) then
5:             return true
6:         end if
7:     end if
8: end for
9: return false
```

tested pattern $P \in \mathcal{P}$. Indeed, line 4 tests whether the edge label is included in P (respectively, whether the reverse edge label is included in $R \setminus P$) after the first condition in line 2 guaranteed $v \in \mathcal{V}_i$ with $i < |P|$ (respectively, $i > |P|$). In particular, the vertex standing for P itself is never reached. Line 5 returns **true** as soon as a vertex representing a pattern with a quality of $q(P)$ or more is reached. If that never happens, line 9 returns **false**.

For each of the $O(n)$ candidate muscly pattern (Theorem 10), lines 2 and 8 in Algorithm 5 take $O((m-k)n)$ time to initialize the bitset $\mathcal{E}$ indicating what vertex is visited, because there are at most $(m-2k+1)n$ vertices (Theorem 2). In the worst case, in each of the $O(n)$ executions of Algorithm 6, line 1 enumerates all $O((m-k)n)$ edges (see Section 4.2.1.1) or all $O((m-k)n)$ reverse edges and line 4 checks the inclusion of all their labels in the tested pattern. Any edge or reverse edge is considered at most once, thanks to $\mathcal{E}$. The labels in either ds or dp totalize $O((m-k)n)$ row identifiers. Checking whether a row identifier is in the tested pattern takes $O(1)$ time, that pattern being stored as a bitset. Overall, FIND_BEST_AMONG_COMPARABLE therefore requires at most $O((m-k)n^2)$ time. Its space complexity equals that of FIND_BEST_ON_PATH, $O(mn)$, for the same reason: the candidate muscly patterns dominate it.

4.2.3.2 FIND_BEST_AMONG_COMPARABLE for Sparse Matrices

Section 4.2.2.2 specifies that, for sparse matrices, the set of row identifiers $\Delta \subset R$ labeling any edge $\overset{\Delta}{\Rightarrow} v$ in $dp(R)$ is the pattern the sink v represents, rather than its complement (for dense matrices). Algorithm 6's search of a proper superpattern of

$P \in \mathcal{P}$ with a quality of $q(P)$ or more (line 9 of Algorithm 5) is given P (instead of $R \setminus P$) as a first argument and is adapted to both changes: when $u = R$, the first conditions in lines 2 and 4 respectively become $|\Delta| > |P|$ and $P \subseteq \Delta$ and the third argument of the recursive call in line 4 is $|\Delta| - |P|$; when $u \neq R$, line 4 tests $\Delta \cap P = \emptyset$ instead of $\Delta \subseteq P$[3]. The search of a proper subpattern of $P \in \mathcal{P}$ with a quality of $q(P)$ or more requires no such adaptations.

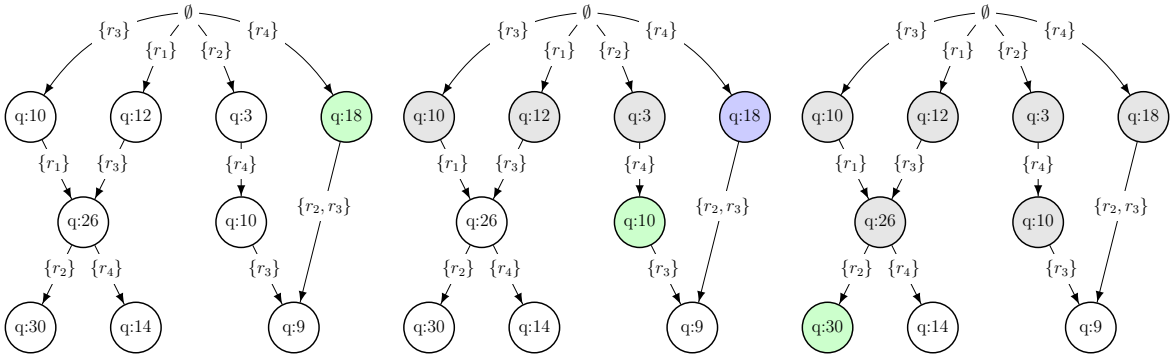
Section 4.2.2.2 specifies as well that any candidate muscly pattern $P \in \mathcal{P}$ is a sorted array containing at most $m_{\text{non-min}}$ row identifiers. Given $\Delta \subset R$ labeling any edge or reverse edge, testing $\Delta \subseteq P$ or $\Delta \cap P = \emptyset$ in line 4 of Algorithm 6 is achieved through binary searches in P of every row identifier of Δ. For a given pattern P, all those searches require $O(m_{\text{non-min}} \log(m_{\text{non-min}})n)$ time, because the edge labels and the reverse edge totalize $O(m_{\text{non-min}}n)$ row identifiers (see Section 4.2.1.2). As the previous paragraph explains, line 4 tests as well $P \subseteq \Delta$, where Δ is a pattern a sink represents. Since P and Δ are sorted arrays, each containing at most $m_{\text{non-min}}$ row identifiers, and since there are at most n sinks, those tests take $O(m_{\text{non-min}}n)$ time for a given pattern P. In the worst case, there are n such candidate patterns (Theorem 10) and the time complexity of FIND_BEST_AMONG_COMPARABLE is $O(m_{\text{non-min}} \log(m_{\text{non-min}})n^2)$. Its space complexity equals that of FIND_BEST_ON_PATH, $O(m_{\text{non-min}}n)$.

4.2.3.3 FIND_BEST_AMONG_COMPARABLE: Example

Figure 4.8 illustrates the step-by-step execution of Algorithm 5 given the DAG generated for the toy matrix in Figure 1.1 and the candidate muscly patterns output by Algorithm 3.

Figures 4.8a–4.8c each correspond to an iteration of the *for* loop in line 1. The green vertex represents the tested pattern. The remaining non-white vertices are visited and those in blue stand for adequately-sized subpatterns. In this example, only one candidate muscly pattern, $\{r_2, r_4\}$, graded 10, is discarded. It happens in Figure 4.8b where $\{r_4\}$ is found to be a better-graded adequately-sized subpattern. Two patterns remain: $\{r_4\}$ and $\{r_1, r_2, r_3\}$. Figures 4.7d–4.7e each correspond to an iteration of the *for* loop in line 7. Reverse edges, in *dp*, are shown. They are followed to search adequately-sized superpatterns that would be at least as good as the tested pattern. In this example, no such superpattern exists. As a consequence, the two patterns are output.

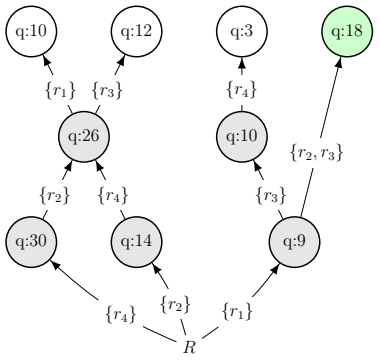
[3]That change is made for dense matrices too. In this way, the first argument of Algorithm 5's second call to BETTER_PATTERN? is P instead of $R \setminus P$. Not flipping the m bits representing each of the $O(n)$ patterns of $\mathcal{P}$ saves $O(mn)$ time. Flipping them avoids what would essentially be a duplication of Algorithm 6 in the text. No complexity in big O notation is altered.



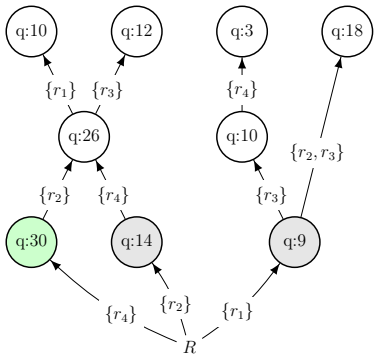
(a) No adequately-sized subpattern of $\{r_4\}$ is at least as good.

(b) $\{r_4\}$ is a better adequately-sized subpattern of $\{r_2, r_4\}$.

(c) No adequately-sized subpattern of $\{r_1, r_2, r_3\}$ is at least as good.



(d) No adequately-sized superpattern of $\{r_4\}$ is at least as good.



(e) No adequately-sized superpattern of $\{r_1, r_2, r_3\}$ is at least as good.

Figure 4.8: Step by step execution of Algorithm 5. In each subfigure, the vertex in green represents the tested candidate pattern, the blue vertices represent either its adequately-sized subpatterns or its adequately-sized superpatterns and gray vertices are additionally marked as visited.

4.3 Improvements

The previous section presented `Biceps` in the simplest way we found that still allows to prove the complexities in Tables 4.1 and 4.2. Nevertheless, the big O notation hides significant improvements that can be (and were) made.

4.3.1 Improvement for Dense and Sparse Matrices

Memory requirements are lowered by getting rid of dp, except for $dp(R)$. The DAG is inverted whenever it must be traversed in the opposite direction. To do so, every vertex $u \in \cup_{i=k}^{m-k} \mathcal{V}_i$ is enumerated, starting with those of $\mathcal{V}_{\min(m-k, m_{\text{non-min}})-1}$, then of $\mathcal{V}_{\min(m-k, m_{\text{non-min}})-2}$ and so on down to $\mathcal{V}_k$. Every edge $\overset{\Delta}{\Rightarrow} v$ in $ds(u)$ defines the related

inverted edge $\overset{\Delta}{\to} u$ that is inserted in $ds(v)$ and $ds(u)$ is cleared. By Lemma 7, $ds(u)$ is cleared before any inverted edge is inserted in it. As a consequence, after the last vertex of $\mathcal{V}_k$ is enumerated, ds associates every vertex with its direct predecessors. To reinvert the DAG, the same procedure is followed but enumerating the vertices of $\mathcal{V}_{k+1}$, then of $\mathcal{V}_{k+2}$ and so on up to $\mathcal{V}_{\min(m-k,m_{\text{non-min}})}$. Every (re)inversion essentially takes the time to write all the $O((m_{\text{non-min}} - k)n)$ edge labels. The first inversion and reinversion occur in Algorithm 3, between lines 12 and 13 and between lines 23 and 25. In Algorithm 5, the DAG is only inverted one more time, between lines 6 and 7.

4.3.2 Improvements for Dense Matrices

`Biceps` remains correct if the test in line 8 of Algorithm 2 only conditions the execution of line 14. That test essentially always passes if the matrix is dense, because its values in any column are usually all distinct. That modification for dense matrices lowers the memory requirements: any edge label becomes one single row identifier rather than an array. Because the arrays are dynamic, $3(m - 2k + 1)n$ pointers are saved, if there are indeed no repeated value in any column. If there are a few repetitions, some vertices may relate to patterns with empty supports and q must have its domain extended with these patterns that it grades $-\infty$. The modification is detrimental if $m_{\text{distinct}} \not\approx m_{\text{non-min}}$, typically for count matrices, which are sparse: the data associated with many patterns with empty supports (the empty arrays for those supports, the additional patterns in the indices, etc.) take more memory than the saved memory.

BUILD_DAG's indexation of the vertices dominates `Biceps`' space complexity. For dense matrices, one bitwise trie per number of rows $i \in \{k, \ldots, m-k\}$ indexes the vertices of $\mathcal{V}_i$. Thanks to an entry table of size $m-i$, the storage of a bitset representing a pattern starts with the smallest identifier of a row in it. The trie therefore consists of one binary tree for each of the $m - i$ possible smallest row identifiers in the patterns. Any edge in those trees is labeled with a bitset: the largest suffix of the largest bitset prefixing all the patterns represented by the leaves below the node at the head of the edge and not prefixing any other pattern represented by the leaves below the node at the tail of the edge. The storage of a bitset representing a pattern ends with the largest identifier of a row in it. This is possible because every represented pattern has the same number of rows, i.

4.3.3 Improvements for Sparse Matrices

Section 4.2.3.2 explains that the time Algorithm 6 requires to find the sinks representing proper superpatterns of a candidate muscly pattern is, in the *worst* case, dominated by that of finding its other proper and adequately-sized superpatterns. In *usual* cases though, where $m_{\text{non-min}} \ll m$, only a few sinks stand for superpatterns of the tested pattern, a small proportion of the DAG is traversed, and figuring out what sinks represent superpatterns takes most of the time. One easy improvement to reduce that time is to not store the sinks standing for minimally-sized patterns, by adding the condition $|U| \neq k$ to Algorithm 4's line 4. Those sinks are indeed useless to the discovery *proper* superpatterns of any $P \in \mathcal{P} \subseteq \mathcal{A}$, because $|P| \geq k$, by Definition 2.

More importantly, an array of size m indexes $dp(R)$. For any $\overset{\Delta}{\to} u \in dp(R)$, where Δ is still an ordered array identifying the rows of the pattern that the sink u represents, its position in $dp(R)$ is stored $|\Delta|$ times in the index, at the positions that are the row identifiers in Δ. In this way, the index is as large as $dp(R)$ itself, hence no increase of the space complexity in big O notation. Given any candidate muscly pattern $P \in \mathcal{P}$, the index allows to access the sinks that represent patterns involving one of the rows of P. Doing so takes linear time in the size of that subset of sinks, usually well below $|dp(R)| = O(n)$. Moreover, by always taking the first row identifier in the sorted array P, $\min(P)$, $\xrightarrow{\Delta \setminus \{\min(\Delta)\}} u$ can (and does) substitute $\overset{\Delta}{\to} u$ in $dp(R)$ and $P \setminus \{\min(P)\} \subseteq \Delta \setminus \{\min(\Delta)\}$ is tested instead of $P \subseteq \Delta$. When executed, the two inclusion tests give the same result. Indeed, either $\min(\Delta) \leq \min(P)$ and they are equivalent, or $\min(\Delta) > \min(P)$ and the test is not executed in the first place, thanks to the index, because $\min(P) \notin \Delta$.

For the call to BETTER_PATTERN? in line 3 of Algorithm 5, an analog index provides a fast access to every $\overset{\Delta}{\to} v \in ds(\emptyset)$ that is such that Δ includes a row of the tested candidate muscly pattern $P \in \mathcal{P}$. For that call, line 4 in Algorithm 6 checks $\Delta \subseteq P$. That is why the index is accessed $|P| = O(m_{\text{non-min}})$ times, to retrieve each time the edges in $ds(\emptyset)$ whose labels include a different row of P. A same edge $\overset{\Delta}{\to} v \in ds(\emptyset)$, among $O(n)$ (in each of the n iterations of the loop in line 3 of Algorithm 2, only the first iteration of the inner loop can add an edge to $ds(\emptyset)$), may therefore be retrieved $O(m_{\text{non-min}})$ times, for any $P \in \mathcal{P}$, among $O(n)$ (Theorem 10). However, line 4 tests $\Delta \subseteq P$ only if line 2 finds, in constant time, that v has not been visited yet. Consequently, in the *worst* case, the indexation of the edges in $ds(\emptyset)$ brings an additional $O(m_{\text{non-min}} n^2)$ term to FIND_BEST_AMONG_COMPARABLE's time complexity, which is unchanged in big O notation. In *usual* cases, where $m_{\text{non-min}} \ll m$, most of the sets of row identifiers labelling the edges in $ds(\emptyset)$ do not intersect with the tested candidate

muscly pattern and the index saves time.

4.4 Summary

This Chapter described a subquadratic algorithm to list muscly patterns in a matrix which is correct and complete. The proposed algorithm is divided into three steps. In the first step, the adequately-sized patterns are listed by creating a DAG where they correspond to the vertices, and the edges represent inclusion relationships between these patterns. In the second step, dynamic programming is used to list every adequately-sized pattern that is strictly better, according to the chosen quality function, than all of its predecessors and successors in the graph. And finally, in the third step, the muscly patterns are filtered from the output of the second step, resulting in a output consisting only of muscly patterns and their respective supports.

Chapter 5

Experiments

`Biceps`' implementation, in C++, is distributed under the terms of the GNU GPLv3. Clang++ 9 compiles it, with the O3 optimizations. All the experiments reported in this section are performed on a GNU/Linux™ system on top of an Intel® Core™ i7-8565U processor running at 4.1 GHz and 20 GB of RAM.

Three sets of experiments were conducted. The first two sets of experiments show that the biclusters discovered by `Biceps` are relevant to the applications in which the algorithm is employed. In both sets of experiments, the biclusters are validated in an unsupervised qualitative manner, as validating a biclustering is a challenge on its own, which is further amplified by the novelty of this type bicluster and the lack of a baseline for it. The first set of experiments deals with Twitter. Bendimerad et al. [2019] collected geolocated and timestamped hashtags and mentions sent from New York City, between 2016/10/08 and 2017/01/07, and from Los Angeles and London, between 2017/05/17 and 2017/07/27. Muscly biclusters are computed in the matrix where the rows stand for the hashtags and user mentions and the columns for geographic subdivisions of the cities associated with 24-hour periods. A cell in the matrix gives how many times the hashtags and user mentions were sent from the geographic subdivision during the 24-hour period. The second set of experiment uses a dataset originating from a Quantitative structure–activity relationship (QSAR) model, which provides the Coulomb and Lennard-Jones potentials for tens of thousands of points (the columns of the matrix) sampled around molecules (the rows). The third and final set of experiments focuses on performance. The runtime and memory consumption of `Biceps` are analyzed using the two datasets, and a third one: a large dense matrix about the expression of genes in different biological samples.

5.1 Mining Sparse Matrices Originating From Twitter

5.1.1 Dataset

Bendimerad et al. [2019] extracted time-stamped geolocated posts from Twitter using its provided API. These posts were extracted for three major cities in different periods. Posts for New York City (NYC) were collected between 2016/10/08 and 2017/01/07, while posts for Los Angeles (LA) and London were collected between 2017/05/17 and 2017/07/27. Tweets from accounts with more than 100 tweets in a 10-day period (probably bots) are ignored. The Twitter API gives access to less than 1% of the posted tweets.

One matrix is created for each city in the dataset: NYC, LA and London. The rows in each matrix are the hashtags and user mentions that were sent at least once in the corresponding city. The columns are the geographical locations (i. e., squares of a grid, as defined by Bendimerad et al. [2019]) during a 24-hour time window, starting at 4 a.m. local time. Thus, the spacial and temporal information becomes an unstructured set of columns and a same location may appear in the support of more than one muscly pattern, as long as the associated windows differ. Any value in the matrix therefore is how many times the term (the row it is in) was sent in the square during a 24-hour period.

In each of the three matrices, the rows with the top-10 sums of values are removed. They stand for generic hashtags, such as: "*#nyc*", "*#brooklyn*" or "*#la*". Keeping them leads to the discovery that those hashtags are indeed much used all over the city, whatever the date. Finally, columns that do not contain any term are removed. They are locations during 24 hours with no tweets. They cannot be in the support of any adequately-sized pattern, even with $k = 1$.

Table 5.1 shows, after preprocessing, the number of terms sent from each city, the number of locations in the grid, as well as the collection period. For each of the three matrices built from those datasets, Table 5.2 gives the number of rows, which is always the number of terms, and the number of columns, which is smaller than the product of the number of locations by the number of days, because some pairs (location, day) relate to no tweet with terms. $m_{\text{non-min}}$, m_{distinct} and the matrix sparsity are reported too.

Unsurprisingly, in each matrix, some terms occur much more than others. Figure 5.1 shows how many times each term was sent, in the whole city over the whole period. The terms are ranked by decreasing count. The curve decreasing faster than

Table 5.1: Description of the data collected for each city

City	nb of Terms	nb of Locations	nb of 24-hour periods
New York City	330,217	1,448	89 (2016/10/08 – 2017/01/05)
Los Angeles	222,494	3,698	71 (2017/05/17 – 2017/07/27)
London	175,477	4,264	71 (2017/05/17 – 2017/07/27)

Table 5.2: Description of the matrix built for each city

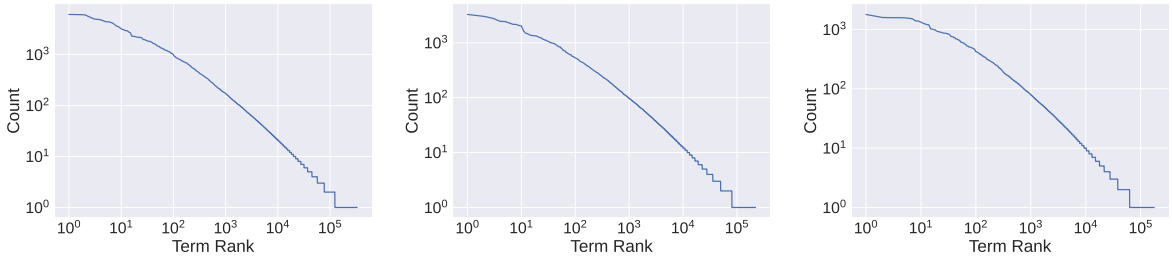
City	m	n	$m_{\text{non-min}}$	m_{distinct}	Sparsity
New York City	330,217	53,009	2,344	41	99.993%
Los Angeles	222,494	65,824	1,860	25	99.995%
London	175,477	55,951	1,532	22	99.994%

linearly in the log-log diagram, the distribution is more skewed than a power law.

Figure 5.2 uses the same type of diagram. It shows the distributions of the number of nonzero values and of distinct values across the columns ordered by decreasing count, hence a different order, for each of the two distributions. $m_{\text{non-min}}$ and m_{distinct} therefore are the ordinates of the leftmost points of the two curves. In the three diagrams, for each of the three matrices, the number of nonzero values in a column suddenly drops shortly before reaching the 100th column with the most nonzero values. The columns, locations associated with 24-hour periods, can therefore be split into two clusters: tens of columns relating to crowded locations where more than a thousand different terms are sent in a day and all the remaining columns where the distribution of the number of different terms is very skewed. Most of the columns in the NYC, LA and London matrices have at most 972, 712 and 382 nonzero values respectively.

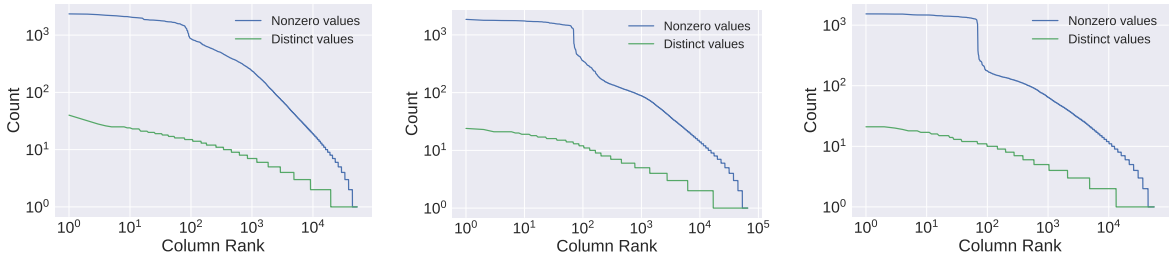
As the beginning of Section 4.2.1.2 explains, sparse matrices are input column by column as pairs (row, value) and the minimal value, 0 in the Twitter matrices, only needs to be input once. As a consequence, the size of `Biceps`' input and the time to read it depend on how many nonzero values there are in the matrix.

The number of vertices in `Biceps`' DAG depends on the number of distinct values per column. So does the size of the indices that dominate `Biceps` sparse requirements. Indeed, every vertex relates to one single entry in one of the indices. Although every matrix has approximately 1,000 columns with more than 100 nonzero values, their number of distinct values is typically more than on order of magnitude smaller and m_{distinct} is two orders of magnitude smaller than $m_{\text{non-min}}$.



(a) New York City (b) Los Angeles (c) London

Figure 5.1: In each city of the Twitter dataset, distribution of the number of occurrences per term.



(a) New York City (b) Los Angeles (c) London

Figure 5.2: In each city of the Twitter dataset, distributions of the number of nonzero values per column (blue curve) and of the number of distinct values per column (green curve).

5.1.2 Results Using the Quality Function in Equation 3.1

This set of experiments uses the natural quality function with the ℓ_1 norm presented in Equation 3.1 to define the muscly biclusters. `Biceps` for sparse matrices is used. It lists sets of terms that, in some locations, are standing out in comparison to other terms sent from the same locations. For each city, a table describes the results for multiple values of k and the muscly biclusters with ten highest grades that are discovered with $k = 2$ are presented and analyzed.

5.1.2.1 Biclusters Discovered in the New York City Matrix

Table 5.3 shows basic statistics of the collections of biclusters `Biceps` returns given the NYC matrix and $k \in \{1, \ldots, 10\}$. The average quality and the average support size of the muscly biclusters for $k = 2$ are considerably smaller than those of the top-10 muscly bicluster, shown in Table 5.4. For that value of k, 88.1% of the muscly patterns are actually minimally-sized, supported by one single column and with the smallest possible quality, 1. Similar observations are made for the the other values of k. They are sets of terms all with nonzero counts in one single column of the matrix.

As a consequence, the k terms relating to nonzero values in a column with exactly k nonzero values are often unique: in any other pair (location, period), at least one of the k terms is never sent.

Table 5.3: Statistics of the collections of muscly biclusters in the NYC matrix with $k \in \{1, \ldots, 10\}$.

k	$\lvert\mathcal{M}\rvert$	$q(P)$			$\lvert P\rvert$			$\lvert supp(P)\rvert$		
		Max	Min	Median	Max	Min	Median	Max	Min	Median
1	9,234	2,229.00	1.00	1.00	23	1	2	235	1	1
2	20,895	564.00	1.00	1.00	163	2	4	100	1	1
3	23,458	169.00	1.00	1.00	163	3	6	38	1	1
4	22,891	118.00	1.00	1.00	163	4	7	38	1	1
5	21,255	118.00	1.00	1.00	163	5	9	30	1	1
6	19,428	76.00	1.00	1.00	164	6	11	30	1	1
7	17,817	52.00	1.00	1.00	164	7	12	19	1	1
8	16,334	52.00	1.00	1.00	164	8	14	19	1	1
9	14,990	52.00	1.00	1.00	179	9	16	19	1	1
10	13,852	18.00	1.00	1.00	179	10	18	8	1	1

Table 5.4 lists the muscly patterns with the 10 highest grades. It also gives the days appearing in the support. Figure 5.3 additionally shows the related locations.

The pattern with the highest grade stand for an event that is precisely located in time and space: the New York Comic-Con in 2016[1]. It took place from October 6th to October 9th. Most of the quality of this bicluster comes from days 8 and 9, when the convention was still happening. The terms in the quality that relate to the following days are small. They show the Comic-Con remained the most commented topic in the supporting locations shortly after it was over. Figure 5.3a pinpoints the location in which the event happened, the Jacob K. Javits Convention Center. A few other locations far from the the convention center appear in the support of the pattern. Nevertheless, they only contribute to 7.3% of the grade.

The second bicluster does not relate to an event. The time window in which it occurs is almost the whole period in which tweets for NYC were collected. The pattern refers to Times Squares, a highly visited place by both tourists and residents alike. Unsurprisingly, the location including Times Square is the only one involved in the support. It also makes sense that, in the support, the location is paired with almost all dates: shows on Broadway, constantly happen.

[1] `https://en.wikipedia.org/wiki/New_York_Comic_Con`

Table 5.4: Muscly biclusters with the ten highest grades in the NYC matrix with $k = 2$.

Pattern id	P	$\lvert supp(P)\rvert$	Grade	Dates appearing in $supp(P)$
1	#nycc2016, #nycc	8	564.00	2016/10/08 - 2016/10/11; 2016/10/14
2	#broadway, #timessquare	44	249.00	Multiple days between 2016/10/08 and 2017/01/03
3	#rockefellercenter, #christmas	22	191.00	Multiple days between 2016/11/28 and 2016/12/27
4	#lotusflame, #thank	3	186.00	2016/11/06; 2016/11/20
5	#happyhalloween, #halloween	31	170.00	2016/10/29 - 2016/10/31
6	#vote, #imwithher, #election2016	26	169.00	2016/11/07 - 2016/11/08
7	#teb, #flightdelay	40	163.00	Multiple days between 2016/10/13 and 2017/01/02
8	#clifton, #accounting	61	153.00	Multiple days between 2016/10/09 and 2017/01/03
9	#endorphins, #endomondo	100	145.00	Multiple days between 2016/10/11 and 2017/01/04
10	#lga, #flightdelay	39	139.00	Multiple days between 2016/10/09 and 2017/01/02

The third and fifth patterns do not stand for specific events either, but for holiday celebrations. The third pattern refers to the Rockefeller Center, whose location is displayed in Figure 5.3c. A large Christmas Tree[2] is exhibited there every month of December. That is why the bicluster spans most of that month. The fifth pattern on the other hand, refers to the Halloween celebration on October, 31st. Its support involves three days, from the 29th to the 31st, and locations all over the city and its surroundings, as displayed in Figure 5.3e. Nevertheless, analyzing the terms summing up to the grade reveals a large concentration of tweets around Lower Manhattan, where the New York Halloween Parade took place[3].

The fourth pattern deals with two events that took place in a restaurant called Lotus Flame, in the Bronx, as shown in Figure 5.3d. The first was the Diwali[4] Motorcade, which occurred on November 5th[5] and contributed a gap of 137 tweets to

[2] `https://tinyurl.com/y3qb5l3v`
[3] `https://tinyurl.com/yyxww86l`
[4] The Hindu festival of lights
[5] `https://tinyurl.com/yc474ozv`

the grade of the pattern. The second one was an event for cancer awareness[6], which contributed a gap of 49 tweets to the grade. A location adjacent to the one containing the restaurant supports this pattern during the 24-hour period of the first event. This suggests the Diwali Motorcade extended further than the location of the restaurant.

The support of the sixth pattern involves several locations. The pattern deals with the US election of 2016 and the New York City support of the Democratic presidential candidate at the time, Hillary Clinton. The 2016 United States elections were held on November 8th[7], and the discovered bicluster spans from November 7th to the actual election day.

The seventh and tenth patterns both refer to airports and flight delays. Flight delays are fairly common, and the patterns span almost the entire data collection period. The locations appearing in the support of the patterns pinpoint the Teterboro Airport in New Jersey in Figure 5.3g and the LaGuardia Airport in Figure 5.3j.

The eighth pattern is associated with one single location, in the city of Clifton, New Jersey, around an accounting firm. The related dates also span almost the entire data collection period. Tweets sent from this specific place therefore refer to that accounting firm more than to any other topic. They suggest that the people in the firm are highly active on Twitter.

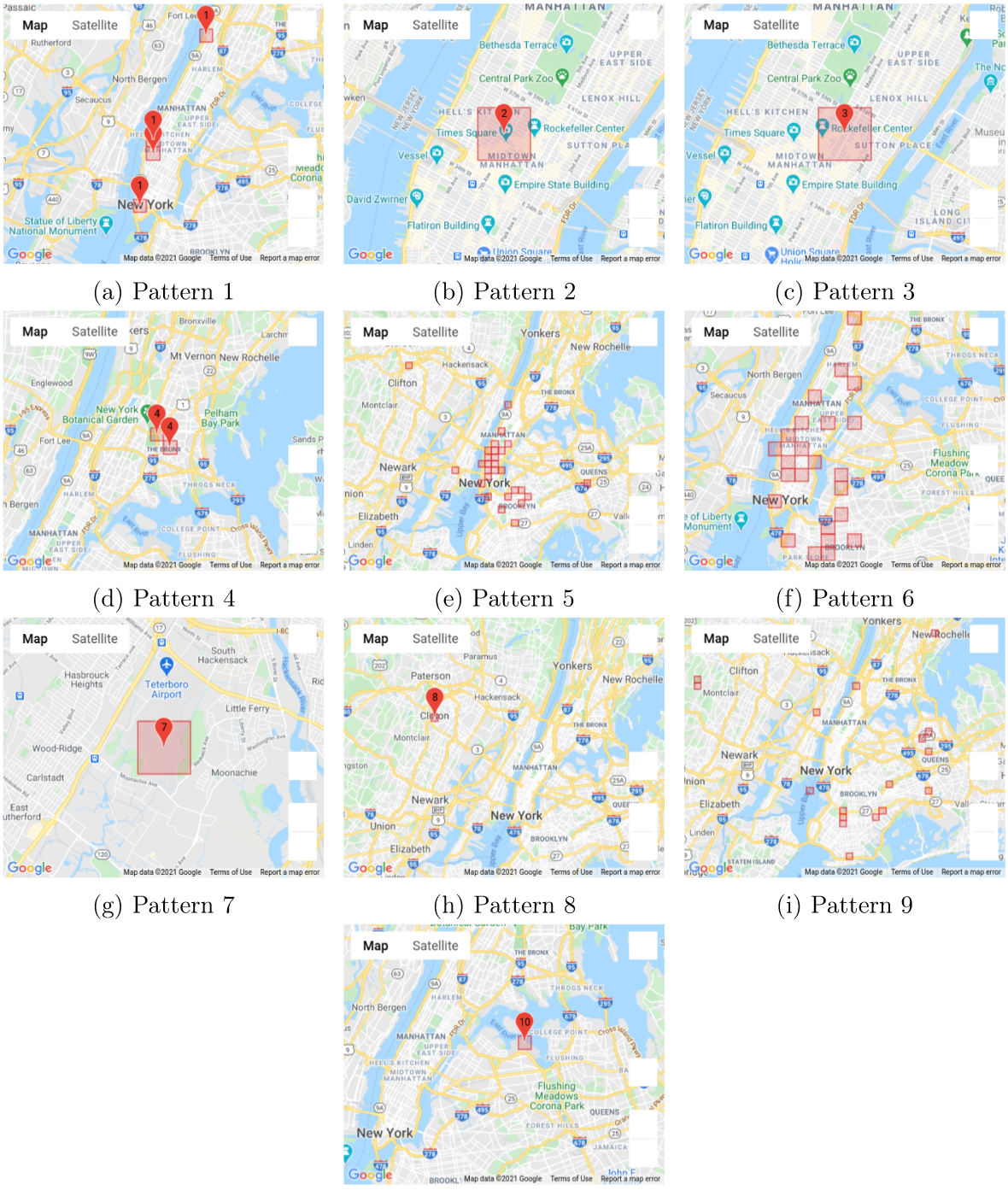
The ninth pattern includes the hashtag *#endomondo*. *Endomondo* is a now retired social fitness app that belonged to the company *Under Armour*[8]. It allowed people to share their workouts on Twitter[9]. The dates appearing in the support span almost the entire tweet collection period. The associated places are where people worked out and posted about it.

5.1.2.2 Biclusters Discovered in the Los Angeles Matrix

Table 5.5 shows the basic statistics of the collections of biclusters `Biceps` returns given the LA matrix and $k \in \{1, \ldots, 10\}$. More biclusters are discovered in LA than in NYC, and a greater proportion receives the grade 1: 92.94% for $k = 2$. Yet, as in NYC, the top-10 muscly biclusters, shown in Table 5.6, are graded significantly higher.

Besides the patterns and their grades, Table 5.6 reports the number of columns in their supports and the smallest time interval containing all dates associated with their columns. Figure 5.4 shows on maps the related locations.

[6] `https://tinyurl.com/ycpg672x`
[7] `https://en.wikipedia.org/wiki/2016_United_States_presidential_election`
[8] `https://tinyurl.com/y4ko7dhf`
[9] `https://www.hardreset.info/devices/apps/apps-endomondo/share-workout-twitter/`



(a) Pattern 1 (b) Pattern 2 (c) Pattern 3

(d) Pattern 4 (e) Pattern 5 (f) Pattern 6

(g) Pattern 7 (h) Pattern 8 (i) Pattern 9

(j) Pattern 10

Figure 5.3: Locations relating to the columns supporting each of the muscly patterns with the highest grades in the NYC matrix with $k = 2$.

Table 5.5: Statistics of the collections of muscly biclusters in the LA matrix with $k \in \{1, \ldots, 10\}$.

k	$\lvert\mathcal{M}\rvert$	$q(P)$			$\lvert P\rvert$			$\lvert supp(P)\rvert$		
		Max	Min	Median	Max	Min	Median	Max	Min	Median
1	13,535	1,871.00	1.00	1.00	20	1	2	245	1	1
2	30,013	454.00	1.00	1.00	82	2	4	104	1	1
3	31,067	207.00	1.00	1.00	117	3	6	40	1	1
4	28,766	87.00	1.00	1.00	129	4	7	40	1	1
5	25,673	87.00	1.00	1.00	129	5	8	40	1	1
6	22,486	87.00	1.00	1.00	166	6	10	18	1	1
7	19,789	87.00	1.00	1.00	166	7	11	18	1	1
8	17,429	87.00	1.00	1.00	166	8	13	14	1	1
9	15,547	87.00	1.00	1.00	166	9	15	14	1	1
10	13,962	10.00	1.00	1.00	214	10	16	5	1	1

The pattern with the highest grades deals with the E3[10] 2017[11], a trade event which took place from June 13th to June 15th. The dates in the support span a period that starts shortly before the expo, and ends almost a week after it was over. Those before and after the event relate to columns that only contribute to 1.32% of the grade. The tweets posted after the event and including the hashtags in the pattern are mostly public reactions to the news presented at the E3. The event took place at the Los Angeles Convention Center. Figure 5.4a shows its location, unsurprisingly.

The second pattern deals with the Anime Expo 2017[12], an anime convention which took place from July 1st to July 4th. The date appearing in the support of the pattern almost perfectly match those of the event. Differently from a trade event such as the E3, it is not common for announcements to be made in a convention. That is probably why the discussion does not go on much after the convention is over. Like the E3, the convention took place in the Los Angeles Convention Center, and as shown in Figure 5.4b, the columns in the support all relate to it.

Two biclusters in the top-10 relate to musical events. The third pattern deals with the FYF Fest[13], a three day music event that took place from July 21st to July 23rd. It was held in the Los Angeles Exposition Park, as Figure 5.4c indicates. The fifth pattern relates to two U2 shows for the *Joshua Tree Tour 2017*[14]. Both shows took place at the Rose Bowl stadium, and, as expected, Figure 5.4e shows the hashtags

[10] Electronic Entertainment Expo
[11] `https://en.wikipedia.org/wiki/E3_2017`
[12] `https://animecons.com/events/info/7489/anime-expo-2017`
[13] `https://www.jambase.com/article/fyf-fest-reveals-2017-lineup`
[14] `https://en.wikipedia.org/wiki/The_Joshua_Tree_Tours_2017_and_2019`

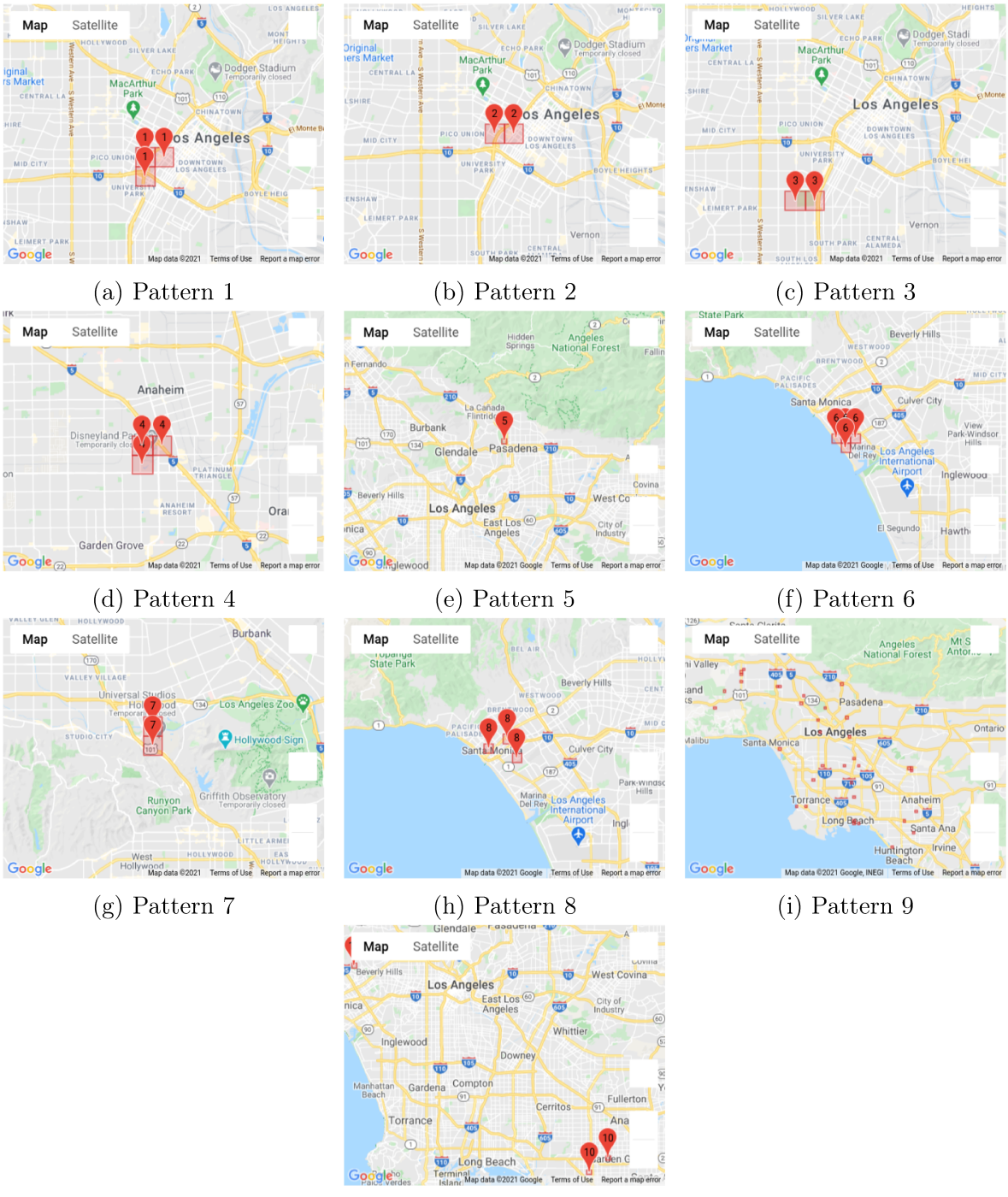
Table 5.6: Muscly biclusters with the ten highest grades in the LA matrix with $k = 2$.

Pattern id	P	$\lvert supp(P)\rvert$	Grade	Dates appearing in $supp(P)$
1	#e3, #e32017	12	454.00	2017/06/10 - 2017/06/17; 2017/06/19; 2017/06/21
2	#ax2017, #animeexpo, #animeexpo2017	9	207.00	2017/06/29 - 2017/07/05
3	#fyf, #fyffest	8	165.00	2017/07/21 - 2017/07/25
4	@disneyland, #disney	50	138.00	Multiple days between 2017/05/17 and 2017/07/26
5	#u2thejoshua-tree2017, #u2	2	124.00	2017/05/20 - 2017/05/21
6	#venice, #venicebeach	56	114.00	Multiple days between 2017/05/17 and 2017/07/26
7	@unistudios, #universalstudios	28	112.00	Multiple days between 2017/05/20 and 2017/07/26
8	#santamonica, #careerarc	40	108.00	Multiple days between 2017/05/17 and 2017/07/05
9	#endorphins, #endomondo	104	107.00	Multiple days between 2017/05/17 and 2017/07/26
10	#fuckfakeshit, #why, #need, #fuckfakes, #gardengrove, #real, #graffiti, #live, #life	14	87.00	Multiple days between 2017/06/02 and 2017/07/26

in the patterns were sent from that stadium on May 20th and 21st.

The support of every remaining pattern in the top-10 involves dates that almost span the whole collected period. Patterns 4, 6, 7 and 8 all are sets of terms referring to specific locations that are unsurprisingly pinpointed in Figures 5.4d, 5.4f, 5.4g and 5.4h respectively. Pattern 8 also indicates that *CarrerArc*, a social recruiting application by the company of the same name[15] is massively used in Santa Monica. The ninth pattern relates to *Endomondo*, the social fitness app whose usage also explained a bicluster in NYC. The tenth pattern contains multiple hashtags, including one referring *#gardengrove*, a city in California, and other terms that are sent from there. But there are locations in the support outside of the city, which contribute to 45.98% of the grade.

[15] `https://www.careerarc.com/about-us`



(a) Pattern 1 (b) Pattern 2 (c) Pattern 3

(d) Pattern 4 (e) Pattern 5 (f) Pattern 6

(g) Pattern 7 (h) Pattern 8 (i) Pattern 9

(j) Pattern 10

Figure 5.4: Locations relating to the columns supporting each of the muscly patterns with the highest grades in the LA matrix with $k = 2$.

5.1.2.3 Biclusters Discovered in the London Matrix

Table 5.7 shows the basic statistics of the collections of biclusters `Biceps` returns given the London matrix and $k \in \{1, \ldots, 10\}$. As for the previous two cities, patterns graded 1 and supported by one single column are predominant. Table 5.8 lists the patterns with the ten highest grades that are discovered for London, their grades and the dates appearing in their supports. Figure 5.5 complements Table 5.8. It shows the locations involved in the supports.

Table 5.7: Statistics of the collections of muscly biclusters in the London matrix with $k \in \{1, \ldots, 10\}$.

k	$\lvert\mathcal{M}\rvert$	$q(P)$			$\lvert P\rvert$			$\lvert supp(P)\rvert$		
		Max	Min	Median	Max	Min	Median	Max	Min	Median
1	12,567	738.00	1.00	1.00	22	1	2	686	1	1
2	26,705	738.00	1.00	1.00	60	2	4	686	1	1
3	27,275	238.00	1.00	1.00	109	3	6	47	1	1
4	24,405	64.00	1.00	1.00	109	4	7	33	1	1
5	21,367	44.00	1.00	1.00	113	5	8	33	1	1
6	18,696	44.00	1.00	1.00	124	6	10	33	1	1
7	16,321	44.00	1.00	1.00	141	7	11	33	1	1
8	14,312	44.00	1.00	1.00	141	8	13	33	1	1
9	12,722	22.00	1.00	1.00	141	9	15	8	1	1
10	11,342	22.00	1.00	1.00	141	10	16	8	1	1

The first pattern is once again refers to the *Endomondo* app, and the columns in its support are widespread in time and space. The second pattern contains terms referring job hiring. Its support doesn't involve as many locations as that of the first pattern, but, as for the first pattern, the related dates are all over the whole data collection period.

Three of the patterns with the top-10 grades deal with musical events: patterns 3, 6 and 8. Pattern 3 refers to the *Wonderland Live*[16], a concert tour by the British band *Take That*. Multiple concerts took place in the O_2 Arena in London between June 6th and June 12th. Besides the O_2 Arena, Figure 5.5c pinpoints another location where extra dates were included in the support. They only account for a some small contribution to the grade: 2.78%. Pattern 6 deals with the *Adele Live 2017* tour[17]. It took place on June 28th and June 29th. A couple more concerts were to happen on July 1st and July 2nd, but they were canceled on June 30th[18]. Adele's performances

[16] `https://en.wikipedia.org/wiki/Wonderland_Live`
[17] `https://en.wikipedia.org/wiki/Adele_Live_2016`
[18] `https://tinyurl.com/y442o2t3`

Table 5.8: Muscly biclusters with the ten highest grades in the London matrix with $k = 2$.

Pattern id	P	$\lvert supp(P)\rvert$	Grade	Dates appearing in $supp(P)$
1	#endorphins, #endomondo	686	738.00	Multiple days between 2017/05/17 and 2017/07/26
2	#hiring, #job	63	399.00	Multiple days between 2017/05/17 and 2017/07/26
3	#wonderland, #takethat	13	218.00	Multiple days between 2017/06/06 and 2017/06/20
4	#jfgt, #grenfelltower	34	125.00	Multiple days between 2017/06/17 and 2017/07/26
5	@wimbledon, #tennis, #wimbledon	15	89.00	Multiple days between 2017/06/26 and 2017/07/15
6	#adele, #wembley	4	81.00	2017/06/28 - 2017/06/30; 2017/07/02
7	#londonpride, #pride	9	58.00	2017/07/08 - 2017/07/09
8	#notinthislifetime, #gunsnroses	6	50.00	2017/06/16 - 2017/06/20
9	#tasteoflondon, @tasteoflondon	6	50.00	2017/06/14 - 2017/06/18; 2017/06/21
10	#kentonmandir, #temple, #sksst, #skststharrow, #swaminarayan, #mandir, #dailypic, #ghanshyam-mahraj,	33	44.00	Multiple days between 2017/05/17 and 2017/07/25

took place at the Wembley Stadium. Unsurprisingly, the single location appearing in the support covers that stadium, as Figure 5.5f indicates. Pattern 8 relates to the *Not in This Lifetime... Tour*[19] from the band Guns N' Roses. Two concerts took place on June 16th and June 17th, in the London Stadium, as Figure 5.5h indicates. A second location north to the stadium, but still inside the Queen Elizabeth Olympic Park, where the stadium is located, adds a contribution of 20.0% to the grade.

The dates in the support of the fourth pattern almost span the whole data collection period, with the exception of May. The pattern refers to the violent fire that

[19] `https://en.wikipedia.org/wiki/Not_in_This_Lifetime..._Tour`

broke out in Grenfell Tower[20], a residence building located as shown in Figure 5.5d. The fire broke out on June 14th, three days before the earliest date appearing in the support. The online emotional response with the hashtag *#jfgt* (Justice for Grenfell Tower), and general discussions went on throughout the whole data collection period for the city.

The fifth pattern refers to the Wimbledon Qualifying 2017[21], which happened from June 26th to June 29th, and to the 2017 Wimbledon Championships[22], from July 3rd to July 16th. Both events took place at the "All England Lawn Tennis and Croquet Club", located in the Wimbledon Park. Figure 5.5e shows an extra location besides the park. It only contributes 1.23% of the grade.

The seventh pattern is a set of hashtags associated with the annual LGBT pride parade in London[23]. It took place on July 8th. The parade started just north of Oxford Circus on Regent Street, and it finished in Whitehall. Its location is pinpointed in Figure 5.5g. An additional location to the north contributes 1.72% of the grade.

The ninth pattern refers to the *Taste of London*[24], an annual restaurant festival that took place in 2017 from June 14th to June 18th. Figure 5.5i the Regent's Park, where the festival took place.

The dates appearing in the support of the tenth pattern are all over the data collection period. The pattern refers to the Shree Kutch Satsang Swaminarayan Temple (Mandir) in London[25]. It is located in the London Borough of Harrow. Every column in the support is associated with that location, as Figure 5.5j indicates. Many events and activities, such as IT classes and Yoga, are organized in the temple all year long.

5.1.3 Results Using a Quality Function Tailored to the Search of Events

The quality function presented in Equation 3.1 and used in the previous experiments doesn't favor patterns whose supports relate to short periods over long periods. In other terms, it is no tailored to the discovery of events. Consequently, biclusters standing for recurring behaviors, such as tweeting about a touristic place or about working out with the *Endomondo* app, receive high grades.

[20] `https://www.bbc.com/news/uk-40301289`
[21] `https://tinyurl.com/yxugxbgr`
[22] `https://en.wikipedia.org/wiki/2017_Wimbledon_Championships`
[23] `https://prideinlondon.org/news-and-views/pride-in-london-2017-date-announced/`
[24] `https://tinyurl.com/y3e5wvhu`
[25] `https://sksst.org/`

(a) Pattern 1 (b) Pattern 2 (c) Pattern 3

(d) Pattern 4 (e) Pattern 5 (f) Pattern 6

(g) Pattern 7 (h) Pattern 8 (i) Pattern 9

(j) Pattern 10

Figure 5.5: Locations relating to the columns supporting each of the muscly patterns with the highest grades in the London matrix with $k = 2$.

In order to specifically discover events, q should take into account the 24-hour periods associated with the columns in the support. Equation 5.1 proposes such a quality function. In its definition, $\|.\|$ is any norm on $\mathbb{R}^n$, $\mathcal{D}$ is the set of days in the data collection period and date is a function that, given a column in C, returns the associated date:

$$q(P) = \max_{d \in \mathcal{D}} \left\| \left(\max \left(\min_{r \in P} \mathbf{M}_{r,c} - \max_{r \in R \setminus P} \mathbf{M}_{r,c}, 0 \right) \right)_{c \in supp(P) \wedge date(c) \in \{d, d+1, d+2\}} \right\| \quad (5.1)$$

This function is a simple adaptation of the quality function presented in Equation 3.1. It grades a pattern using only columns in its support that relate to three consecutive days: the three consecutive days providing the highest grade. In essence, a 3-day sliding window is added, the previous quality function and the highest grade obtained with the supporting columns falling into such a window is returned. In this way, a pattern relating to multiple days receives the same grade as if its support only contains the columns associated with consecutive days.

By using this new quality, it is expected that the patterns that do not deal with events receive lower grades. They may even not be muscly anymore and enable the discovery of subpatterns or superpatterns where complementary terms relate to an event. For example, given the pattern $\{\#broadway, \#timessquare\}$, let's suppose that there is a second pattern $\{\#broadway, \#timessquare, \#lionking\}$ which relates to a hypothetical broadway show that took place during one night only. The quality of the pattern $\{\#broadway, \#timessquare\}$ may decrease so much that it gets lower than the quality of the pattern $\{\#broadway, \#timessquare, \#lionking\}$, thus ceasing to be a muscly pattern, while the latter pattern emerges as a muscly pattern.

5.1.3.1 Biclusters Discovered in the New York City Matrix

Table 5.9 shows basic statistics of the collection of biclusters `Biceps` outputs given the NYC matrix, $k \in \{1, \ldots, 10\}$ and, this time, q as in Equation 5.1. Comparing it to Table 5.3, one can observe that the number of muscly patterns is different. With the new quality function, patterns disappear from the result set, while others took their places.

Table 5.10 lists the 10 patterns that are given the highest grades by the new quality function. Five new patterns emerge, while patterns 2, 7, 8, 9 and 10 in Table 5.4 disappear. Figure 5.6 shows the locations appearing in the support of every new pattern.

Table 5.9: Statistics of the collections of muscly biclusters in the NYC matrix with $k \in \{1, \ldots, 10\}$ and q as in Equation 5.1.

k	$\lvert\mathcal{M}\rvert$	Quality		# of rows ($\lvert P\rvert$)			# of cols ($\lvert supp(P)\rvert$)		
		Avg	Std Dev	Max	Min	Avg	Max	Min	Avg
1	8,629	2.65	15.17	23	1	2.41	235	1	2.08
2	20,578	1.39	3.64	163	2	5.80	100	1	1.12
3	23,308	1.22	2.00	163	3	8.53	38	1	1.05
4	22,807	1.16	1.42	163	4	10.88	38	1	1.03
5	21,208	1.11	1.14	163	5	13.06	30	1	1.02
6	19,409	1.08	0.91	164	6	15.36	30	1	1.01
7	17,807	1.06	0.66	164	7	17.61	19	1	1.01
8	16,326	1.04	0.53	164	8	19.96	19	1	1.00
9	14,989	1.03	0.41	179	9	22.26	19	1	1.00
10	13,850	1.02	0.29	179	10	24.62	8	1	1.00

Patterns 5, 6 and 9 all refer to holiday celebrations. The locations appearing in the supports of Patterns 5 and 9 are specific locations: the route of the Macy's Thanksgiving Day Parade[26], outlined in Figure 5.6a, and Times Square, in Figure 5.6e, where New Year's Eve was celebrated[27]. Pattern 6 deals with a Christmas celebration that took place throughout the city.

It is not easy to identify the events relating to Patterns 7 and 8. Both occurred in single locations during small windows of time, but the terms used in the pattern, while consistent between them, are too generic. Pattern 7 is about style and shopping, whereas pattern 8 deals with graphic design and contemporary art.

5.1.3.2 Biclusters Discovered in the Los Angeles Matrix

Table 5.11 shows basic statistics of the collections of biclusters `Biceps` outputs given the LA matrix, $k \in \{1, \ldots, 10\}$, and the quality function in Equation 5.1. As with the NYC matrix, fewer biclusters are discovered with the quality function tailored to the discovery of events.

Table 5.12 lists the muscly patterns getting the ten highest grades. Again, five new patterns emerge, while patterns 4, 6, 7, 8 and 9 in Table 5.6 disappear. Figure 5.7 shows the locations appearing in the supports of the new patterns, patterns 4, 5, 7, 8 and 9 in Table 5.12.

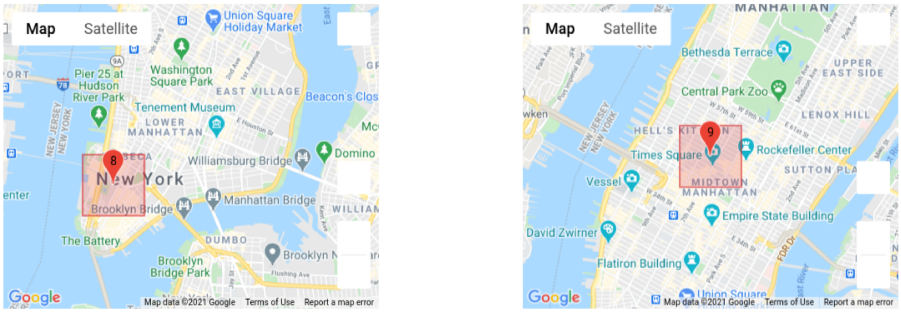
Pattern 4 deals with VidCon[28], a multi-genre online video tech conference that

[26] `https://tinyurl.com/y3r7umya`

[27] `https://www.cbsnews.com/news/new-year-celebrations-times-square-2017/`

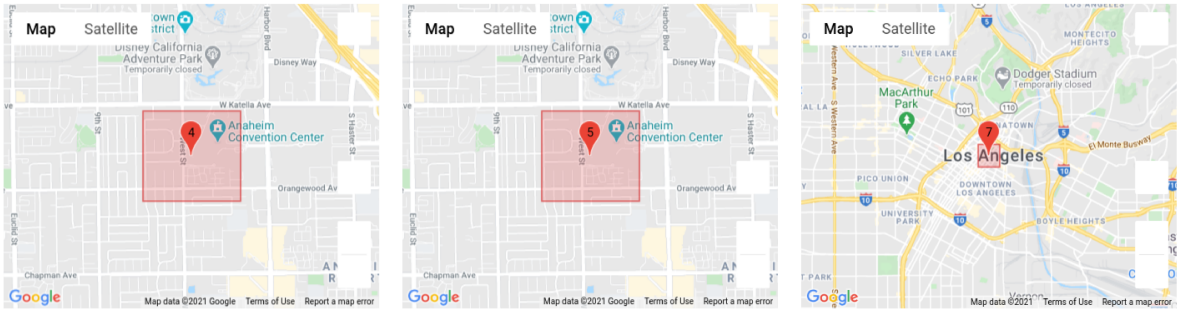
[28] `https://www.crunchbase.com/event/vidcon-2017-2017621`

(a) Pattern 5 (b) Pattern 6 (c) Pattern 7

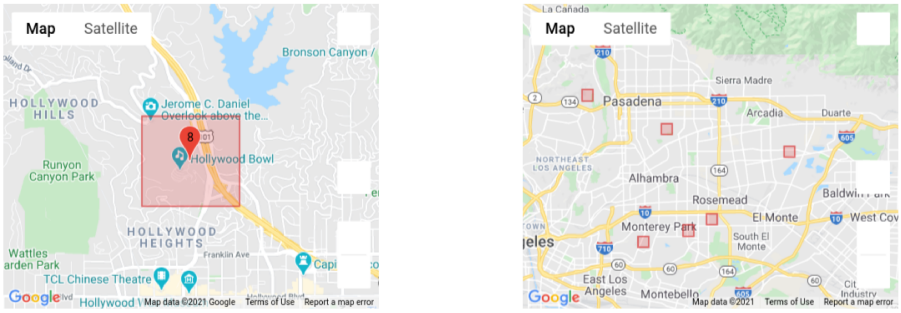


(d) Pattern 8 (e) Pattern 9

Figure 5.6: Locations relating to the columns supporting each of the new muscly patterns obtained with q as in Equation 5.1, among those with the highest grades in the NYC matrix with $k = 2$.



(a) Pattern 4 (b) Pattern 5 (c) Pattern 7



(d) Pattern 8 (e) Pattern 9

Figure 5.7: Locations relating to the columns supporting each of the new muscly patterns obtained with q as in Equation 5.1, among those with the highest grades in the LA matrix with $k = 2$.

Table 5.10: Muscly biclusters with the ten highest grades in the NYC matrix with $k = 2$ and q as in Equation 5.1.

Pattern	P	$\lvert supp \rvert$	Quality	Period
1	#nycc2016, #nycc	8	303.00	2016/10/08 - 2016/10/11; 2016/10/14
2	#happyhalloween, #halloween	31	170.00	2016/10/29 - 2016/10/31
3	#vote, #imwithher, #election2016	26	169.00	2016/11/07 - 2016/11/08
4	#lotusflame, #thank	3	137.00	2016/11/06 ; 2016/11/20
5	#macysthanksgiving-dayparade, #thanksgiving	5	126.00	2016/11/24
6	#merrychristmas, #christmas	12	106.00	2016/12/23 - 2016/12/25
7	#stylemepretty, #stylebloggers, #styleoftheday, #shopping, #stylist, #style	3	76.00	2016/12/13 - 2016/12/15
8	#graphicdesign, #contemporaryart, #design, #art	1	72.00	2017/01/04
9	#timessquare, #happynewyear	1	57.00	2016/12/31
10	#rockefellercenter, #christmas	22	53.00	Multiple days between 2016/11/28 and 2016/12/27

happened from June 21st to June 24th. It was held at the Anaheim Convention Center, which is the location associated with every column in the support, as Figure 5.7a indicates.

Pattern 5 relates to the D23 Expo[29], a biennial Disney fan event, which took place from July 14th to July 16th. The Anaheim Convention Center hosted it too. The related location is again the only one appearing in the support, as Figure 5.7b indicates.

Pattern 7 refers to the lighting up of the "Bat-signal" in Los Angeles on June 15th. It was a tribute to the late American actor Adam West[30], who passed away on June 9th. The Bat-signal was projected onto the Los Angeles City Hall. One single

[29] `https://d23.com/d23-expo-2017-dates-announced/`

[30] `https://tinyurl.com/y5u9b8jz`

Table 5.11: Statistics of the collections of muscly biclusters in the LA matrix with $k \in \{1, \ldots, 10\}$ and q as in Equation 5.1.

k	$\lvert\mathcal{M}\rvert$	Quality		# of rows ($\lvert P\rvert$)			# of cols ($\lvert supp(P)\rvert$)		
		Avg	Std Dev	Max	Min	Avg	Max	Min	Avg
1	12,674	1.81	9.32	20	1	2.44	245	1	1.68
2	29,704	1.18	3.16	82	2	5.59	104	1	1.07
3	30,881	1.10	1.55	117	3	7.83	40	1	1.04
4	28,644	1.07	1.01	129	4	9.91	40	1	1.02
5	25,620	1.05	0.73	129	5	11.94	40	1	1.01
6	22,456	1.04	0.70	166	6	14.01	18	1	1.01
7	19,777	1.04	0.69	166	7	16.23	18	1	1.01
8	17,421	1.03	0.67	166	8	18.44	14	1	1.00
9	15,544	1.02	0.43	166	9	20.68	14	1	1.00
10	13,961	1.01	0.19	214	10	22.98	5	1	1.00

Table 5.12: Muscly biclusters with the ten highest grades in the LA matrix with $k = 2$ and q as in Equation 5.1.

Pattern	P	$\lvert supp\rvert$	Quality	Period
1	#e3, #e32017	12	419.00	Multiple days between 2017/06/10 and 2017/06/21
2	#ax2017, #animeexpo, #animeexpo2017	9	198.00	2017/06/29 - 2017/07/05
3	#fyf, #fyffest	8	160.00	2017/07/21 - 2017/07/25
4	#vidconus, @vidcon, #vidcon2017, #vidcon	5	83.00	2017/06/21 - 2017/06/25
5	#d23expo2017, #d23, #d23expo, #disney	4	83.00	2017/07/14 - 2017/07/16; 2017/07/18
6	#u2thejoshuatree2017, #u2	2	65.00	2017/05/20 - 2017/05/21
7	#adamwest, #batman	1	63.00	2017/06/15
8	#hollywoodbowl, #lalaland	3	51.00	2017/05/26 - 2017/05/28
9	#myphoto, #mypic, #natureaddict, #flower, #flowerslovers, #flowers, #blossoms, #nature	12	48.00	Multiple days between 2017/05/20 and 2017/06/08
10	#fuckfakeshit, #why, #fuckfakes, #need, #gardengrove,#real, #graffiti, #live, #life	14	42.00	Multiple days between 2017/06/02 and 2017/07/26

Table 5.13: Statistics of the collections of muscly biclusters in the London matrix with $k \in \{1, \ldots, 10\}$ and q as in Equation 5.1.

k	$\lvert\mathcal{M}\rvert$	Quality		# of rows ($\lvert P\rvert$)			# of cols ($\lvert supp(P)\rvert$)		
		Avg	Std Dev	Max	Min	Avg	Max	Min	Avg
1	11,828	1.76	4.88	22	1	2.34	686	1	1.63
2	26,473	1.17	1.60	60	2	5.47	686	1	1.07
3	27,179	1.10	0.84	109	3	7.67	47	1	1.02
4	24,340	1.07	0.70	109	4	9.63	33	1	1.01
5	21,330	1.05	0.56	113	5	11.55	33	1	1.01
6	18,674	1.04	0.42	124	6	13.55	33	1	1.01
7	16,306	1.03	0.38	141	7	15.63	33	1	1.01
8	14,307	1.02	0.32	141	8	17.78	33	1	1.01
9	12,720	1.01	0.27	141	9	19.85	8	1	1.00
10	11,340	1.01	0.21	141	10	21.96	8	1	1.00

column support the pattern. It associates the day signal was lit up with the location of the City Hall, as expected and shown in Figure 5.7c.

Pattern 8 deal with the *La La Land in Concert: A Live-to-Film Celebration*[31], an event that featured the live orchestra accompanying the 2017 film *La La Land*'s voice recording. The concert premiered on May 26th. A second screening happened on May 28th. Supporting columns associated with both days contributed to the grade, because they are in a 3-day window. All those columns relate to the locations of the Hollywood Bowl, as indicated by Figure 5.7d. Both presentations took place there.

Pattern 9 does not refer to a specific event we could identify. As with pattern 10, which remains in the top-10 despite the new quality function. Multiple days appear in its support and the pattern itself contains generic terms. They are, however, consistent among themselves.

5.1.3.3 Biclusters Discovered in the London Matrix

Table 5.13 shows basic statistics of the collections of bicluster `Biceps` outputs given the London matrix, $k \in \{1, \ldots, 10\}$, and the quality function in Equation 5.1. The same conclusions as in the previous experiments are drawn comparing Table 5.13 to Table 5.7.

Using the new quality function tailored to the discovery of events, patterns 4 and 10, in Table 5.8, leave the top-10. Some false positive patterns remain, patterns 3, 5 and 9. Only two new patterns appear in the top-10: patterns 9 and 10 in Table 5.14.

[31] `https://tinyurl.com/y2g9ntsp`

Table 5.14: Muscly biclusters with the ten highest grades in the London matrix with $k = 2$ and q as in Equation 5.1.

Pattern	P	$\lvert supp \rvert$	Quality	Period
1	#wonderland, #takethat	13	134.00	Multiple days between 2017/06/06 and 2017/06/20
2	#adele, #wembley	4	77.00	2017/06/28 - 2017/06/30; 2017/07/02
3	#endorphins, #endomondo	686	69.00	2017/05/17 - 2017/05/31; 2017/06/03 - 2017/07/26
4	#londonpride, #pride	9	58.00	2017/07/08 - 2017/07/09
5	#hiring, #job	63	51.00	Multiple days between 2017/05/17 and 2017/07/26
6	#notinthislifetime, #gunsnroses	6	48.00	2017/06/16 - 2017/06/20
7	@wimbledon, #tennis, #wimbledon	15	38.00	Multiple days between 2017/06/26 2017/07/15
8	#tasteoflondon, @tasteoflondon	6	37.00	2017/06/14 - 2017/06/18; 2017/06/21
9	#prayforlondon, #londonbridge	3	37.00	2017/06/03 - 2017/06/04
10	#u2thejoshuatree2017, #u2	3	36.00	2017/07/08 - 2017/07/09; 2017/07/11

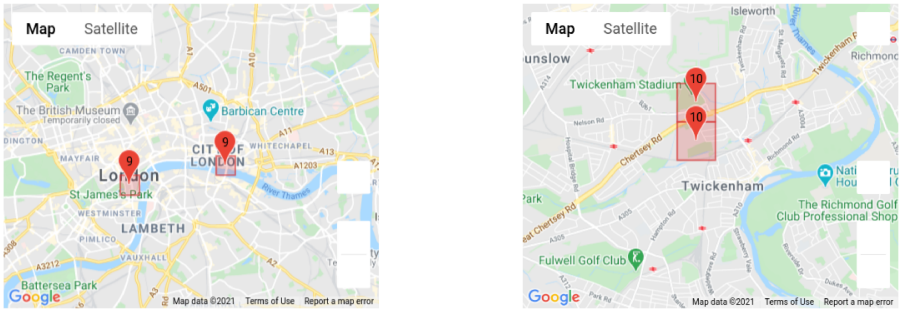
Pattern 9 refers to the 2017 London Bridge attack[32]. That terrorist attack targeted the aforementioned bridge on June 3rd. The supporting columns are associated with that day and the day after, when the public was still strong. The two locations highlighted in Figure 5.8a correspond to both the London Bridge and the center of the city.

Pattern 10 relates to two U2 shows for the *Joshua Tree Tour 2017*. In London, the shows took place on July 8th and July 9th, at the Twickenham Stadium, covered by the highlighted locations in Figure 5.8b.

5.2 Mining Dense Matrices Originating From a QSAR Model

This set of experiments uses datasets generated from a Quantitative structure–activity relationship (QSAR) model. QSAR models are used to study the relationship between

[32] `https://www.bbc.com/news/uk-england-london-40147164`



(a) Pattern 9 (b) Pattern 10

Figure 5.8: Locations relating to the columns supporting each of the new muscly patterns obtained with q as in Equation 5.1, among those with the highest grades in the London matrix with $k = 2$.

the chemical structure (i. e., the position of atoms in a tridimensional space, the electronic structure, among other factors) and the biological activity, which describes the effects of a drug on a living matter. It is well accepted in the scientific community that the physical-chemical properties and the structural attributes of the chemical compounds are responsible for the biological activity of the proteins with which molecules interact. Thus, the ability of a molecule to interact with proteins in a biological system, for example, is related to the configuration in which its atoms are organized in a tridimensional space and to the electronic forces that act on this interaction [Ferreira, 2002].

The motivation for applying `Biceps` on this type of dataset is to discover interesting biclusters based on the electronic forces generated around some molecules. We want to discover common regions around a set of molecules, for which the electronic forces in the common regions follow the same trend in all molecules, e. g., these regions presents the highest level of interaction within all the space around the molecule. We then expect that these biclusters are useful to explain some characteristic about these molecules.

5.2.1 Dataset

The dataset used in this first set of experiments is developed using a new approach to a dataset first used by Martins et al. [2009]. 49 molecules are studied here. They are benzylidene, oxazolidinediones and thiazolidinediones derivatives inhibitors of the enzyme 17β-hydroxysteroid dehydrogenase type 3. That enzyme converts androstenedione to active androgen testosterone, which is over-expressed in hormone-dependent prostate cancer and is considered a potential target to the chemotherapy of that cancer [Harada

et al., 2012]. Every molecule is associated with two scalar fields: at each point of the tridimensional space, the Lennard-Jones potential models repulsive (at very short distance) and attractive (at moderate distance) interactions, considering the molecule is electronically neutral, whereas the Coulomb potential models the interactions due to the electric charges in the molecule. The potential at a given point is measured as the interaction between the molecule and a probe, which can be an atom, an ion, or a functional group. Muscly biclusters are searched in two matrices, corresponding to the two types of interactions. The rows and columns respectively stand for the 49 molecules and for a finite set of tridimensional points where the potentials (the values in the matrices) are measured.

Tenório et al. [2017] proposes a new way to sample the tridimensional space around the molecule. That method takes into consideration the shape of the molecule. The molecules sharing a same core, they are easily aligned. Given the position of the atoms in a molecule, the convex hull is computed. The sampled points, hence the columns, are defined in a spherical coordinate system. Its origin is the center of the molecule. Every pair (θ, ϕ) of angles enumerated between 0 and $360 - \delta$ degrees in steps of δ degrees are associated with seven radial distances. Those distances are defined from that of the intersection between the surface of the convex hull and the line passing by the origin with the direction given by (θ, ϕ). The first radial distance is 2.5 Å greater, because the molecule and the protein it interacts with can hardly be closer. Every subsequent radial distance increases the previous one by 1 Å, until 8.5 Å away from the convex hull, where both interactions become negligible. Tenório et al. [2017] have shown that way of sampling the space around the 49 molecules supports more accurate predictions, by regression, of their biological activities than using a Cartesian grid, as originally proposed by Martins et al. [2009].

Figure 5.9 shows two vectors that each end in a point where the Coulomb potential and the Lennard-Jones potential are computed. Figure 5.10 represents in blue the sampled points with a same θ coordinate. The red polygon is the surface of the convex hull of the molecule.

The number of columns in the matrix depends on δ. Table 5.15 shows the number of columns for all values of δ that are used during the experiments with this dataset. Whatever that value, $m_{\text{non-min}} = 48$, $m_{\text{distinct}} = 49$ and the resulting matrix is dense.

5.2.2 Results

Muscly patterns are mined in the two matrices (relating to the two types of potential) that are obtained with $\delta = 2^o$. Each matrix therefore has 112,154 columns, in accor-

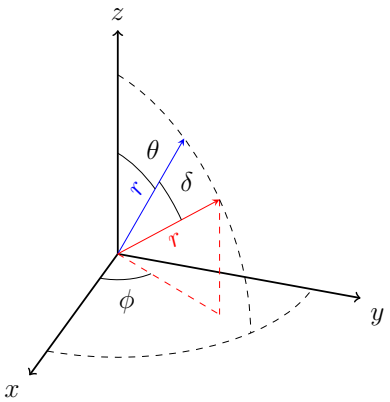


Figure 5.9: Two points with the same ϕ and r coordinates and consecutive θ coordinates, i. e., differing by δ

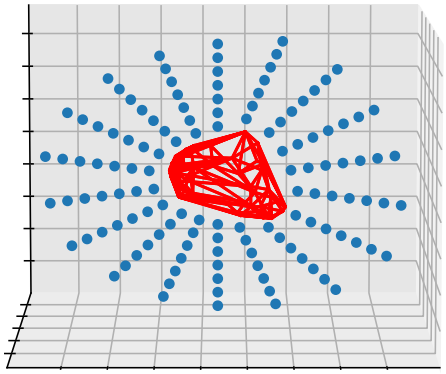


Figure 5.10: Points with a same θ coordinate sampled around a molecule with $\delta = 20$

dance with Table 5.15. In each experiment the quality functions in Equations 3.1 and 3.2 are employed, hence `Biceps` for dense matrices is executed twice on each matrix.

5.2.2.1 Muscly Biclusters Graded With the Function in Equation 3.1

Equation 3.1 defines quality function used in this experiment. That quality function quantifies to what extent a bicluster stands out in the matrix. As a consequence, muscly patterns are subsets of the molecules for which the potentials at the points in the support stand out when compared with those of the remaining molecules. More precisely, at the point relating to any column in the support, any molecule in the muscly pattern generates a higher potential than any molecule not in the muscly pattern. The grade is the sum over every such point of the difference between the smallest potential generated by a molecule in the muscly pattern and the highest potential generated by a molecule not in in the muscly pattern. It is almost proportional to the approximation by the midpoint rule of the integral over the region of the space encompassing the

Table 5.15: Number of columns in the matrix for each value chosen for δ

δ	n
2	112,154
3	49,574
4	27,734
5	17,654
6	12,194
7	9,114
8	6,944
9	5,334
10	4,298

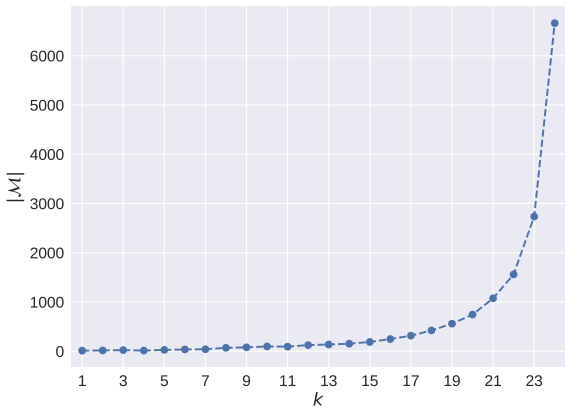
points of the difference between the min of the scalar fields generated by the molecules in the pattern and the max of the scalar fields generated by the molecules out of it. It is "almost" proportional because the voxels centered on points farther away from the molecules are larger.

The biclusters discovered in the Lennard-Jones dataset are summarized in Figure 5.11 for all possible values of k. The choice of k, i. e., the minimum number of rows in a muscly pattern, has a great influence on the qualities of the discovered biclusters. Figure 5.11b the median quality decreases when k increases and the ratio between the maximum and minimum increases.

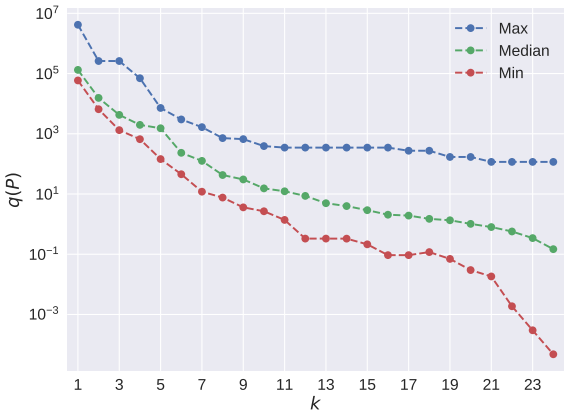
By Definition 2, the highest grade is necessarily nonincreasing with k. Indeed, a greater k always defines a subset of adequately-sized patterns. Highly graded patterns in the difference disappear from the output and subsets with lower grades replace them. Usually, several subsets or supersets per disappearing pattern, as the increase of $|\mathcal{M}|$, the number of muscly patterns in function of k indicates.

Although a muscly pattern P, or its complement $R \setminus P$, containing one single molecule may receive a high grade $q(P)$, such a pattern is irrelevant for this application. It aims to discover larger sets of molecules that may bind with a protein for the same reason: a low Lennard-Jones potential in the same region around them. Figure 5.11d shows that with this matrix, a higher k brings smaller supports. At $k = 12$ or more, there is always at least on muscly pattern supported by one single columns. Nevertheless, there are always remains muscly patterns with much larger supports.

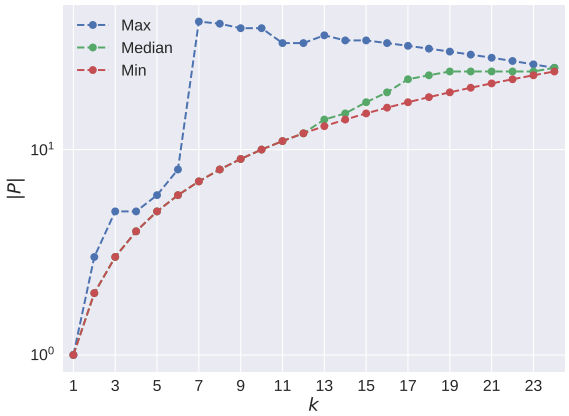
The analogous results for the Coulomb matrix are shown in Figure 5.12. Figure 5.12a shows that the number of muscly patterns obtained in this matrix is smaller than in the Lennard-Jones matrix, whatever k. Besides that, the general conclusions drawn from the statistical analysis of the collections of muscly patterns in the Lennard-



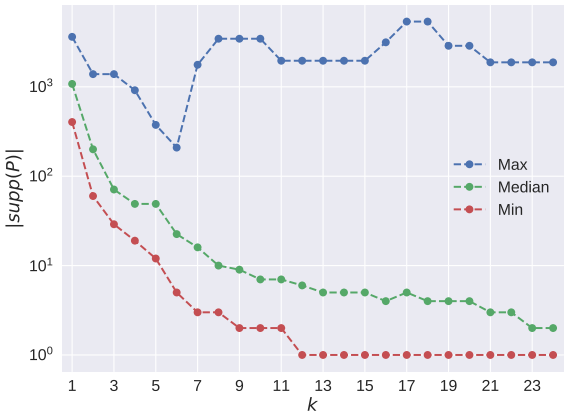
(a) Number of muscly biclusters discovered



(b) Minimum, median and maximum grade



(c) Minimum, median and maximum muscly pattern sizes



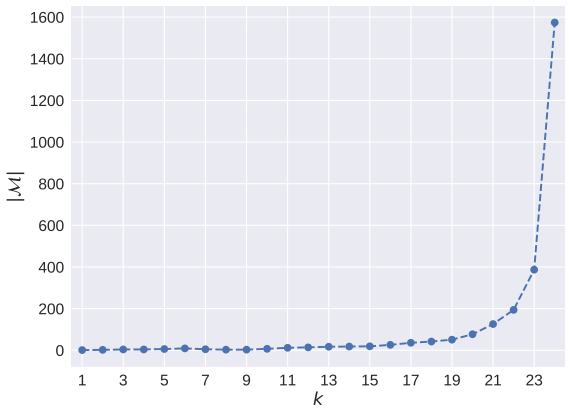
(d) Minimum, median and maximum support sizes

Figure 5.11: Statistics about the muscly patterns in the Lennard-Jones matrix in function of $k \in \{1, \ldots, 24\}$.

Jones matrix for $k \in \{1, \ldots, 24\}$ apply to the Coulomb matrix.

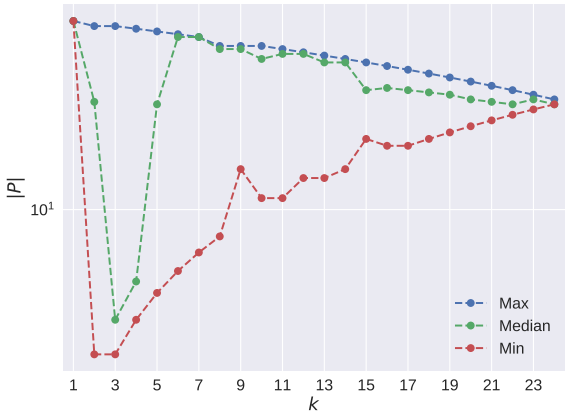
`Biceps` implicitly outputs the complementary patterns of the muscly patterns: every $R \setminus P$ such that $P \subseteq R$ is a muscly pattern. $supp(P)$ contains the columns where the values in the rows of $R \setminus P$ are all strictly *smaller* than those on the remaining rows, in P. When smaller values are meaningful, it makes more sense to discuss the biclusters $(P \subseteq R, supp(P))$ than to discuss $(P, supp(P))$. It is the case for the matrices containing Lennard-Jones potentials: regions of the space where that potential is more negative are regions of interest. In matrices with Coulomb potentials, the muscly pattern P is interpreted when $supp(P)$ relates to points around the molecule where positive potentials are measured and the complementary pattern, $P \subseteq R$, is interpreted instead if negative potentials are measured at those same points.

Using domain knowledge, a chemist, João Paulo Ataide Martins, interpreted the

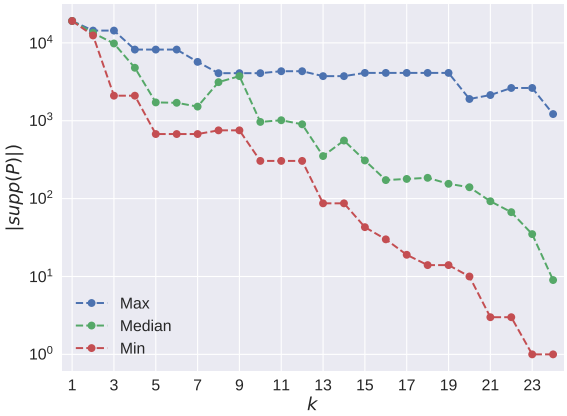


(a) Number of muscly biclusters discovered

(b) Minimum, median and maximum grade



(c) Minimum, median and maximum muscly pattern sizes



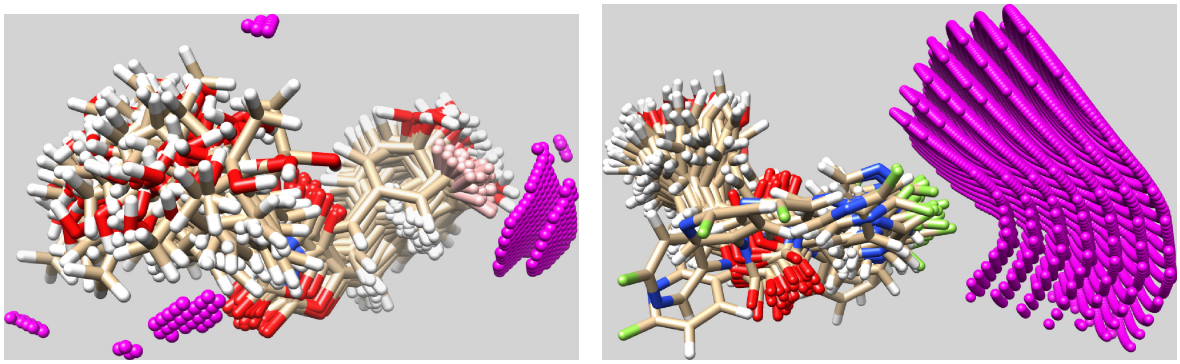
(d) Minimum, median and maximum support sizes

Figure 5.12: Statistics about the muscly patterns in the Coulomb matrix in function of $k \in \{1, \ldots, 24\}$.

16 (respectively, 2) biclusters `Biceps` outputs given the Lennard-Jones (respectively, Coulomb) matrix and $k = 2$. Although he affirms all 18 biclusters are relevant, this thesis, focusing on computer science, only reports below his interpretation of two biclusters, one per matrix. Only one molecule of the pattern is displayed, but the analysis is valid for all molecules in said bicluster.

It is worth noticing that the points in the supports of any of the 18 muscly patterns are clustered in regions. The scalar fields being continuous, it chemically makes sense. Nothing in the definition of the muscly patterns favors the discovery of whole regions of the space. That discovery is purely data-driven. A different quality function could however be designed to focus on biclusters involving clustered points. It may even only use the potentials in the most significant region of the space to compute the grade.

A total of 16 biclusters were found for the Lennard-Jones matrix. In Figure 5.13a, bromine atoms (pink), which are big, are responsible for the very negative Lennard-



(a) Lennard-Jones (b) Coulomb

Figure 5.13: For each QSAR matrix, positions of the points in the support of a muscly pattern, plotted around one of the molecules in the patterns

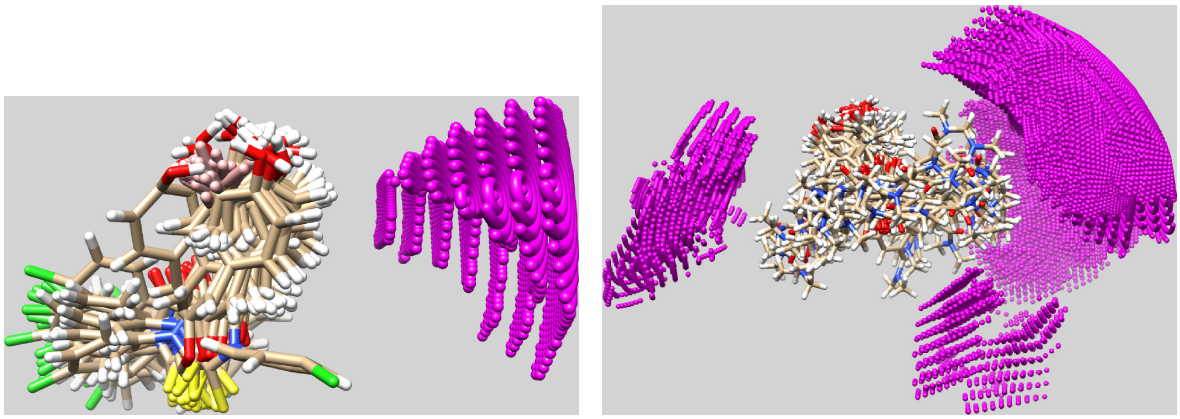
Jones potentials measured in the region on the right of the Figure. The points of the support that are in the smaller regions on the left are close to alkyl groups that may interact with a protein through hydrophobic bonds.

A total of 2 biclusters were found for the Coulomb matrix. Figure 5.13b shows one of the molecule of a muscly pattern in the Coulomb matrix and the points of the support. Those points correspond to a region close to the clorine atoms (green) that may be interacting with amino acids through dipole-dipole interactions or halogen bonds, i. e., polar interactions. The same interpretation holds for the remaining molecules in the pattern.

5.2.2.2 Muscly Biclusters Graded With the Function in Equation 3.2

In this experiment, the quality function defined in Equation 3.2 grades the adequately-sized patterns. It only depends on the number of molecules (rows) and points (columns) in the graded bicluster. For a same number of points, the grade exponentially grows with how close the number of molecules is to $\frac{49}{2} = 24.5$. That is why, in the following results, every muscly pattern P, and its complement $R \setminus P$, contains at least three molecules, although `Biceps` is called with $k = 1$. Calling it with $k = 2$ or $k = 3$ therefore provides the same output.

Tables 5.16 and 5.17 report the size and qualities of the one single bicluster in the Lennard-Jones matrix and two muscly biclusters in Coulomb matrix. Both the numbers of molecules and of points are large. Given the choice of quality function, that was expected. As before, for each dataset, the analysis of single bicluster is provided. Also, a single molecule per muscly pattern is displayed, but the given explanations are valid for all its molecules.



(a) Coulomb (b) Lennard-Jones

Figure 5.14: For each QSAR matrix, positions of the points in the support of a muscly pattern, graded with the quality function in Equation 3.2, plotted around one of the molecules in the patterns

Figure 5.14b represents the only muscly bicluster in the Lennard-Jones matrix. The points in the support belong to three distinct regions. Two of them are close to nitrogen atoms (blue). The last one, in the opposite side of the molecule, may relate to nitrogen atoms as well as to oxygen atoms (red).

Table 5.16: Muscly biclusters, graded with the quality function in Equation 3.2, in the Lennard-Jones matrix

P	$q(P)$	$\|P\|$	$\|supp(P)\|$
P_1	236,091	45	17,273

Figure 5.14a depicts one of the two biclusters in the Coulomb matrix. The points of the support are all in a region around aromatic rings in the ligand that may be interacting with an amino acid through π-stacking interactions.

Table 5.17: Muscly biclusters, graded with the quality function in Equation 3.2, in the Coulomb matrix

P	$q(P)$	$\|P\|$	$\|supp(P)\|$
P_1	175,693	18	4,698
P_2	141,860	46	14,386

According to João Paulo Ataide Martins, although they can be interpreted, the biclusters `Biceps` outputs given the quality function in Equation 3.2 are far less relevant to QSAR than those obtained with the function in Equation 3.1. In particular, there

are too few muscly patterns and their supports cover too much of the space around the molecules. The function in Equation 3.2 favors large biclusters. It does not take into account the values in the matrix, i. e., the same muscly patterns are defined in the matrix where every value is replaced by its ranking in the column. For those reasons, the lack of relevance and, in particular, of specificity, were expected.

5.2.3 Robustness to Changes of the Sampling Granularity

The smaller δ, the finer the sampling of the space around the molecules and the more columns in the matrix. The results in Section 5.2.2 are obtained with $\delta = 2^o$, a rather high granularity. It allows to relate the support of a pattern to precise regions of the space around the molecules. This section studies the robustness of the collection of muscly biclusters with respect to δ. Larger values of δ are defined, hence grosser samplings of the space, resulting in matrices with fewer columns. A similarity between the muscly biclusters in each of those matrices and the muscly biclusters in the matrices obtained with $\delta = 2^o$ is reported. The latter biclusters therefore play the role of a ground truth. Given two patterns in two different matrices, computing how similar their supports are requires to map every column of the matrix $\mathbf{M}$ obtained with the larger δ to a set of columns of the matrix $\mathbf{M}'$ obtained with the smaller δ (always 2^o in the experiment). To do so, the Voronoi diagram of the set of points corresponding to the columns of $\mathbf{M}$ is considered. In this way, any column c of $\mathbf{M}$ is associated with a Voronoi cell. c is mapped to the set of columns of $\mathbf{M}'$ that corresponds to points belonging to the Voronoi cell. Given the regularity of the considered samplings of the space, the mapping can be formalized more simply. Let C be the set of spherical coordinates identifying the columns of $\mathbf{M}$, C' the analogous set for $\mathbf{M}'$ and δ the angle step used to define C, the mapping is:

$$map : C \longrightarrow 2^{C'}$$
$$(r' =, \theta, \phi) \mapsto \left\{ (r', \theta', \phi') \in C' \mid (r = r') \wedge (\theta - \frac{\delta}{2} \leq \theta' \leq \theta + \frac{\delta}{2}) \wedge (\phi - \frac{\delta}{2} \leq \phi' \leq \phi + \frac{\delta}{2}) \right\}$$

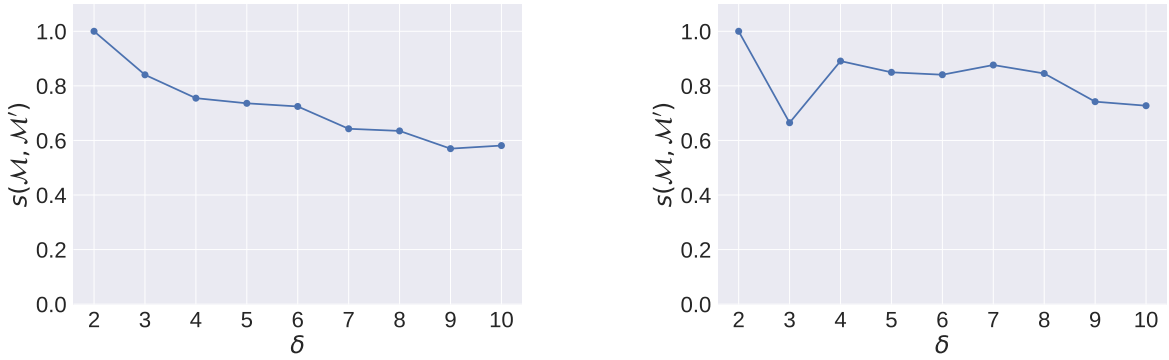
In order to compare the set of muscly patterns $\mathcal{M}$ in a matrix to the set $\mathcal{M}'$ in the matrix obtained with $\delta = 2^o$, the match score in 2.2, proposed by Liu and Wang [2007], is adapted in Equation 5.2: the columns in the support of any $P \in \mathcal{M}$ are

replaced by $\bigcup_{c \in supp(P)} map(c)$.

$$s(\mathcal{M}, \mathcal{M}') = \frac{1}{|\mathcal{M}'|} \sum_{P' \in \mathcal{M}'} \max_{P \in \mathcal{M}} \frac{|P \cap P'| + |\bigcup_{c \in supp(P)} map(c) \cap supp(P')|}{|P \cup P'| + |\bigcup_{c \in supp(P)} map(c) \cup supp(P')|} \tag{5.2}$$

That score averages the similarity, based on the Jaccard index, between every bicluster discovered in the matrix obtained with $\delta = 2^o$ and the most similar bicluster discovered in the other matrix. The value of k is set to 2, the quality function to that of Equation 3.3.1 with the ℓ_1 norm.

Figure 5.15 shows the similarity score tends to decrease, when δ increases by steps of 1^o from 2^o to 10^o, as grosser samplings of the space around the molecules lead to smaller similarities. Indeed, the shape of the regions encompassing the points in the supports are more and more poorly defined. Nevertheless, the similarity remains high up to $\delta = 10$ degrees, especially for the biclusters dealing with Coulomb potentials. The similarity measure does not penalize the presence of more muscly patterns than in the ground truth. However, whatever the tested sampling granularity δ, there are exactly two muscly patterns in the matrices dealing with Coulomb potentials and at most two supernumerary (i. e., at most 18) muscly patterns in the matrices whose values are Lennard-Jones potentials. The small former number, two, explains why, at $\delta = 3$ degrees in Figure 5.15b, one single molecule missing from one single pattern with three molecules (in the ground truth) is mostly responsible for a large drop of the similarity.



(a) Lennard-Jones potential

(b) Coulomb potential

Figure 5.15: Similarity between the biclusters `Biceps` discovers in the matrix obtained with $\delta = 2^o$ and those in matrices related to grosser samplings of the space ($\delta > 2^o$). k is 2.

5.3 Running Time and Memory Usage

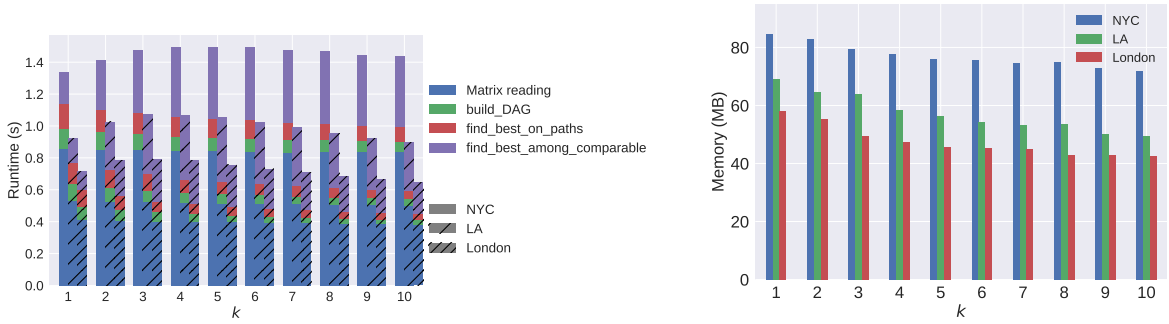
This section analyzes `Biceps`' runtime and memory usage. For all datasets, the reported measures are averages over 30 executions of the algorithm.

5.3.1 Performance on the Twitter Matrices

As stated before, the Twitter matrices are sparse. That is why `Biceps` for sparse matrices is used. Figure 5.16a shows how long `Biceps`' execution takes when given each of the three Twitter matrices, the quality function in Equation 3.1 and $k \in \{1, \ldots, 10\}$. Most of the time is spent reading the matrix from the input file, and Algorithm 1's last step fills most of the remaining time.

The time required to build the DAG is low: 0.13 seconds for the NYC matrix, 0.10 seconds for the LA matrix and 0.08 seconds for the London matrix, when $k = 1$. It decreases when k increases. The peak memory usage, shown in Figure 5.16b and always reached at the end of the construction of the DAG, is low as well: 84.80 MB for the NYC matrix, 69.13 MB for the LA matrix and 57.92 MB for the London matrix, when $k = 1$. It also decreases with k. To understand those observations, Figure 5.17 helps. In it, the number of adequately-sized patterns $|\mathcal{A}|$, which is equal to the number of vertices $\sum_{i=k}^{m-k} |\mathcal{V}_i|$ in the DAG, is rather small and decreasing with k. That number, $|\mathcal{A}|$, depends on the area under the green curve in Figure 5.2 (without the logarithmic scales). That is why, in Tables 4.1 and 4.2, the complexities associated with the DAG construction involve m_{distinct}, which is 41 for the NYC matrix, 25 for the LA matrix and 22 for the London Matrix (see Table 5.2). The computational requirements for that step would be significantly higher if `Biceps` for sparse matrices would not take advantage of the many repetitions of values in every column of the Twitter matrix: they would depend on the area under the blue curves in each subfigure in Figure 5.2 and, in Tables 4.1 and 4.2, the related complexities would involve $m_{\text{non-min}} = 2{,}344$ instead of $m_{\text{distinct}} = 40$ for the NYC matrix, and $m_{\text{non-min}} = 1{,}860$ and $m_{\text{non-min}} = 1{,}532$ for the LA and London matrices, respectively. `Biceps` for dense matrices requires more than the available 20 GB of RAM to build the DAG from the Twitter matrix, because it creates far more vertices for assuming few repeated values per column (see Section 4.3.2) and indexing each vertex takes m bits (see Table 5.2) in a compressed bitwise trie, instead of at most $m_{\text{non-min}}$ row identifiers (integers) in a hash map.

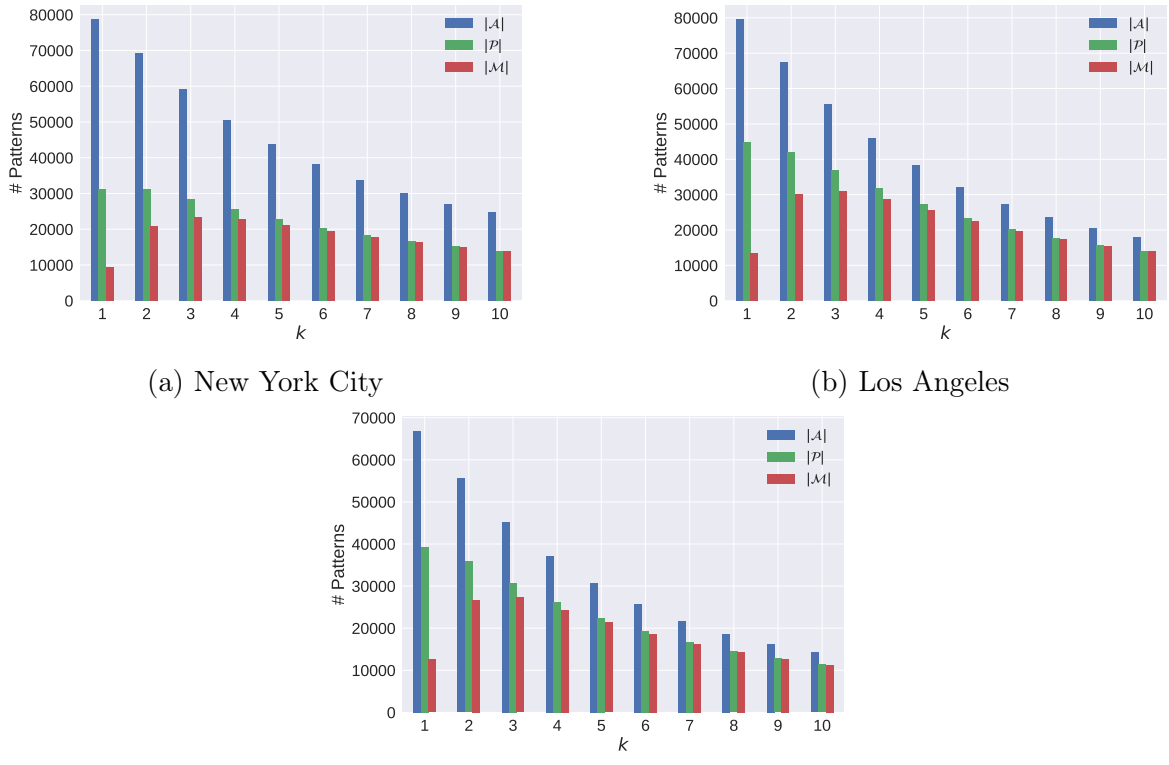
Given any of the three matrices, `Biceps`' overall runtimes are small. Using $k = 2$, as in Section 5.1.2, `Biceps` takes 1.41 seconds (0.56 s disconsidering the parsing) to list the muscly biclusters in the NYC matrix, 1.03 s (0.5 s disconsidering the parsing)



(a) Runtime

(b) Peak memory usage

Figure 5.16: `Biceps`' time and space requirements on the Twitter matrices with q as in Equation 3.1 and $k \in \{1, \dots, 10\}$.

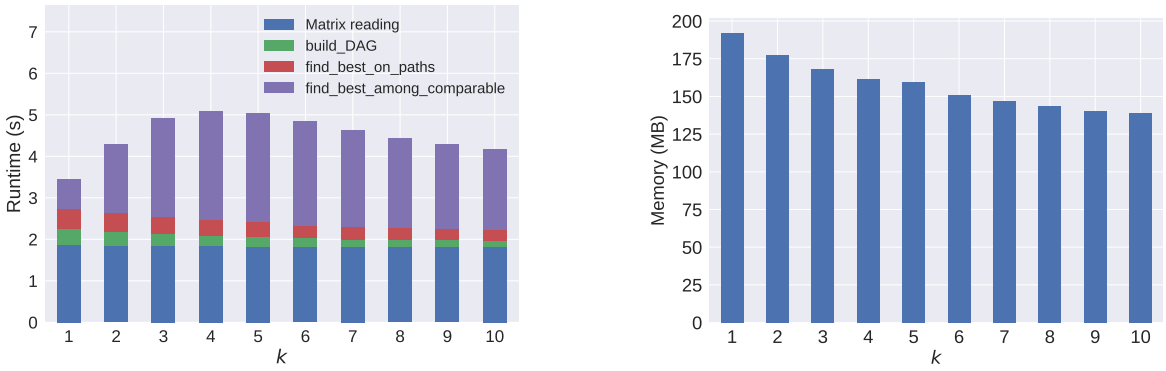


(a) New York City

(b) Los Angeles

(c) London

Figure 5.17: Decreasing number of candidate patterns along `Biceps`' executions on the Twitter matrices with q as in Equation 3.1 and $k \in \{1, \dots, 10\}$.



(a) Runtime

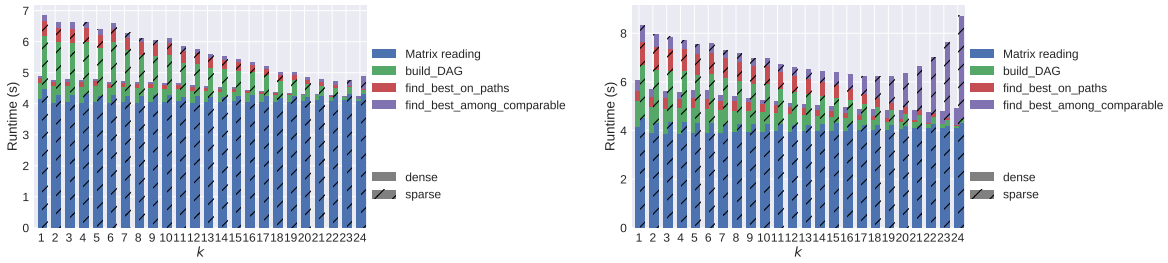
(b) Peak memory usage

Figure 5.18: `Biceps`' time and space requirements on the combined Twitter matrix with q as in Equation 3.1 and $k \in \{1, \ldots, 10\}$.

in the LA matrix, and 0.78 s (0.38 s disconsidering the parsing) in the London matrix. In an attempt to get larger time and space requirements, the three input files are concatenated. The resulting matrix has $m = 631{,}532$ rows, $n = 174{,}559$ columns and 2,575,425 nonzero values. $m_{\text{non-min}}$ is 2,339 and $m_{\text{distinct}} = 40$. As Figure 5.18 shows, `Biceps` still given the quality function in Equation 3.1 and $k \in \{1, \ldots, 10\}$, requires between 3.46 and 5.092 seconds and between 139 MB and 192 MB to compute the muscly patterns. For $k = 2$, as in Section 5.1.2, the muscly patterns receiving the ten highest grades are present in Table 5.4, 5.6 or 5.8. Among them, $\{\#endorphins, \#endomondo\}$ is top-graded, because locations in all three cities appear in its support.

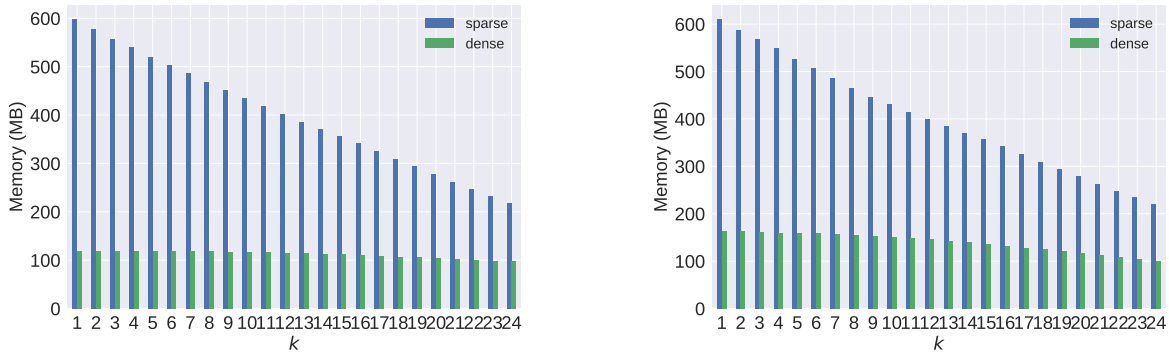
5.3.2 Performance on the QSAR Matrices

The QSAR matrices are dense. Anyway, for comparison, both the dense and sparse versions of `Biceps` are executed, always using the quality function in Equation 3.1. Figure 5.19 shows their runtimes for all possible values of k. As expected, `Biceps` for dense matrices is here faster than the version for sparse matrices. Significantly faster if the time to read the input from the disk is deducted. Indeed, reading the input takes most of the time. After the data are read, whatever k, the dense version of `Biceps` computes the muscly biclusters in less than 1 second for the Coulomb matrix; less than 2 seconds for the Lennard-Jones matrix. With `Biceps` for sparse matrices, those times are respectively 2.36 seconds and 3.80 seconds, hence around three and two times greater, respectively. Figure 5.20 shows the memory peak usages of the same executions. `Biceps` for sparse matrices requires between 2 and 5 times more space than the version for dense matrices, which suits the QSAR matrices.



(a) Coulomb (b) Lennard-Jones

Figure 5.19: Runtimes of the dense and sparse versions of `Biceps` on the QSAR matrices with q as in Equation 3.1 and $k \in \{1, \ldots, 10\}$.



(a) Coulomb (b) Lennard-Jones

Figure 5.20: Peak memory usage of the dense and sparse versions of `Biceps` on the QSAR matrices with q as in Equation 3.1 and $k \in \{1, \ldots, 10\}$.

5.3.3 Performance on a Gene Expression Matrix

`Biceps`' computational requirements on the QSAR matrices are low, as they only contain 49 rows. Moreover, with $\delta = 2$ degrees, the four seconds to read the matrix from the hard disk dominate the reported run times. To more meaningfully study the performances of `Biceps` for dense matrices, muscly patterns are mined in a larger dense matrix. A gene expression matrix is chosen, for being an ubiquitous benchmark dataset to assess the performance of biclustering algorithms. The muscly patterns in that matrix are not interpreted because we do not have the necessary knowledge in genomics to do so. Any value in the matrix is the expression level of a gene (a column among $n = 20{,}531$) measured for a patient (a row, among $m = 801$) affected by a tumor [Weinstein et al., 2013][33]. The matrix is dense: $m = 801$, $m_{\text{distinct}} = 801$. That is why, `Biceps` for dense matrices is used. In addition to the matrix, it is given the quality function in Equation 3.1 and $k \in \{50\delta + 1 \mid \delta \in \{0, \ldots, 7\}\}$.

In addition to the matrix, `Biceps` for dense matrices is given the quality function

[33] `https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq`

Table 5.18: Numbers of the patterns computed by each of Algorithm 1's three steps on the gene expression matrix with q as in Equation 3.1 and $k \in \{50\delta + 1 \mid \delta \in \{0, \ldots, 7\}\}$.

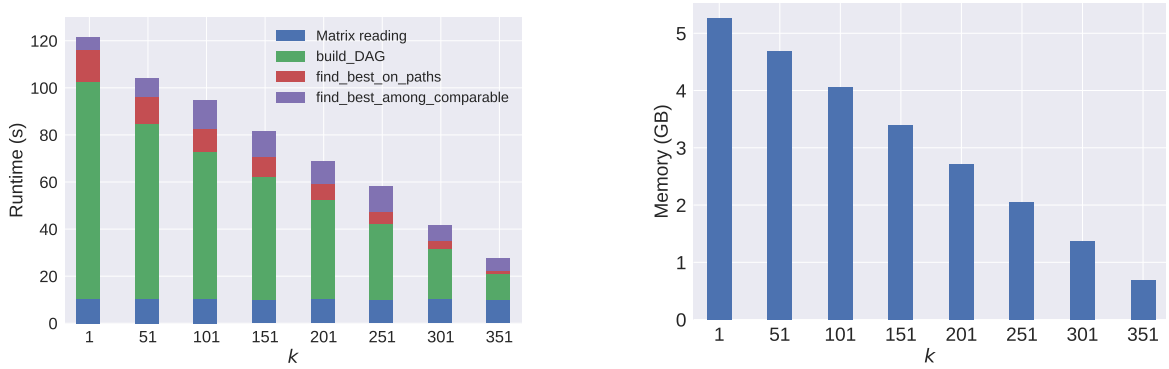
k	$\sum_{i=k}^{m-k} \lvert\mathcal{V}_i\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{M}\rvert$
1	14,043,871	95	2
51	12,363,298	19,484	98
101	10,609,029	19,122	3,644
151	8,847,799	18,827	12,577
201	7,082,452	18,596	16,123
251	5,313,696	18,372	16,899
301	3,543,742	18,157	17,258
351	1,772,153	17,935	17,522

in Equation 3.1 with the ℓ_1 norm and $k \in \{1, \ldots, 15\}$. For each execution, Figure 5.21a shows how long it takes to read the matrix from the hard disk and to complete each of Algorithm 1's three steps. Building the DAG requires most of the time. Given Table 4.1 and, for that matrix, $m \ll n$, the small duration of the last step may come as a surprise. The reported time complexity is actually very pessimistic: in practice, for each pattern Algorithm 5 tests, reaching its adequately-sized subpatterns and superpatterns only requires traversing a small proportion of the DAG, not all of it as in the worst-case scenario.

Figure 5.21b depicts the peak memory consumption (always reached at the end of the construction of the DAG) in function of k. In the tested settings, `Biceps` for dense matrices takes about two minutes and less than 5.3 GB of RAM to list the muscly patterns. In contrast, `Biceps` for sparse matrices requires more than the available 20 GB of RAM. That is mainly because, for dense matrices, using hash maps for the vertex indices, which dominate the space complexity, is much worse than using compressed bitwise tries.

Table 5.18 reports $\sum_{i=k}^{m-k} \lvert\mathcal{V}_i\rvert$, $\lvert\mathcal{P}\rvert$ and $\lvert\mathcal{M}\rvert$, i. e., the number of patterns at the output of each of Algorithm 1's three steps. Given the first improvement Section 4.3.2 presents, the space the DAG occupies is essentially proportional to its number of vertices $\sum_{i=k}^{m-k} \lvert\mathcal{V}_i\rvert$. Since $k \ll m$, its slow decreasing is unsurprising. It explains why the time to build the DAG and the memory usage slowly decrease too. With $k = 100$, they are respectively 60.91 seconds and 4.06 GB. With $k = 200$, 41.21 seconds and 2.73 GB.

`Biceps` for sparse matrices variation of the algorithm causes the memory usage to exceed the 20 GB available. The reason for that is that, the data, especially the bitsets in the tries used in the sparse version, dominate the memory requirements, not the structure to hold them.



(a) Runtime

(b) Peak memory usage

Figure 5.21: `Biceps`' time and space requirements on the gene expression matrix with q as in Equation 3.1 and $k \in \{50\delta + 1 \mid \delta \in \{0, \ldots, 7\}\}$.

5.4 Summary

This Chapter presented three sets of experiments. The first set of experiments was conducted on sparse matrices of Twitter terms. The quality function in Equation 3.1 was used first, and then adapted to become the tailored quality function presented in Equation 5.1, which better takes explores the underlying event detection characteristic presented in the results for this dataset. The second set of experiments was conducted on dense matrices generated from a Quantitative structure–activity relationship (QSAR) model. For this experiment, the quality functions presented in both Equation 3.1 and Equation 3.2 were used. The results were analyzed by a chemist, to ensure the discovered biclusters were relevant. Finally, the third set of experiments explored the runtime and memory usage performance of the algorithm, in both dense and sparse matrices.

Chapter 6

Conclusion

In this work, we proposed a new type of bicluster, namely the muscly bicluster. In any of its columns, the values in the rows of the bicluster must be all strictly greater than those in the rows absent from it. Moreover, the rows of the bicluster must not be a subset or a superset of the rows of another bicluster of greater or equal quality. Any computable function can be chosen so assign qualities to the biclusters. In that respect, the proposed definition is generic. This type of bicluster, while similar to existing classifications of biclusters, had not yet been explored in the literature. In addition to the defining the muscly biclusters, the thesis has detailed an algorithm, namely `Biceps`, to efficiently discover them. `Biceps` relies on dynamic programming. It is complete, which brings certainty: `Biceps` exhaustively lists all the muscly biclusters, and only them. The thesis has described two versions of `Biceps`, one for dense matrices and one for sparse matrices. They mainly differ in their choices of data structures. The time complexity of either version is subquadratic in $m_{\text{non-min}}$, where $m_{\text{non-min}}$ is the maximal number of non-minimal values in a column, hence essentially the total number of rows if the matrix is dense, but possibly much less if it is sparse.

The simplicity and the genericity of the definition make it useful in various applications dealing with the analysis of dense or sparse matrices. The experimental chapter has shown it, by interpreting biclusters discovered in datasets originating from very different domains. The biclusters discovered by `Biceps` were relevant to the applications in which the algorithm was used. Furthermore, the speed to compute the biclusters and the completeness of the output are great advantages derived from its polynomial nature. In the first set of experiments, geolocated and timestamped hashtags and user mentions published on Twitter from New York City, Los Angeles and London have been analyzed. Using a general-purpose quality function, the top-graded biclusters have been found to mostly indicate events happening in those cities. That

has motivated a modification of the quality function that penalizes biclusters that are widespread in time. After the modification, essentially all the well-graded muscly biclusters relate to events, hence an example of how `Biceps` is adaptable to specific pattern discovery. The second set of experiments has dealt with a very different domain of application, chemistry, or more precisely, Quantitative structure–activity relationship. An expert has found the muscly biclusters in matrices containing either Coulomb or Lennard-Jones potential to be relevant. Moreover, they have been shown robust to changes in the granularity with which the space around the molecules is sampled. The last set of experiments has focused on `Biceps`' time and memory consumption using the Twitter and QSAR matrices, as well as an additional large and dense matrix. `Biceps`' low theoretical requirements have been shown practical too, as long as the adequate version, for either spare or dense matrices, is chosen.

The subquadratic complexity and the limited size of the output, which cannot reach that of the input, enable large-scale uses. Those uses are still largely to be invented. The article has not much explored the possibilities that the genericity of the definition opens up. Since any computable function can grade the biclusters, it may take into account information not necessarily present in the mined matrix and process it in an arbitrary complex way, maybe tailored to the particularities of the application. For example, in a supervised setting, where the rows or the columns of the matrix are either partitioned into classes (for instance the cities in the Twitter matrix) or associated with the values of a target variable (for instance quantifying the biological activities of the molecules in a QSAR matrix), the function could favor biclusters that appear as relevant features for classification or regression.

The completeness of the collection looks particularly useful for future work that would post-process `Biceps`' output. In particular, it may help the computation of less constrained patterns, perhaps of every muscly pattern P with a δ-noise-tolerant support, which could be every column of the matrix where the top-$(|P| + \delta)$ values include all those in P's row. To feed biclusters to a larger system, several collections obtained with different minimal numbers of rows, k, may be desired. `Biceps`' low time requirements and the fact that its first step would only have to run once facilitate such future usages.

Bibliography

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, page 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ayadi, W., Elloumi, M., and Hao, J.-K. (2009). A biclustering algorithm based on a Bicluster Enumeration Tree: application to DNA microarray data. *BioData Mining*, 2(1):9. ISSN 1756-0381.

Ben-Dor, A., Chor, B., Karp, R., and Yakhini, Z. (2003). Discovering Local Structure in Gene Expression Data: The Order-Preserving Submatrix Problem. *Journal of Computational Biology*, 10(3-4):373--384. ISSN 1066-5277.

Benczúr, A., Bíró, I., Brendel, M., Csalogány, K., Daróczy, B., and Siklósi, D. (2007). Cross-modal retrieval by text and image feature biclustering. In Nardi, A., Peters, C., and Ferro, N., editors, *Working Notes for the CLEF 2007 Workshop*, volume 1173, Budapest, Hungary. ISSN 1613-0073.

Bendimerad, A., Plantevit, M., Robardet, C., and Amer-Yahia, S. (2019). User-driven geolocated event detection in social media. *IEEE Transactions on Knowledge and Data Engineering*, 33(2):796--809. ISSN 1041-4347.

Burdick, D., Calimlim, M., and Gehrke, J. (2001). MAFIA: a maximal frequent itemset algorithm for transactional databases. In *Proceedings 17th International Conference on Data Engineering*, pages 443--452, Heidelberg, Germany. IEEE Comput. Soc. ISSN 1063-6382.

Busygin, S., Jacobsen, G., Kramer, E., Krämer, E., and Ag, C. (2002). Double Conjugated Clustering Applied to Leukemia Microarray Data. In *Proceedings of the 2nd SIAM ICDM, Workshop on clustering high dimensional data*.

Busygin, S., Prokopyev, O., and Pardalos, P. M. (2008). Biclustering in data mining. *Computers & Operations Research*, 35(9):2964--2987. ISSN 0305-0548.

Cheng, K.-O., Law, N.-F., Siu, W.-C., and Liew, A. (2008). Identification of coherent patterns in gene expression data using an efficient biclustering algorithm and parallel coordinate visualization. *BMC Bioinformatics*, 9(1):210. ISSN 1471-2105.

Cheng, Y. and Church, G. M. (2000). Biclustering of Expression Data. *Proceedings. International Conference on Intelligent Systems for Molecular Biology*, 8:93--103. ISSN 1553-0833.

Cheung, L., Cheung, D. W., Kao, B., Yip, K. Y., and Ng, M. K. (2007). On mining micro-array data by order-preserving submatrix. *International Journal of Bioinformatics Research and Applications*, 3(1):42--64.

Chun Tang, Li Zhang, Aidong Zhang, and Ramanathan, M. (2001). Interrelated two-way clustering: an unsupervised approach for gene expression data analysis. In *Proceedings 2nd Annual IEEE International Symposium on Bioinformatics and Bioengineering (BIBE 2001)*, pages 41--48. IEEE. ISSN 0250-4162.

Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pages 269--274, New York, New York, USA. ACM Press.

Dhillon, I. S., Mallela, S., and Modha, D. S. (2003). Information-theoretic co-clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, page 89, New York, New York, USA. ACM Press.

Ferreira, M. M. C. (2002). Multivariate QSAR. *Journal of the Brazilian Chemical Society*, 13(6):742--753. ISSN 0103-5053.

Gao, B., Liu, T.-y., and Ma, W.-y. (2006). Star-Structured High-Order Heterogeneous Data Co-clustering Based on Consistent Information Theory. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 880--884. IEEE. ISSN 1550-4786.

Getz, G., Levine, E., and Domany, E. (2000). Coupled two-way clustering analysis of gene microarray data. *Proceedings of the National Academy of Sciences*, 97(22):12079--12084. ISSN 0027-8424.

Gouda, K. and Zaki, M. (2001). Efficiently mining maximal frequent itemsets. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 163--170. IEEE Comput. Soc. ISSN 1550-4786.

Goyal, A., Ren, R., and Jose, J. M. (2010). Feature Subspace Selection for Efficient Video Retrieval. In Boll, S., Tian, Q., Zhang, L., Zhang, Z., and Chen, Y.-P. P., editors, *Advances in Multimedia Modeling*, pages 725--730. Springer Berlin Heidelberg, Berlin, Heidelberg.

Greco, G., Guzzo, A., and Pontieri, L. (2007). An Information-Theoretic Framework for Process Structure and Data Mining. *International Journal of Data Warehousing and Mining*, 3(4):99--119. ISSN 1548-3924.

Gu, J. and Liu, J. S. (2008). Bayesian biclustering of gene expression data. *BMC Genomics*, 9(Suppl 1):S4. ISSN 1471-2164.

Harada, K., Kubo, H., Tanaka, A., and Nishioka, K. (2012). Identification of oxazolidinediones and thiazolidinediones as potent 17β-hydroxysteroid dehydrogenase type 3 inhibitors. *Bioorganic & Medicinal Chemistry Letters*, 22(1):504--507. ISSN 0960-894X.

Hartigan, J. A. (1972). Direct Clustering of a Data Matrix. *Journal of the American Statistical Association*, 67(337):123--129. ISSN 0162-1459.

Hofmann, T. and Puzieha, J. (1999). Latent class models for collaborative filtering. In *IJCAI International Joint Conference on Artificial Intelligence*. ISSN 1045-0823.

Jaccard, P. (1912). THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. *New Phytologist*, 11(2):37--50. ISSN 0028-646X.

Kluger, Y. (2003). Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions. *Genome Research*, 13(4):703--716. ISSN 1088-9051.

Lazzeroni, L. and Owen, A. (2002). Plaid Models for Gene Expression Data. *Statistica Sinica*, 12(1):61--86. ISSN 1017-0405.

Liu, J. and Wang, W. (2003). OP-Cluster: Clustering by Tendency in High Dimensional Space. In *Third IEEE International Conference on Data Mining*, pages 187--194. IEEE Comput. Soc. ISSN 1550-4786.

Liu, X. and Wang, L. (2007). Computing the maximum similarity bi-clusters of gene expression data. *Bioinformatics*, 23(1):50--56. ISSN 1367-4803.

Liu, Z., Xue, Y., Li, M., Ma, B., Zhang, M., Chen, X., and Hu, X. (2017). Discovery of deep order-preserving submatrix in dna microarray data based on sequential pattern mining. *International Journal of Data Mining and Bioinformatics*, 17(3):217--237.

Madeira, S. and Oliveira, A. (2004). Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24--45. ISSN 1545-5963.

Martins, J. P. A., Barbosa, E. G., Pasqualoto, K. F. M., and Ferreira, M. M. C. (2009). LQTA-QSAR: A New 4D-QSAR Methodology. *Journal of Chemical Information and Modeling*, 49(6):1428--1436. ISSN 1549-9596.

Mirkin, B. (1996). *Mathematical Classification and Clustering*, volume 11 of *Nonconvex Optimization and Its Applications*. Springer US, Boston, MA. ISBN 978-1-4613-8057-3.

Murali, T. M. and Kasif, S. (2003). Extracting Conserved Gene Expression Motifs From Gene Expression Data. *Pacific Symposium on Biocomputing*, 8:77--88. ISSN 2335-6928.

Padilha, V. A. and Campello, R. J. G. B. (2017). A systematic comparative evaluation of biclustering techniques. *BMC Bioinformatics*, 18(1):55. ISSN 1471-2105.

Peeters, R. (2003). The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651--654. ISSN 0166-218X.

Prelić, A., Bleuler, S., Zimmermann, P., Wille, A., Bühlmann, P., Gruissem, W., Hennig, L., Thiele, L., and Zitzler, E. (2006). A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22(9):1122--1129. ISSN 1460-2059.

Rege, M., Dong, M., and Fotouhi, F. (2006). Co-clustering Documents and Words Using Bipartite Isoperimetric Graph Partitioning. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 532--541. IEEE. ISSN 1550-4786.

Segal, E., Taskar, B., Gasch, A., Friedman, N., and Koller, D. (2001). Rich probabilistic models for gene expression. *Bioinformatics*, 17(Suppl 1):S243–S252. ISSN 1367-4803.

Serin, A. and Vingron, M. (2011). DeBi: Discovering Differentially Expressed Biclusters using a Frequent Itemset Approach. *Algorithms for Molecular Biology*, 6(1):18. ISSN 1748-7188.

Sheng, Q., Moreau, Y., and De Moor, B. (2003). Biclustering microarray data by Gibbs sampling. *Bioinformatics*, 19(Suppl 2):ii196–ii205. ISSN 1367-4803.

Sim, K., Gopalkrishnan, V., Chua, H. N., and Ng, S.-K. (2009). MACs: Multi-Attribute Co-clusters with High Correlation Information. In Buntine, W., Grobelnik, M., Mladeniç, D., and Shawe-Taylor, J., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 398--413, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sun, X., Hou, Q., Ren, Z., Zhou, K., and Guo, B. (2011). Radiance Transfer Biclustering for Real-Time All-Frequency Biscale Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):64--73. ISSN 1077-2626.

Tanay, A., Sharan, R., and Shamir, R. (2002). Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(Suppl 1):S136–S144. ISSN 1367-4803.

Tanay, A., Sharan, R., and Shamir, R. (2005). Biclustering Algorithms: A Survey. *Handbook of Computational Molecular Biology*, 9(May):1--20.

Teng, L. and Chan, L. (2008). Discovering Biclusters by Iteratively Sorting with Weighted Correlation Coefficient in Gene Expression Data. *Journal of Signal Processing Systems*, 50(3):267--280. ISSN 1939-8018.

Tenório, J. V. S., Cerf, L., and Ataide, J. P. (2017). A new approach for sampling descriptors in 4D-QSAR methodology using computational geometry. In *SBQT'17: Atos do XIX Simpósio Brasileiro de Química Teórica*, pages 4--5.

Trapp, A. C., Li, C., and Flaherty, P. (2018). Recovering all generalized order-preserving submatrices: new exact formulations and algorithms. *Annals of Operations Research*, 263(1):385--404.

Ungar, L. H. and Foster, D. P. (1998). A Formal Statistical Approach to Collaborative Filtering. *Proceedings of the conference on automated learning and discovery (CONALD'98)*.

Veroneze, R. (2016). *Enumerating all maximal biclusters in numerical datasets*. PhD dissertation, Universidade Estadual de Campinas.

Wang, H., Wang, W., Yang, J., and Yu, P. S. (2002). Clustering by pattern similarity in large data sets. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data - SIGMOD '02*, page 394, New York, New York, USA. ACM Press. ISSN 0730-8078.

Weinstein, J. N., Collisson, E. A., Mills, G. B., Shaw, K. R. M., Ozenberger, B. A., Ellrott, K., Shmulevich, I., Sander, C., and Stuart, J. M. (2013). The Cancer Genome Atlas Pan-Cancer analysis project. *Nature Genetics*, 45(10):1113--1120. ISSN 1061-4036.

Xue, Y., Li, T., Liu, Z., Pang, C., Li, M., Liao, Z., and Hu, X. (2018). A new approach for the deep order preserving submatrix problem based on sequential pattern mining. *International Journal of Machine Learning and Cybernetics*, 9(2):263--279.

Xue, Y., Li, T., Zhang, H., Wu, X., Li, M., and Hu, X. (2016). An apriori-based algorithm for mining semi-order-preserving submatrix. *International Journal of Computational Science and Engineering*, 13(1):66--79.

Xue, Y., Li, Y., Deng, W., Li, J., Tang, J., Liao, Z., and Li, T. (2014). Mining order-preserving submatrices based on frequent sequential pattern mining. In *Proceedings of the 3rd International Conference on Health Information Science*, pages 184--193.

Xue, Y., Liao, Z., Li, M., Luo, J., Kuang, Q., Hu, X., and Li, T. (2015). A new approach for mining order-preserving submatrices based on all common subsequences. *Computational and Mathematical Methods in Medicine*, 2015.

Yang, J., Wang, H., Wang, W., and Yu, P. (2003). Enhanced biclustering on expression data. In *Third IEEE Symposium on Bioinformatics and Bioengineering, 2003. Proceedings.*, pages 321--327, Bethesda, MD, USA. IEEE Comput. Soc.

Yang, J., Wang, W., Wang, H., and Yu, P. (2002). δ-Clusters: Capturing Subspace Correlation in a Large Data Set. In *Proceedings 18th International Conference on Data Engineering*, pages 517--528, San Jose, CA, USA. IEEE Comput. Soc. ISSN 0305-0548.

Zhao, H., Wee-Chung Liew, A., Z. Wang, D., and Yan, H. (2012). Biclustering Analysis for Pattern Discovery: Current Techniques, Comparative Studies and Applications. *Current Bioinformatics*, 7(1):43--55. ISSN 1574-8936.