Mach Learn (2011) 82: 123–155
DOI 10.1007/s10994-010-5210-y

Multi-way set enumeration in weight tensors

Elisabeth Georgii · Koji Tsuda · Bernhard Schölkopf

Received: 1 June 2009 / Accepted: 1 May 2010 / Published online: 25 September 2010

Abstract The analysis of n-ary relations receives attention in many different fields, for instance biology, web mining, and social studies. In the basic setting, there are n sets of instances, and each observation associates n instances, one from each set. A common approach to explore these n-way data is the search for n-set patterns, the n-way equivalent of itemsets. More precisely, an n-set pattern consists of specific subsets of the n instance sets such that all possible associations between the corresponding instances are observed in the data. In contrast, traditional itemset mining approaches consider only two-way data, namely items versus transactions. The n-set patterns provide a higher-level view of the data, revealing associative relationships between groups of instances. Here, we generalize this approach

Editors: S.V.N. Vishwanathan, Samuel Kaski, Jennifer Neville, and Stefan Wrobel.

E. Georgii
Department of Empirical Inference, Max Planck Institute for Biological Cybernetics, Tübingen, Germany

E. Georgii
Friedrich Miescher Laboratory of the Max Planck Society, Tübingen, Germany

Present address:
E. Georgii (✉)
Department of Information and Computer Science, Helsinki Institute for Information Technology, HIIT, Aalto University School of Science and Technology, P.O. Box 15400, 00076 Aalto, Finland
e-mail: elisabeth.georgii@tkk.fi

K. Tsuda
Computational Biology Research Center, National Institute of Advanced Industrial Science and Technology, AIST, Tokyo, Japan
e-mail: koji.tsuda@aist.go.jp

K. Tsuda
ERATO Minato Project, Japan Science and Technology Agency, Tokyo, Japan

B. Schölkopf
Department of Empirical Inference, Max Planck Institute for Biological Cybernetics, Tübingen, Germany
e-mail: bernhard.schoelkopf@tuebingen.mpg.de

in two respects. First, we tolerate missing observations to a certain degree, that means we are also interested in n-sets where most (although not all) of the possible associations have been recorded in the data. Second, we take association weights into account. In fact, we propose a method to enumerate all n-sets that satisfy a minimum threshold with respect to the average association weight. Technically, we solve the enumeration task using a reverse search strategy, which allows for effective pruning of the search space. In addition, our algorithm provides a ranking of the solutions and can consider further constraints. We show experimental results on artificial and real-world datasets from different domains.

Keywords Tensor · Multi-way set · Dense pattern enumeration · Quasi-hyper-clique · N-ary relation · Graph mining

1 Introduction

Higher-order data analysis is an important subfield in modern pattern mining. Associations relating multiple types of instances are often represented by multi-way arrays (also known as tensors). As an example, let us consider sales data that contain information about the products, the regions, and the weeks of customer transactions. These data can be compiled into a three-way array where the first dimension represents the products, the second dimension represents the regions, and the third dimension represents the weeks. The array elements can be arbitrary weight values. In the given example, different scenarios are conceivable. For instance, we can consider binary values, setting an element to one if the product was sold in the corresponding region during the given week, and otherwise to zero. Another possibility would be to report the total number of customers who bought the product in the given region and week (which results in non-negative integer values). Finally, one could store average customer ratings, which could be arbitrary real values.

The analysis of such multi-way data becomes increasingly popular in the data mining community. Tensor decomposition is one of the most prominent topics (see Kolda and Bader 2007 for a review). This can serve as a basis for applications like clustering of the instances in each dimension or anomaly detection (Kolda and Sun 2008). Furthermore, there exist approaches to analyze dynamic changes in tensors (Sun et al. 2006). A common approach to investigate binary-valued n-way data comes from relational data mining. This line of research focuses on the detection of n-sets (Cerf et al. 2008; Jaschke et al. 2006; Ji et al. 2006), which generalize the concept of itemsets (Agrawal and Srikant 1994; Han and Kamber 2006) to multi-way association data. More precisely, let $V_1, \dots, V_n$ be instance sets (for example, products, regions, and weeks). Each association of individual instances $(v_1, \dots, v_n)$, $v_i \in V_i$, has a value that is either 1 (if it actually has been observed) or 0 (otherwise). Then, a combination of instance subsets $(S_1, \dots, S_n)$, $S_i \subset V_i$ is called an n-set if all possible associations $(v_1, \dots, v_n)$, $v_i \in S_i$, have value 1 (see Fig. 1a). The original itemset mining approach considers only two-way data, involving a set of items and a set of transactions (i.e., $n = 2$). In the tensor representation, an n-set corresponds to a subtensor that contains only 1-elements. Such patterns reveal associations between groups of instances and thereby yield insights into the higher-level association structure in the data. In our above example, an n-set analysis would detect groups of products that were popular in certain regions during a number of weeks.

In this paper, we propose a more general definition of n-set as well as an enumerative mining algorithm for this kind of pattern. A preliminary version of this work has appeared in Georgii et al. (2009b). Our criterion is the average value of the elements within a subtensor;

Example of an n-set for $n = 3$: $(\{a_1, a_2\}, \{b_1, b_2\}, \{c_1, c_2, c_3\})$

(a) Relational approach

Association	Value
(a_1, b_1, c_1)	1
(a_1, b_1, c_2)	1
(a_1, b_1, c_3)	1
(a_1, b_2, c_1)	1
(a_1, b_2, c_2)	1
(a_1, b_2, c_3)	1
(a_2, b_1, c_1)	1
(a_2, b_1, c_2)	1
(a_2, b_1, c_3)	1
(a_2, b_2, c_1)	1
(a_2, b_2, c_2)	1
(a_2, b_2, c_3)	1
average	$=1$

(b) Our approach

Association	Value
(a_1, b_1, c_1)	0.9
(a_1, b_1, c_2)	1
(a_1, b_1, c_3)	1
(a_1, b_2, c_1)	0.8
(a_1, b_2, c_2)	0.9
(a_1, b_2, c_3)	1
(a_2, b_1, c_1)	0.7
(a_2, b_1, c_2)	1
(a_2, b_1, c_3)	0.9
(a_2, b_2, c_1)	1
(a_2, b_2, c_2)	0
(a_2, b_2, c_3)	0.9
average	$\geq \theta$

Fig. 1 Illustration of the n-set properties in the relational approach and in our approach. While the relational approach is based on binary values and requires that all associations within the n-set have value 1, we allow for arbitrary weights and require that the overall average across the n-set associations is larger than a threshold

subtensors where this value exceeds a given threshold are considered as solutions. Here, the tensor is not restricted to binary values, but may contain arbitrary real-valued association weights (see Fig. 1b). Furthermore, a solution pattern might also contain some 0-elements, so missing observations are tolerated to a certain degree. This can be advantageous in applications where data are sparse (i.e., where it is likely that some observations are missing) and where the associations should be weighted differently (e.g., because the reliability or the significance of the observations is subject to variation). Consequently, we can detect strong associations between sets of instances in noisy data; such higher-level patterns strengthen individual observations and can assist in making reliable predictions of missing values.

In the following, these subtensor patterns are called (*multi-way*) *clusters*, and the average association value within a cluster is called the *cluster density*; a cluster that satisfies the minimum density threshold is called a *dense cluster*. We present a novel method to enumerate all dense clusters in a general data matrix or tensor. It extends a dense cluster enumeration approach for *symmetric* matrices (Uno 2007; Georgii et al. 2009a). Our contribution consists in (1) the formulation of n-way dense cluster enumeration by means of reverse search (Avis and Fukuda 1996), (2) implementation details to speed up the search, and (3) various extensions of the basic scheme, including output reduction and ranking as well as the integration of additional constraints. From a graph-theoretic point of view, an n-way tensor corresponds to a weighted multi-partite hyper-graph, where each hyper-edge connects n nodes that represent instances of the n different dimensions. So our approach can also be seen as extracting all densely connected subgraphs from such a hyper-graph.

The paper is organized as follows. Section 2 presents related work on pattern mining in association data. In Sect. 3, we review the basic concepts of reverse search and the dense cluster enumeration technique for symmetric matrices. In Sect. 4, we describe our algorithmic approach for dense cluster enumeration in n-way tensors. Section 5 provides some extensions, and Sect. 6 describes adaptations for tensors with inherent symmetries. Finally, we present our experimental results in Sect. 7 and discuss the general applicability of our approach in Sect. 8. The appendix contains the proofs of several lemmas.

2 Related work

The problem of identifying clusters (also called modules or communities) in symmetric pairwise association data, often represented as graphs or networks, has been studied extensively, see Schaeffer (2007) for a survey. One fundamental approach is clique discovery, i.e., the enumeration of fully connected subgraphs from an unweighted input graph (e.g., see Spirin and Mirny 2003). As real-world datasets are usually incomplete, various methods relax this criterion by tolerating missing edges to a certain extent (Haraguchi and Okubo 2006; Palla et al. 2005; Farkas et al. 2007; Uno 2007; Zeng et al. 2006) or by explicitly considering edge weights (Georgii et al. 2009a; Ulitsky and Shamir 2009). Beside these enumerative approaches to cluster detection, there exist a number of methods that employ local search techniques starting from a set of seed clusters (Everett et al. 2006; Bader and Hogue 2003). Finally, the most commonly used class of methods is based on graph partitioning (Newman 2006; Schaeffer 2007). In some approaches, clusters are not defined as hard sets of instances, but describe continuous membership levels across all instances (Höppner et al. 1999).

Furthermore, the coanalysis of multiple networks or graphs has received much attention, in particular in the context of computational biology and chemistry. While many approaches focus on the detection of frequently occurring subgraphs in large databases of small graphs (Yan and Han 2002), methods for larger networks often use density criteria combined with additional constraints (Hu et al. 2005; Jiang and Pei 2009; Robardet 2009; Yan et al. 2005; Zeng et al. 2006).

Another popular cluster detection paradigm is bi-cluster analysis; it deals with general data matrices. A bi-cluster pattern (also called bi-set) is defined by specifying a subset of rows and a subset of columns of a given matrix, and various criteria have been used in the literature to assess the interestingness of such patterns (Madeira and Oliveira 2004). For instance, some methods consider the homogeneity of values within a bi-cluster (Besson et al. 2006). In contrast, other approaches focus on the strength or density of the association (Tanay et al. 2002; Mishra et al. 2004; Yan et al. 2005).

Finally, mining of higher-order association data has recently become an important area of research. Multi-way arrays, known as tensors, arise in many different application fields, for instance sales analysis (Cerf et al. 2008), web mining (Kolda et al. 2005; Acar et al. 2006; Jaschke et al. 2006), neuroscience (Beckmann and Smith 2005), and computational biology (Zhao and Zaki 2005; Baranzini et al. 2004; Ji et al. 2006; Acar et al. 2007). The goal of tensor clustering is to partition each dimension into a pre-defined number of clusters such that the corresponding multi-way clusters are as homogeneous as possible. Current approaches include tensor decomposition methods (Kolda and Sun 2008; Kolda and Bader 2007) and approximations by clustering each dimension individually (Jegelka et al. 2009). Zhao and Zaki (2005) also mine for homogeneous clusters in three-way data (i.e., tri-clusters), but instead of specifying the number of clusters, they fix thresholds regarding the homogeneity of values along each dimension and detect overlapping cluster patterns. Binary-valued tensors can be equivalently represented as relations, and there exists a vast amount of data mining literature on extracting relevant patterns from such data, to mention in particular itemset and n-set approaches (Agrawal and Srikant 1994;

Cerf et al. 2008; Jaschke et al. 2006; Ji et al. 2006; Han and Kamber 2006). Infinite relational models (Kemp et al. 2006) also focus on binary tensors, but in contrast to relational mining methods, they aim at partitioning the tensor into blocks that contain either mostly ones or mostly zeros.

3 Reverse search

This section reviews the reverse search paradigm (Avis and Fukuda 1996). We illustrate the basic idea for the application of enumerating dense sets (clusters) in a symmetric matrix (Georgii et al. 2009a; Uno 2007), which we will extend to n-way cluster enumeration in the following section.

3.1 Dense set enumeration problem

Let us consider a symmetric association matrix, which can be seen as the adjacency matrix of a weighted graph. We denote by w_{ij} the matrix element representing the association weight between instance i and instance j. Furthermore, we define the density of an instance subset U as

$$\rho(U) = \sum_{i,j \in U, i<j} w_{ij} \Big/ \frac{|U|(|U|-1)}{2}.$$

The task is to enumerate all subsets U with $\rho(U) \geq \theta$, where θ is a pre-specified constant. From the graph-theoretic point of view, $\rho(U)$ corresponds to the weighted edge density of the subgraph induced by U.

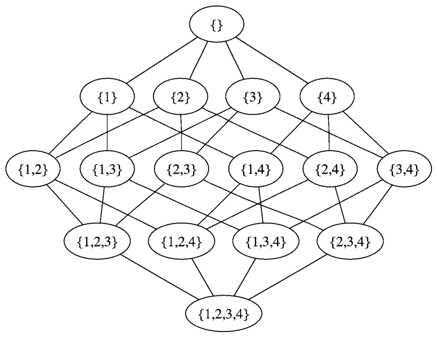
3.2 Search space

All possible subsets form a natural graph-shaped search space with multiple levels, where one can move one level downwards or upwards by adding or removing an instance, respectively; the root node of the search space corresponds to the empty set. Figure 2b shows the search space corresponding to the association matrix in Fig. 2a. For an efficient search procedure, it is crucial to avoid duplicate traversals of subspaces. This is usually achieved by defining a tree structure that spans the original graph-shaped search space. In many pattern mining approaches it is common to use lexicographical set enumeration trees (Rymon 1992) (Fig. 2c). Pruning of such a tree is straightforward if the search criterion satisfies the downward closure property: any subset of a solution is a solution, i.e., if a child is a solution, its parent is a solution as well. This is the case for frequent itemset mining (Han and Kamber 2006) because subsets have at least the same frequency as their supersets. However, in contrast to the frequency criterion, the density is generally *not* anti-monotonic. For example, while $\{1,3,4\}$ has a lower density than $\{1,3\}$ ($\rho(\{1,3\}) = 1.0$, $\rho(\{1,3,4\}) = 0.93$), the density increases when going from $\{1,2\}$ to $\{1,2,3\}$ (from 0.1 to 0.53). Consequently, the lexicographical tree cannot provide guarantees regarding the maximum density in certain subtrees, which makes it impossible to define effective pruning rules. However, there exists a search tree that shows the anti-monotonicity property with respect to the density (Fig. 2d), i.e., when following any path from the root to a leaf, the density of the visited sets is monotonically decreasing (the root has maximum density by definition). In the next subsection, we explain how such a tree can be constructed.

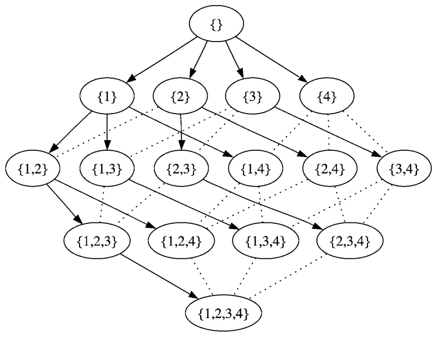
(a) Example matrix

	1	2	3	4
1	0	0.1	1.0	0.9
2	0.1	0	0.5	0
3	1.0	0.5	0	0.9
4	0.9	0	0.9	0

(b) Graph-shaped search space



(c) Lexicographical tree



(d) Reverse search tree

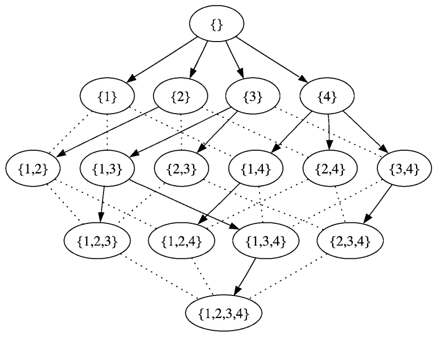


Fig. 2 Motivating example for the reverse search enumeration strategy

3.3 Reduction map

In the reverse search approach, the search tree is specified by defining a *reduction map* $f(U)$, which transforms an instance subset U into its parent. In our case, the parent is obtained by removing from the child the instance with minimum degree. Here, the *degree* of an instance is defined as the sum of its association weights to the other instances in U. If there are multiple instances with minimum degree, the one with the smallest index is removed. In Georgii et al. (2009a), it is proven that the parent constructed in this way has at least the same density as the original set U. For unweighted graphs (i.e., binary-valued matrices), the corresponding result has been stated in Uno (2007). This ensures the *anti-monotonicity* of the search tree that is induced by the reduction map; in particular, any instance set that satisfies the density threshold descends from a subset that also satisfies the density threshold. This property has also been exploited in other mining approaches, for instance as additional pruning criterion in graph pattern mining (Zhu et al. 2007); furthermore, iterative removal of minimum degree instances has been used to approximate dense subgraphs (Asahiro et al. 2000). In addition to the anti-monotonicity, a valid reduction map must satisfy the following *reachability* condition: starting from any node of the search space, we can reach the root node after applying the reduction map a finite number of times. This condition ensures that each possible solution can be reached from the root node, i.e., the induced search tree is indeed spanning. For the reduction map stated above, this is trivial, because any cluster is reduced to the empty set by iteratively removing instances.

3.4 Search strategy

To enumerate all clusters (subsets) with density $\geq \theta$, one has to traverse the implicitly defined search tree in a depth-first or breadth-first manner. During the traversal, children are generated *on demand* if the density threshold is satisfied. As the reduction map defines how to get from children to parents and not vice versa, we cannot directly derive the children from a given parent. Instead, to generate the children of a cluster U, we have to consider all candidates $U \cup \{i\}$, $i \notin U$, and apply the reduction map to every candidate; candidates with $f(U \cup \{i\}) = U$ are the actual children (*reverse search principle*). Due to the anti-monotonicity, one can prune the search tree as soon as the density threshold is violated. To conclude, reverse search provides a systematic way of obtaining anti-monotonic search trees based on reduction maps. This is helpful for score functions where the anti-monotonicity of level-wise search is not given in general, but can be achieved by a well-defined reduction map, like in the case of cluster density.

4 Multi-way dense cluster enumeration

This section presents a novel n-way cluster enumeration method based on reverse search. In order to develop a reverse search algorithm, we have to (1) define a search space, (2) design a reduction map, and (3) engineer the child generation process for efficiency. After defining the problem, we proceed in this order.

4.1 Problem definition

Our goal is to detect all dense clusters from a multi-way data array (tensor). To formalize the problem, we first introduce some notation. Let $n > 0$ be the number of *dimensions* in the given array (also called *ways* or *modes*). Then we write the input data in the following form,

$$W = (w_{k_1,\dots,k_n})_{k_i \in V_i,\, i=1,\dots,n}. \tag{1}$$

The index k_i is used to access the ith dimension and takes values from a finite index set $V_i = \{1, \dots, I_i\}$, where I_i is a natural number that can differ from dimension to dimension. V_i is also called the *instance set* or *range* for the ith dimension; the cardinality of the set is denoted by $|V_i|$ and equals I_i. The elements (entries) of W are weights indicating the association strength between the n instances. Negative values are generally possible, but non-negative input data allow for additional speed-up techniques during the search, as described in Sects. 4.6 and 5.4. For convenience, we normalize the array such that $w_{k_1,\dots,k_n} \leq 1$ for all $k_i \in V_i$, $i = 1, \dots, n$. An n-way *cluster* U is defined by specifying for each dimension a non-empty subset of the corresponding index set,

$$U = (U_1, \dots, U_n), \quad U_i \subset V_i,\ |U_i| \geq 1,\ i = 1, \dots, n. \tag{2}$$

The induced subarray is given by

$$W|_U = (w_{k_1,\dots,k_n})_{k_i \in U_i,\, i=1,\dots,n}. \tag{3}$$

Let us define the *cardinality* of a cluster as the sum of the cardinalities of the index subsets in all n dimensions, i.e., the total number of instances involved in the cluster:

$$\mathrm{card}(U) = \sum_{i=1}^{n} |U_i|.$$

This is not to be confused with the *cluster size*, which corresponds to the number of elements in the induced subarray,

$$\operatorname{size}(U) = \prod_{i=1}^{n} |U_i|.$$

Our cluster definition implies $\operatorname{size}(U) \geq 1$. The *density* of a cluster U is defined as the average value of the elements in the induced subarray,

$$\rho_W(U) = \frac{1}{\operatorname{size}(U)} \sum_{k_i \in U_i} w_{k_1,\ldots,k_n}. \tag{4}$$

Due to our normalization of the data array W, the largest possible cluster density is 1. Using the above definitions, we state the problem of dense cluster enumeration as follows.

Definition 1 (Dense cluster enumeration) Given an n-way data array W and a minimum density threshold θ with $0 < \theta \leq 1$, find all clusters U such that $\rho_W(U) \geq \theta$.

Note that different clusters are allowed to overlap. For $\theta = 1$, the problem is equivalent to n-set or hyper-clique enumeration (Cerf et al. 2008; Jaschke et al. 2006; Ji et al. 2006).

4.2 Global index representation

As defined in (1), an n-way array is represented using multiple index sets $V_1, \ldots, V_n$. To identify the vth element of set V_i in a concise way, we map it to a global index as follows:

$$\mathcal{C}(v, i) = v + \sum_{j=1}^{i-1} |V_j|.$$

For $i = 1$, the summation term is zero, i.e., $\mathcal{C}(v, 1) = v$. Then the global index set is given by

$$\mathcal{V} = \left\{1, \ldots, \sum_{i=1}^{n} |V_i|\right\}.$$

The array dimension to which $v \in \mathcal{V}$ belongs is denoted by $d(v)$.

A cluster $U = (U_1, \ldots, U_n)$ can also be represented as a subset of $\mathcal{V}$,

$$\mathcal{U} = \bigcup_{i=1}^{n} \bigcup_{u \in U_i} \{\mathcal{C}(u, i)\}.$$

Note that U and $\mathcal{U}$ are alternative representations of a uniquely determined cluster and can easily be transformed into each other. In the following, we will use the representation that is more convenient in the particular context.

4.3 Search space

The search space for our dense cluster enumeration problem consists of all possible clusters. Similarly to Sect. 3, it can be organized as a lattice, i.e., a graph with multiple levels where each node corresponds to a cluster. The root level consists of *trivial clusters*.

Definition 2 (Trivial cluster) A cluster U is called trivial if it corresponds to exactly one array element, i.e., $\mathrm{size}(U) = 1$.

Consequently, $|U_i| = 1$ for $i = 1, \dots, n$ and $\mathrm{card}(U) = n$. In the subsequent levels of the search lattice, the cluster cardinality is incrementally increased. Each cluster U is connected to the clusters from the next level that can be obtained by adding exactly one index to one particular set U_i, $1 \le i \le n$; in other words, we reach the next level when extending $\mathcal{U}$ by exactly one instance. Next, we define a reduction map for the search space.

4.4 Reduction map

A reduction map specifies an anti-monotonic search scheme, which is the core component of the reverse search framework. Our reduction map for multi-way cluster enumeration resembles the approach for symmetric matrices described in Sect. 3. However, in our case the search space has multiple root nodes, so the reduction map defines a spanning forest instead of a single spanning tree. In the following, we give its precise definition. First of all, the degree in an n-way array is computed as follows.

Definition 3 (Degree) Given a cluster U, the degree of $v \in U_j$ with respect to U is defined as

$$\deg_U(v, j) = \sum_{k_i \in U_i, i \neq j} w_{k_1,\dots,k_{j-1},v,k_{j+1},\dots,k_n}. \tag{5}$$

In the global index representation, there is no ambiguity for instances of different dimensions, so we simply write $\deg_{\mathcal{U}}(v)$ for $v \in \mathcal{U}$. Furthermore, W is not included as an explicit parameter of deg because our method deals with one given data array at a time; likewise, we often write ρ instead of ρ_W. To illustrate this definition, we consider the three-way cluster in Fig. 3. Fixing the jth dimension to a specific value v defines a slice of the cluster. The degree of v corresponds to the sum of all array elements that fall within this slice. In the following, we will use the term *slice* also for the general n-way case; the v-*slice* of a given cluster is the slice specified by the instance v. Using the global index representation, the reduction map is defined as follows.

Definition 4 (Reduction map) Let $\mathcal{U}$ be a cluster and let v be the smallest among the minimum degree elements in $\mathcal{U}$. Then, $f(\mathcal{U}) = \mathcal{U} \setminus \{v\}$.

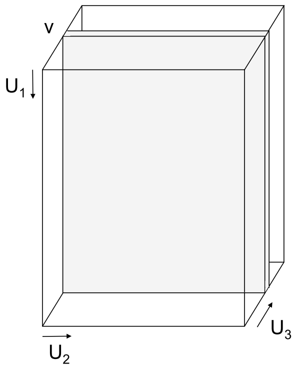


Fig. 3 Visualization of a three-way cluster. A particular instance v specifies a slice of the cluster

Algorithm 1 Pseudocode of DCE. W is the given n-dimensional data array with global index set $\mathcal{V}$ (and corresponding mapping $\mathcal{C}$), and θ denotes the minimum density threshold

```
1: DCE ($\mathcal{V}, \mathcal{C}, W, \theta$) :
2:    for each ($k_1, \dots, k_n$) with $w_{k_1,\dots,k_n} \geq \theta$ do
3:       DCE_Rec($\mathcal{V}, W, \theta, \bigcup_{i=1}^{n}\{\mathcal{C}(k_i, i)\}$)
4:    end for

1: DCE_Rec ($\mathcal{V}, W, \theta, \mathcal{U}$) :
2:    for each $v \in \mathcal{V} \setminus \mathcal{U}$ do
3:       if $\rho_W(\mathcal{U} \cup \{v\}) \geq \theta$ then
4:          if $\mathcal{U} \cup \{v\}$ is child of $\mathcal{U}$ then
5:             DCE_Rec ($\mathcal{V}, W, \theta, \mathcal{U} \cup \{v\}$)
6:          end if
7:       end if
8:    end for
9:    output $\mathcal{U}$
```

This induces the following parent-child relationship between clusters.

Definition 5 (Parent-child relationship) Let $\mathcal{U}^*$ be a cluster and $v \in \mathcal{V} \setminus \mathcal{U}^*$. $\mathcal{U} = \mathcal{U}^* \cup \{v\}$ is a child of $\mathcal{U}^*$ if and only if

$$\forall u \in \mathcal{U}^* \quad (\deg_{\mathcal{U}}(v) < \deg_{\mathcal{U}}(u)) \vee ((\deg_{\mathcal{U}}(v) = \deg_{\mathcal{U}}(u)) \wedge (v < u)).$$

It remains to show that this parent-child relationship guarantees the anti-monotonicity and cluster reachability during the search.

Lemma 1 *Let $\mathcal{U}$ be a non-trivial cluster and $\rho(\mathcal{U}) > 0$. Then for all $v \in \mathcal{U}$ with minimum degree, i.e.,*

$$v \in \operatorname*{argmin}_{u \in \mathcal{U}} \deg_{\mathcal{U}}(u),$$

the following properties hold:

1. $\text{size}(\mathcal{U} \setminus \{v\}) \geq 1$;
2. $\rho(\mathcal{U} \setminus \{v\}) \geq \rho(\mathcal{U})$.

Proof See Appendix A. □

The first statement of the lemma refers to the cluster reachability; it ensures that, by iterative application of the reduction map in Definition 4, any cluster with positive density shrinks to a trivial cluster, i.e., a root of the search space; that means, there do not occur degenerate constructs where some dimension-specific instance sets are empty. The second property implies anti-monotonicity, i.e., a parent cluster is at least as dense as any child cluster. Note that these properties hold for any minimum degree instance; however, to ensure that each cluster has a unique parent, the reduction map has to specify which of them is selected (in our case, the one with the smallest index). The definition of the reduction map directly suggests the following algorithm.

4.5 Enumeration algorithm

To enumerate all clusters in an n-dimensional data array W that satisfy a minimum density threshold θ, we perform the procedure shown in Algorithm 1, which is in the following referred to as DCE (dense cluster enumeration algorithm). The first step consists in finding trivial clusters that serve as roots for further expansion. These are simply the elements $\geq \theta$ of the data array. For each of them, we perform a depth-first traversal of the corresponding cluster tree, generating descendants of increasing cardinality according to the parent-child relationship specified in Definition 5. Note that this is done by reverse search, that means it involves the production of child candidates, among which the actual children are selected and further expanded. As soon as a cluster violates the density threshold, the current branch of the search tree is pruned. The correctness of the algorithm follows from Lemma 1.

4.6 Implementation details

To be able to deal with an arbitrary number of dimensions, the input is represented in a sparse format. For each non-zero entry, we create a data object that contains the n-dimensional index vector and the corresponding value. To facilitate the access to entries during the search, we generate for each $v \in \mathcal{V}$ a list of pointers to the objects containing v (also called *adjacency list* of v). For efficient checking of the density and the child conditions during the search, we maintain an array of length $|\mathcal{V}|$ that contains degree values with respect to the current cluster $\mathcal{U}$. More specifically, it contains for $v \in \mathcal{U}$ the value $\deg_{\mathcal{U}}(v)$, and for $v \in \mathcal{V} \setminus \mathcal{U}$ the value $\deg_{\mathcal{U}\cup\{v\}}(v)$. In some cases, it is possible to decide in constant time whether the parent-child relationship holds, without updating the degree values of cluster instances and checking the conditions given in Definition 5. The following lemma states two simple rules; similar rules handling degree equalities are omitted for conciseness.

Lemma 2 *Given a cluster $\mathcal{U}$ in the data array W, let $u^* \in \mathcal{U}$ be the previously added instance and $v \in \mathcal{V} \setminus \mathcal{U}$.*

1. *If W is non-negative and $\deg_{\mathcal{U}\cup\{v\}}(v) < \deg_{\mathcal{U}}(u^*)$, then $\mathcal{U} \cup \{v\}$ is a child of $\mathcal{U}$.*
2. *Let $g_{\mathcal{U}}(u^*, v)$ be the number of elements that the u^*-slice gains by adding v to the cluster $\mathcal{U}$. If $\deg_{\mathcal{U}\cup\{v\}}(v) > \deg_{\mathcal{U}}(u^*) + g_{\mathcal{U}}(u^*, v)$, then $\mathcal{U} \cup \{v\}$ is not a child of $\mathcal{U}$.*

Proof See Appendix B. □

4.7 Complexity

Depending on the choice of the density parameter θ and the structure of the dataset, the dense cluster enumeration algorithm can produce a large number of solution patterns. Already the problem of finding dense subgraphs, which can be solved by the procedure described in Sect. 3, is hard (Uno 2007). As combinatorial enumeration problems can generally have an exponential number of solutions, a conventional complexity measure is the *delay*, which is defined as the computation time needed to reach the subsequent solution (Goldberg 1992). While straightforward enumeration strategies for dense cluster detection have exponential delay (Uno 2007), DCE is a polynomial-delay algorithm. To see this, let us first consider a single iteration of the subroutine DCE_Rec (see Algorithm 1), which corresponds to finding all children of a given cluster. In the implementation described above, this needs $O(|\mathcal{V}| \times (l \cdot n + |\mathcal{U}|))$ operations, where $\mathcal{V}$ is the global index set, l is the average length of an adjacency list, n is the number of dimensions, and $\mathcal{U}$ is the current cluster; for non-sparse input arrays, l is roughly $O(|\mathcal{V}|^{n-1})$. In the worst case, we go for each $v \in \mathcal{V} \setminus \mathcal{U}$ through the whole adjacency list in order to determine the updated degree values for the members of $\mathcal{U}$, and then check all conditions in Definition 5. In practice, the density check will already discard many candidates; furthermore, we can avoid the traversal of the adjacency list in the cases where Lemma 2 can be applied. Finally, when traversing an adjacency list, only objects that are relevant for the update have to be fully processed. Usually, the $l \cdot n$ term will dominate the $|\mathcal{U}|$ term, so the complexity of one iteration is approximately linear in the input size (i.e., the adjacency list representation of the data array W).

Using data representations that allow for constant-time access to specific entries of W, the complexity of one iteration is given by

$$O\left(\sum_{i=1}^{n} |V_i \setminus U_i| \prod_{j=1,\, j\neq i}^{n} |U_j|\right) = O\left(\prod_{i=1}^{n} |V_i|\right),$$

which reflects the costs of traversing the entries of each slice that can be added to the cluster; here, V_i and U_i denote dimension-specific instance sets as introduced in Sect. 4.1. That means, also in this representation the complexity is linear in the size of the input tensor. If we apply the odd-even trick for solution output in recursive calls (Uno 2007), we obtain at least one solution during three executions of DCE_Rec. Thus, the delay between two consecutive solutions within the same search tree is in the order of the input size. Furthermore, the traversal of irrelevant entries between two successful search trees (i.e., the time between two calls of DCE_Rec from the routine DCE in Algorithm 1) is bounded by the size of the input array. Consequently, the proposed method needs polynomial delay to enumerate dense clusters in an n-dimensional array. As different search trees or different branches of the same search tree can be investigated independently, distributed computation is possible. Concerning memory, the described implementation has moderate requirements: it stores the input data and in addition needs $O(|\mathcal{V}|)$ space for each recursive step; the maximum recursion depth is equal to $|\mathcal{U}_{\max}| - n + 1$, where $\mathcal{U}_{\max}$ is the solution cluster with the largest cardinality.

5 Extensions

Now we treat several extensions that aim at facilitating the cluster analysis; in particular, we define additional criteria to direct the search or to filter the output.

5.1 Locally maximal clusters

Usually, the user is not interested in clusters that are subsets of other cluster solutions. A straightforward approach to tackle this problem would be to go for each newly detected cluster through all previous solutions, checking for inclusions. However, the structure of our reverse search algorithm allows us to reduce the number of solutions in the output in a meaningful way without any additional costs. We can set a flag that indicates whether there exists a direct extension (i.e., a cluster with one additional instance) that also satisfies the minimum density threshold. If that is the case, we do not output the current cluster, otherwise we do. This yields us the set of all *locally maximal* clusters. If the density parameter θ is equal to 1, a locally maximal cluster is also maximal, i.e., it is not contained in any other solution.

Trivial clusters (i.e., single-element clusters) are excluded by default. In addition to the density, the output filtering can take minimum degree constraints into account; in that case, the output consists of all clusters $\mathcal{U}$ such that $\rho_W(\mathcal{U}) \geq \theta$, $\min_{u \in \mathcal{U}} \deg_{\mathcal{U}}(u) > t$, $\text{size}(\mathcal{U}) > 1$, and for all $v \in \mathcal{V} \setminus \mathcal{U} : \rho_W(\mathcal{U} \cup \{v\}) < \theta \ \vee \ \min_{u \in \mathcal{U} \cup \{v\}} \deg_{\mathcal{U} \cup \{v\}}(u) \leq t$. By default, we set $t = 0$, i.e., all instances in a solution pattern must have positive degree values.

5.2 Balance constraints

While our density parameter controls the overall weight average across the whole cluster, in many applications it might be desirable to ensure a certain balance between the contributions of individual instances. In the previous section, we mentioned the possibility of minimum degree constraints. Beyond that, we can look at relative degree values, i.e., the ratio of the actual degree and the maximum possible degree. This criterion has the advantage that the threshold can be specified independently of the cluster size, whereas in the other case, a large threshold a priori excludes smaller solution clusters. In the following, we introduce a balance criterion that is based on the minimum relative degree.

Definition 6 (Balanced cluster) Given a threshold $\theta > 0$, a cluster $\mathcal{U}$ is called balanced if for each $u \in \mathcal{U}$, $\deg_{\mathcal{U}}(u)/\text{size}(\mathcal{U}, u) \geq \theta$. Here, $\text{size}(\mathcal{U}, u)$ denotes the size of the u-slice of $\mathcal{U}$.

The relative degree of an instance can be seen as the density of the corresponding cluster slice. Clearly, a balanced cluster is also a dense cluster with respect to θ, but not vice versa. For the specific case of a binary-valued symmetric input matrix, i.e., a graph, the balance criterion is equivalent to the concept of quasi-cliques (Zeng et al. 2006): a γ-quasi-clique is defined as a set of nodes U such that each of them has edges to at least $\lceil \gamma(|U| - 1) \rceil$ other nodes in U. As our implementation stores the degree of each instance with respect to the current cluster, it is straightforward to integrate a filtering step that checks the balance criterion. If the input array contains non-binary weights, it can also make sense to define separate criteria for the relative number of observed entries of a slice or a cluster and for its density; furthermore, one could constrain the density of lower-order slices or fibers of a cluster.

While the described filtering step excludes imbalanced clusters from the results, it does not contribute to speed up the search. Unfortunately, our search framework cannot be directly applied to the balance criterion. Indeed, there does not exist a reduction map that yields anti-monotonicity with respect to the minimum slice density in a cluster. This is illustrated by the following counter example:

	b_1	b_2	b_3
a_1	0	1	1
a_2	1	0	1
a_3	1	1	0

In the shown two-way cluster, each slice has density $\frac{2}{3}$. But, no matter which instance is removed, the minimum slice density in the reduced cluster is shrinking to $\frac{1}{2}$. However, in conjunction with additional criteria like size constraints for the cluster (Sect. 5.5), it is conceivable to define balance-based pruning rules.

5.3 Cluster ranking

In spite of using additional criteria to filter the output, the result set might get large. Consequently, it is important to provide a meaningful ranking criterion for clusters. Due to the exhaustiveness of our approach, it is possible to calculate for each cluster an *exact p-value*, i.e., the probability that random selection produces a cluster with at least the same density (Bejerano et al. 2004). For a cluster $U = (U_1, \dots, U_n)$ from a data array $W = (w_{k_1,\dots,k_n})_{k_i \in V_i,\, i=1,\dots,n}$, the exact p-value is given by

$$\frac{|\{U' = (U'_1, \dots, U'_n) : \rho_W(U') \geq \rho_W(U) \wedge \forall i = 1, \dots, n : (U'_i \subset V_i \wedge |U'_i| = |U_i|)\}|}{\prod_{i=1}^{n} \binom{|V_i|}{|U_i|}}.$$

To obtain the numerator, we have to keep track of the densities and the dimension-specific set cardinalities for all the solutions we pass during the execution of the algorithm (not only the locally maximal ones). This can be done by storing for each occurring combination of cardinalities an individual list of density values. Then we group the output clusters according to their dimension-specific cardinalities. For each group, we sort the corresponding density list and traverse it once to obtain the p-values for the output clusters. Finally, we sort the whole set of output clusters according to their p-values, where the lowest p-values are ranked first. By this, we can compare the statistical significance of patterns without having to rely on simplified data models (Koyutürk et al. 2007) or to perform multiple randomization tests.

5.4 Isolation-based pruning

The basic cluster density criterion we use (Definition 4) does not include any restrictions regarding the associativity of individual member instances. For example, a dense subgraph as defined in Sect. 3 is not necessarily connected. In particular, larger clusters can tolerate instances that have a degree of zero; in the following, such instances are called *isolated*. In principle, if the input array contains negative weights, instances can also have degree values below zero; however, for simplicity, we focus here on the case of non-negative input arrays. Usually, clusters with isolated instances are not desired as outputs; we eliminate them by a simple degree-based filtering step, as mentioned in Sect. 5.1. Nevertheless, the search procedure has to include such clusters in order to guarantee the completeness of the solution set. Indeed, a connected cluster can descend from a cluster that contains an isolated instance:

	b_1	b_2
a_1	1.0	0
a_2	0	0.9
a_3	0.4	0.4

This two-way cluster corresponds to a connected bipartite graph. Assuming a lexicographical order for the instances, the cluster is reduced as follows: $\{a_1, a_2, a_3, b_1, b_2\} \rightarrow \{a_1, a_2, b_1, b_2\} \rightarrow \{a_1, b_1, b_2\}$. In the latter cluster, b_2 is isolated. Similar examples can also be constructed for binary-valued data arrays.

However, the ancestors of connected clusters cannot contain an arbitrary number of isolated instances:

Lemma 3 *Let us consider a non-negative n-dimensional data array. If a cluster contains only non-isolated instances, its ancestors have at most one isolated instance per dimension.*

Proof See Appendix C. □

That means, a search tree can be pruned if the current cluster contains two isolated instances that are members of the same dimension. Similarly, one can show that clusters with at least one zero-degree instance in each dimension cannot have a descendant that contains only non-isolated instances. Note that the non-existence of isolated instances does not imply the connectivity of clusters in a graph-theoretic sense; they could in principle consist of several connected components. However, in practice this happens only for low density thresholds or large clusters.

5.5 Size and branching restrictions

Sometimes it might be useful to restrict the cluster size by pre-defining an upper limit for the number of instances in each dimension. As our search strategy extends the clusters by one instance of one particular dimension in each step, it can naturally incorporate such cardinality constraints and prune as soon as they are violated. On the other hand, minimum cardinality constraints are popular to focus on significant results. They can be exploited for discarding low-degree instances.

Due to the completeness of the dense cluster enumeration, the method might produce a large number of overlapping clusters, in particular it will visit all dense subclusters of a large dense cluster. A simple way to control the generation of similar clusters during the search is to restrict the maximum number of children (branches) per cluster to k. By this, the number of clusters sharing the same ancestor is limited, and decreases with increasing cardinality of the ancestor. Of course, this leads to the loss of the completeness guarantee. However, if we carefully select in each step the most promising children, we are likely to obtain in the end a substantial fraction of the most significant clusters. For that purpose, we propose the following procedure. Among all instances v that produce children $\mathcal{U} \cup \{v\}$ of the current cluster $\mathcal{U}$, we select the k instances with the largest degree. The motivation behind this is that they are most likely to have dense descendants. Among instances with equal degree, we prefer those with the largest (global) indices because according to our reduction map, they are removed last if there exist equally qualified instances.

Further possibilities to restrict the search include the selection of starting entries (i.e., roots of search trees) and the explicit control of instance reusage in different clusters. Apart from these heuristic restrictions, we can exploit external data sources by defining appropriate constraints (Georgii et al. 2009a). Finally, symmetry structure in the data can impose specific requirements on the cluster analysis, which are explained in detail in the next section.

6 Symmetry adaptations

So far, we considered multi-way data with distinct instance sets in each dimension. Here, we discuss how to deal with partial symmetries in the input data and include cluster symmetry constraints. This extension of the dense cluster enumeration formalism will restore the dense subgraph mining task from Sect. 3 as a special case. As a motivating example for the multi-way setting, let us consider a set of weighted undirected networks that share the same set of nodes. They can be represented by a three-way tensor where an entry w_{ijk} corresponds to the weight of the edge between the ith and the jth node in the kth network. It has the following characteristics: (a) the first two dimensions contain identical instance sets (i.e., the same *global* indices); (b) as the networks are undirected, the entries w_{ijk} and w_{jik} are equivalent; we say that the tensor is *symmetric* with respect to the first two dimensions; (c) "diagonal" entries w_{iik} correspond to self-edges. Now we are interested in finding subsets of nodes that induce dense subgraphs in a subset of networks. We can tackle this problem in the tensor framework by extracting dense three-way clusters that have identical instance sets in the first two dimensions; i.e., the clusters are symmetric with respect to these dimensions. In analogy to Sect. 3, the density criterion ignores self-edges (loops). In the following, we introduce definitions for multi-way clusters that respect symmetry constraints.

6.1 Definitions

Our dense cluster enumeration framework is suitable to handle arbitrary symmetry relationships among dimensions; there may exist multiple symmetry groups of different size. For instance, a six-way tensor could be symmetric with respect to dimensions 1 and 2, and also symmetric with respect to dimensions 3, 4, and 6. To keep the notation simple, we illustrate the concept for the case where we have symmetry with respect to the first j dimensions ($j \le n$) and all other dimensions are not involved in symmetry relationships. That means, given a tensor entry $w_{k_1,\dots,k_n}$ with distinct indices $k_1,\dots,k_j$, all entries that can be obtained by permutation of the first j indices are equivalent, so it is sufficient to store only one of the $j!$ possibilities. Then, a cluster $U = (U_1,\dots,U_n)$ is called symmetric (with respect to the first j dimensions) if $U_1 = \cdots = U_j$. Its size is given by

$$\mathrm{size}_j(U) = \binom{|U_1|}{j} \prod_{i=j+1}^{n} |U_i|, \tag{6}$$

and its density is calculated as follows:

$$\rho_{W,j}(U) = \frac{1}{\mathrm{size}_j(U)} \sum_{k_i \in U_i,\, k_1 < \cdots < k_j} w_{k_1,\dots,k_n}. \tag{7}$$

Like in Sect. 3, we count equivalent entries only once and we do not take self-relationships into account (i.e., we only consider entries with distinct indices $k_1,\dots,k_j$). This leads us to a modified definition for the degree of an instance $u \in U_l$:

$$\deg_U(u,l) = \begin{cases} \sum_{k_i \in U_i,\, k_l = u,\, k_1 < \cdots < k_j} w_{k_1,\dots,k_n} & \text{if } l > j, \\ \sum_{k_i \in U_i,\, u \in \{k_1,\dots,k_j\},\, k_1 < \cdots < k_j} w_{k_1,\dots,k_n} & \text{if } l \le j. \end{cases} \tag{8}$$

Now, reduction is again performed by removing the minimum degree instance that has the smallest global index. In analogy to Lemma 1, it can be shown that this reduction map is valid, i.e., it provides anti-monotonicity and cluster reachability.

6.2 Remarks

To perform the dense cluster enumeration, we can use similar data structures and speed-up rules as described in Sect. 4.6. However, we have to adapt the update procedures as well as the computation of $g_U(u^*, v)$, which is the number of new entries in the u^*-slice of the cluster U after adding v. Here, we consider the general case, allowing for an arbitrary number of symmetry groups among the n dimensions. In contrast to the setting without symmetry, an instance v may belong to several dimensions, namely all dimensions involved in the same symmetry relation. For convenience, we define $d(v)$ to be the smallest dimension v belongs to (the representative dimension of each symmetry group). Further, let s_i denote the total number of dimensions belonging to the same symmetry group as the ith dimension, i.e., $s_i > 1$ if the ith dimension has a symmetry relationship with respect to other dimensions, and $s_i = 1$ otherwise. Then, $g_U(u^*, v) = r_U(u^*, v) \cdot \text{size(U)}$, where size(U) is the number of distinct cluster entries (i.e., excluding entries that are equivalent due to symmetry) and

$$r_U(u^*, v) = \begin{cases} \frac{1}{|U_{d(v)}|\cdot|U_{d(u^*)}|} & \text{if } d(v) \neq d(u^*) \wedge s_{d(v)} = 1 \wedge s_{d(u^*)} = 1, \\ \frac{s_{d(u^*)}}{|U_{d(v)}|\cdot|U_{d(u^*)}|} & \text{if } d(v) \neq d(u^*) \wedge s_{d(v)} = 1 \wedge s_{d(u^*)} > 1, \\ \frac{s_{d(v)}}{(|U_{d(v)}|-s_{d(v)}+1)\cdot|U_{d(u^*)}|} & \text{if } d(v) \neq d(u^*) \wedge s_{d(v)} > 1 \wedge s_{d(u^*)} = 1, \\ \frac{s_{d(v)}s_{d(u^*)}}{(|U_{d(v)}|-s_{d(v)}+1)\cdot|U_{d(u^*)}|} & \text{if } d(v) \neq d(u^*) \wedge s_{d(v)} > 1 \wedge s_{d(u^*)} > 1, \\ \frac{s_{d(v)}(s_{d(v)}-1)}{|U_{d(v)}|\cdot(|U_{d(v)}|-s_{d(v)}+1)} & \text{if } d(v) = d(u^*) \wedge s_{d(v)} > 1, \\ 0 & \text{otherwise.} \end{cases}$$

These equations follow directly from the definition of $g_U(u^*, v)$. The extensions described in Sect. 5 are, with small modifications, also applicable in the case of symmetry constraints; the difference is that dimensions involved in symmetry relationships cannot be treated separately anymore, but have to be considered simultaneously. This affects for instance the computation of the ranking criterion and the isolation-based pruning rules.

In this section, we discussed how to extend our enumeration method to extract partially symmetric clusters from partially symmetric data. Apart from that, other application scenarios are conceivable, which can be solved similarly. For example, searching for asymmetric clusters in symmetric tensors can be meaningful if there exist groups of instances with strong inter-group connections, but not necessarily strong inner-group connections. On the other hand, one might be interested in clusters that contain the same set of instances in several dimensions although the data are not symmetric.

7 Experimental results

We implemented our DCE algorithm in C++[1]. In the following, we describe our experiments on synthetic and real-world datasets. If not indicated otherwise, we applied DCE with the output filtering explained in Sect. 5.1.

7.1 Scalability experiments

First we tested the performance of our method on artificial datasets. For that purpose, we generated sparse tensors with hidden clusters. For simplicity, we used binary values,

[1] Implementation available at http://www.kyb.tuebingen.mpg.de/~georgii/dce.html.

i.e., each tensor element is either 0 or 1. Let n be the number of dimensions, and m the number of clusters. Furthermore, we assumed a hypercubic model where each dimension has the same index set size d and each hidden cluster contains exactly s instances in each dimension. The clusters were allowed to overlap without any restriction. In addition, we imposed different levels of noise onto the data. Here, the noise corresponds to random 0–1 flips. Initially, all tensor elements within clusters were set to 1. Given a noise level p, we randomly selected $p\%$ of all 1-elements and the same number of 0-elements. Then the selected elements were flipped, i.e., the 1-elements were set to 0 and vice versa.

Given this model for data generation, we investigated the scalability of DCE with respect to the different model parameters d, m, n, and s. Our basic setting was $d = 100$, $m = 20$, $n = 3$, and $s = 3$; this corresponds to a total of 540 non-zero associations (1-elements) among 300 different instances (from 3 dimensions). Starting from that, we did four series of experiments, varying one of the parameters while keeping the others fixed. The maximum number of 1-elements was 540, 810, 4860, and 2500 in the four series, respectively; the maximum number of instances was 450, 300, 500, and 300, respectively. For each parameter configuration, we generated ten random datasets and considered noise levels from 0% to 30%. The density threshold for the DCE algorithm was chosen in dependence of the noise level, $\theta = (100 - p)\%$. Figure 4 shows the resulting runtime curves for each parameter. In addition to the total runtime, we report the empirical delay, i.e., the average runtime per solution. The values are averages across the ten random datasets. All measurements were performed on a 3.0 GHz machine.

DCE scales favorably with d, the number of instances per dimension (Fig. 4a). For noise levels from 0% to 20%, the runtime remains approximately constant with increasing d, for 30% noise it increases linearly. In the 30% noise case, the noise elements cover more instances and, due to the lowered density threshold, there exist more cluster solutions; consequently, the 30% noise curve depends much stronger on d than the other curves. The delay shows a linear increase in all cases. Regarding the number of hidden clusters, the total runtime increases linearly at all noise levels (Fig. 4b). The delay increases only very slightly; in fact, the higher number of clusters makes cluster overlaps more likely, so some instances may have longer adjacency lists, which can lead to an increased computational effort per solution. Actually, the 0% curve is on top because the cluster instances have longer adjacency lists than in the noisy cases. The number of subcluster solutions per hidden cluster increases exponentially with the number of dimensions n, which is reflected by the total runtime (Fig. 4c). In contrast, the increase of the delay is much more moderate. Likewise, when increasing the number of instances per dimension in the hidden clusters, the delay grows very slowly, although the total runtime of DCE increases significantly (Fig. 4d). Again, this can be explained by the increased number of subcluster solutions. Also, the effect is much stronger for higher noise levels.

To conclude, DCE is appropriate for finding dense clusters in sparse settings; however, the number of solutions may grow considerably with the dimensionality and the cardinality of the clusters. Remarkably, the computational effort per solution scales very well in the latter case, even though we use the adjacency list representation of the tensor and therefore do not have constant-time access to elements. This indicates that our methods to speed up the search process are effective. Furthermore, it encourages to combine the DCE search with heuristics that restrict the generation of overlapping clusters while trying to catch the most significant ones, which is investigated in the next section.

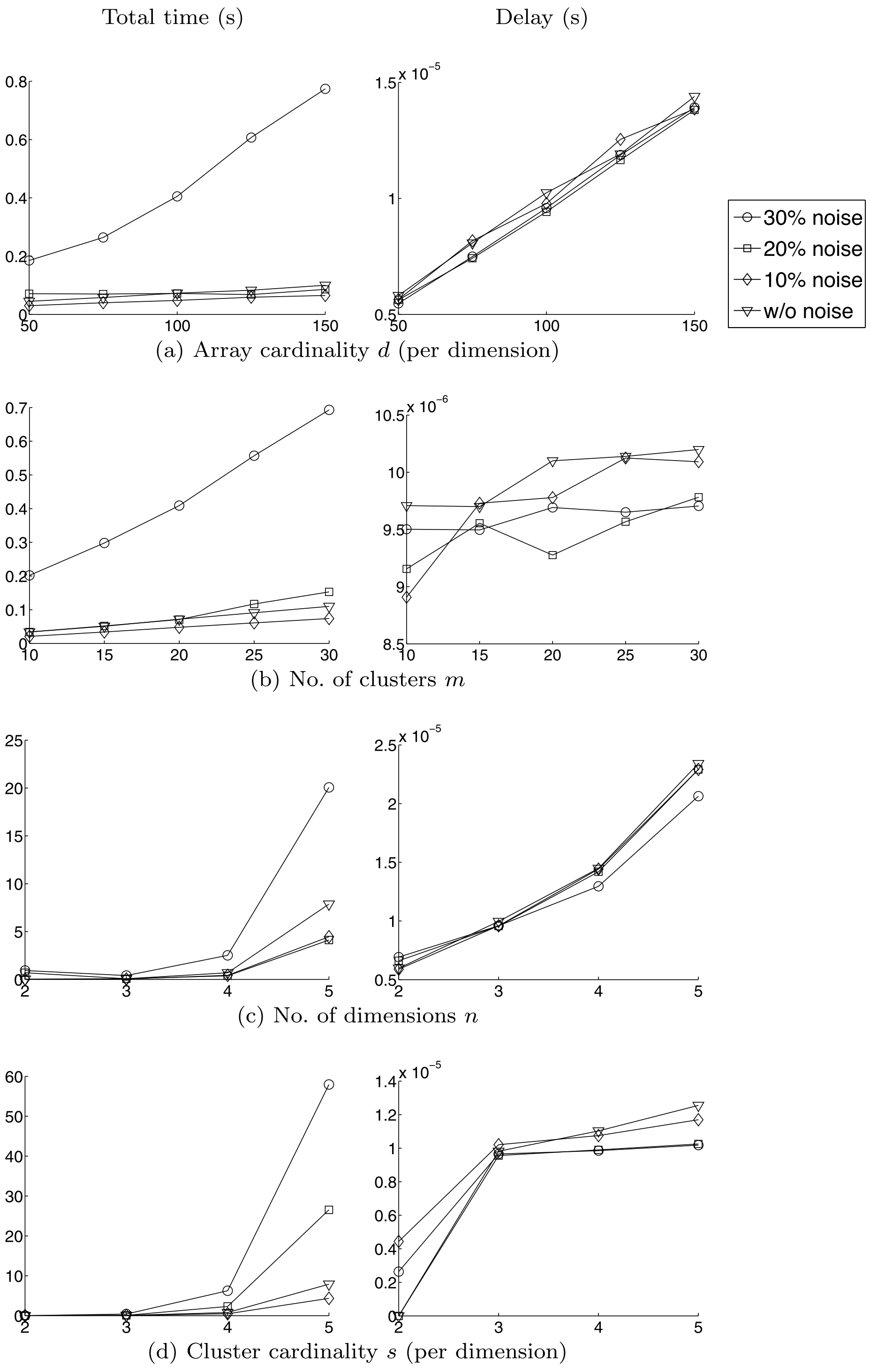


Fig. 4 DCE runtime measurements for artificial data in dependence of different parameters

7.2 Evaluation of branching-restricted search

In the following, we used the search strategy proposed in Sect. 5.5. The idea is to control the number of branches descending from a cluster. That means, in each iteration of the algorithm, we select the k most promising children (if available). Obviously, this restriction leads to a loss of the completeness property. We analyzed the behavior of DCE for different values of k in the context of the artificial datasets from the previous section. Table 1 shows our results for varying dimension-wise cluster cardinality s at noise levels 0% and 30%. We used the same datasets as for Fig. 4d and compared k-values from 1 to 4 with the unrestricted (complete) DCE version. As can be seen, the overall runtime was drastically reduced by introducing the branching restriction, in particular in the high noise case.

In order to evaluate the quality of the results, we used the following precision and recall measures. The precision is given by the fraction of hidden cluster elements among all DCE cluster elements, and the recall is given by the fraction of DCE cluster elements among all hidden cluster elements. Formally, let D be the set of all tensor elements that belong to clusters detected by DCE, and let H be the set of all tensor elements that belong to hidden clusters. Then, recall and precision are defined as follows:

$$\text{Recall} = \frac{|D \cap H|}{|H|},$$

$$\text{Precision} = \frac{|D \cap H|}{|D|}.$$

The averages across the ten random datasets were determined by

$$\text{Recall}_{\text{avg}} = \frac{\sum_{i=1}^{10} |D_i \cap H_i|}{\sum_{i=1}^{10} |H_i|},$$

$$\text{Precision}_{\text{avg}} = \frac{\sum_{i=1}^{10} |D_i \cap H_i|}{\sum_{i=1}^{10} |D_i|}.$$

Note that in our experiments, all hidden clusters had the same number of elements ($|H_i| = s^n$). Since we knew the size of our hidden clusters, we also evaluated recall and precision based on the predicted clusters that have at least this size (i.e., at least s instances in each dimension); these clusters are called *size-restricted*. In addition, we report recall and precision of the results satisfying the balance criterion described in Sect. 5.2.

Trivially, DCE achieved perfect precision and recall for 0% noise (see Table 1). This still held true if the branching was controled. But if we restricted the analysis to results satisfying the size constraint, we lost recall in some cases. However, the recall level was still quite high, and it was perfect for $k = 4$. Furthermore, for 0% noise (density threshold 100%), any predicted cluster trivially satisfies the balance criterion. In contrast, the balance constraint made a big difference in the 30% noise case. While we obtained 100% recall and approximately 29.84% precision for $s = 5$ using the unconstrained DCE algorithm, the balanced DCE clusters achieved 98.57% recall and 91.58% precision. Larger hidden clusters (i.e., higher values of s) generally led to higher recall and lower preci-

Table 1 Performance analysis of DCE and its extensions for artificial data with varying dimension-wise cluster cardinality s at 0% and 30% noise. For each setting, we took the average across ten random datasets. The parameter k refers to the optional branching restriction. For noise level 0%, the clusters are trivially balanced. See text for details

		Noise level 0%				Noise level 30%			
		$s=2$	$s=3$	$s=4$	$s=5$	$s=2$	$s=3$	$s=4$	$s=5$
Time (s)	–	0.00	0.07	0.79	7.91	0.00	0.41	6.26	58.01
	$k=4$	0.00	0.07	0.74	6.83	0.00	0.06	0.63	5.08
	$k=3$	0.00	0.07	0.62	5.10	0.00	0.05	0.47	3.17
	$k=2$	0.00	0.05	0.36	2.22	0.00	0.04	0.24	1.17
	$k=1$	0.00	0.02	0.08	0.26	0.00	0.02	0.06	0.16
Recall (%)	–	100.00	100.00	100.00	100.00	88.19	98.91	99.96	100.00
	$k=4$	100.00	100.00	100.00	100.00	88.19	98.89	99.94	100.00
	$k=3$	100.00	100.00	100.00	100.00	88.19	98.78	99.90	100.00
	$k=2$	100.00	100.00	100.00	100.00	88.19	98.28	99.80	100.00
	$k=1$	100.00	100.00	100.00	100.00	84.81	95.24	98.41	99.78
Precision (%)	–	100.00	100.00	100.00	100.00	95.21	80.26	54.28	29.84
	$k=4$	100.00	100.00	100.00	100.00	95.21	82.02	58.37	34.45
	$k=3$	100.00	100.00	100.00	100.00	95.21	83.73	61.25	38.75
	$k=2$	100.00	100.00	100.00	100.00	95.53	86.60	69.69	51.47
	$k=1$	100.00	100.00	100.00	100.00	97.07	94.07	89.07	84.67
Recall for	–	100.00	100.00	100.00	100.00	57.00	59.50	54.03	48.51
size-restricted	$k=4$	100.00	100.00	100.00	100.00	57.00	59.50	54.03	48.51
clusters (%)	$k=3$	100.00	100.00	99.50	100.00	57.00	59.50	53.53	48.51
	$k=2$	100.00	100.00	99.00	99.00	57.00	59.00	48.52	44.01
	$k=1$	100.00	99.52	97.50	98.00	56.50	39.01	22.01	10.50
Precision for	–	100.00	100.00	100.00	100.00	99.35	99.72	100.00	100.00
size-restricted	$k=4$	100.00	100.00	100.00	100.00	99.35	99.72	100.00	100.00
clusters (%)	$k=3$	100.00	100.00	100.00	100.00	99.35	99.72	100.00	100.00
	$k=2$	100.00	100.00	100.00	100.00	99.35	99.72	100.00	100.00
	$k=1$	100.00	100.00	100.00	100.00	99.67	100.00	100.00	100.00
Recall for	–		See above			71.94	84.62	94.14	98.57
balanced	$k=4$					71.94	84.55	93.93	98.41
clusters (%)	$k=3$					71.94	84.49	93.70	98.15
	$k=2$					71.81	83.99	92.42	97.07
	$k=1$					69.87	78.42	84.43	89.02
Precision for	–					98.38	96.72	94.31	91.58
balanced	$k=4$					98.38	96.72	94.32	91.61
clusters (%)	$k=3$					98.38	96.71	94.33	91.67
	$k=2$					98.37	96.72	94.38	91.73
	$k=1$					98.42	96.58	94.08	91.27
Recall for	–					27.50	1.50	0.00	0.50
balanced	$k=4$					27.50	1.50	0.00	0.50
size-restricted	$k=3$					27.50	1.50	0.00	0.50
clusters (%)	$k=2$					27.50	1.50	0.00	0.50
	$k=1$					27.50	1.50	0.00	0.50

Table 1 (*Continued*)

		Noise level 0%				Noise level 30%			
		$s=2$	$s=3$	$s=4$	$s=5$	$s=2$	$s=3$	$s=4$	$s=5$
Precision for	–					100.00	100.00	–	100.00
balanced	$k=4$					100.00	100.00	–	100.00
size-restricted	$k=3$					100.00	100.00	–	100.00
clusters (%)	$k=2$					100.00	100.00	–	100.00
	$k=1$					100.00	100.00	–	100.00

sion in the noisy case because they allow for more cluster variants. In conjunction with the minimum size constraint, DCE produced 48.51% recall at 100% precision (in the $s=5$ setting). Note that DCE would always recover 100% of the hidden clusters if we applied during the data generation the same fraction of flips to each individual cluster. In our experiments, we just fixed the overall fraction of flips across all clusters, which is a more realistic setting. After adding 30% noise, only a small fraction of the hidden clusters satisfied both size and balance constraints; the average recall was 0.5% for $s=5$ and 27.5% for $s=2$.

Regarding the influence of the parameter k, our empirical results suggest the following tendencies. Naturally, the recall increased for higher values of k. Remarkably, for $k=3$ or $k=4$ the recall was in most cases very close or equal to the recall obtained without branching constraints, although the runtime was significantly reduced. Sometimes, the recall level was already reached with $k=2$. The precision of DCE clusters was higher for small k, which indicates that the heuristics is indeed successful in focusing on significant solutions. With respect to size or balance constraints, the precision remained approximately the same for all k and the unrestricted branching. In summary, the branching restriction is an effective technique to speed up the search procedure while maintaining the precision and recall levels of the complete algorithm.

7.3 Efficiency of reverse search

In order to investigate the efficiency of the reverse search approach, we compared its performance with other set enumeration strategies. Here, we restricted our studies to two-dimensional binary-valued data arrays. On the one hand, we downloaded gene signatures for mouse and rat from GeneSigDB (Culhane et al. 2010); they contain a set of references to biological experiments, each of which is associated with a list of genes. Combining this information in a data matrix allows to do a meta-analysis on results from different biological publications. The size of the data is 122×917 and 12×182 for mouse and rat, respectively; the densities of the whole datasets are 5% and 19%, respectively. On the other hand, we reused artificial data from Sect. 7.1 the previous section, with a size of 100×100 ($m=20$, $s=3$, noise level 30%) and an overall density of 2%.

For comparison with DCE, we implemented two straightforward set enumeration approaches, which we call BruteForce and BruteNeigh. They use exactly the same data structures for tensor access and incremental density calculation as DCE (see description in Sect. 4.6), the only difference is in the traversal of the search space. BruteForce enumerates all possible pairs of subsets with respect to the two dimensions; this is done by two nested loops, each of which constructs subsets incrementally by the aid of a lexicographical search tree. BruteNeigh is a more clever variant that is particularly suited for size-imbalanced datasets. For the dimension with fewer instances, it enumerates all subsets of instances in the same way as BruteForce; for each of these subsets, it exploits neighborhood relationships in order to determine the subsets in the other dimension. More precisely, if U_1 is the subset in the first dimension and V_2 is the total set of instances in the second dimension, it collects a set S of all instances in V_2 that are associated with at least one instance in U_1 (i.e., neighbors in the corresponding bipartite graph); then, all possible subsets of S are generated as before. This resembles mining approaches on bipartite graphs described in Tanay et al. (2002) and basically avoids clusters with isolated instances (Sect. 5.4). In addition, we looked at a combined version, BruteDCE, which enumerates all subsets in the smaller dimension and performs for each of them a separate reverse search on the corresponding neighbor set, in order to enumerate its dense subsets (leaving the subset in the first dimension fixed).

We applied the different search techniques on the three datasets, using maximum cardinality constraints for the dimension-specific instance sets (Sect. 5.5) in order to make the BruteForce approach feasible; the maximum cardinality (denoted as max) was set to the same value for both dimensions and ranged from 2 to 4. Figure 5 shows the resulting time curves for different density thresholds. The measurements were performed on a 2.8 GHz processor. We observed some general trends across the experiments. First of all, the neighborhood-based technique greatly payed off for straightforward set enumeration methods: BruteNeigh improved the performance of BruteForce by approximately one order of magnitude. Second, DCE showed an exponential runtime increase for decreasing density thresholds, but in the upper threshold range from 100% to 60% density, it outperformed BruteNeigh in most cases by a wide margin. Furthermore, considering fixed density thresholds, the performance gain of DCE increased with increasing levels of (maximum) cardinality. For density thresholds close to the overall density of the dataset, DCE was visibly worse than BruteForce (see rat plots); this is due to the overhead of the child generation process in the reverse search, which has to select the true children among all possible candidates (Sects. 4.4 and 4.5).

Regarding the behavior of BruteDCE, there were major differences between the datasets. While the combined enumeration strategy was beneficial for the highly size-imbalanced rat dataset, consistently achieving lower runtimes than DCE, it had mainly a negative effect for the artificial dataset, where the number of rows equals the number of columns. For the mouse dataset, it outperformed DCE only for medium and low density thresholds. Note that the shape of the BruteDCE curve differs heavily between rat and artificial experiments, resembling the DCE curve and the BruteNeigh curve, respectively. This indicates that in the latter case, the brute force enumeration part dominated the computation time of BruteDCE, whereas in the former case the reverse search part played the main role. Overall, the empirical results show that in comparison with straightforward search approaches, the reverse search strategy considerably improves the efficiency of dense cluster enumeration.

7.4 Email traffic analysis

We applied DCE to a subset of the ENRON email dataset (Klimt and Yang 2004) taken from Borgwardt et al. (2006). It records information about the sender, the recipients, and the

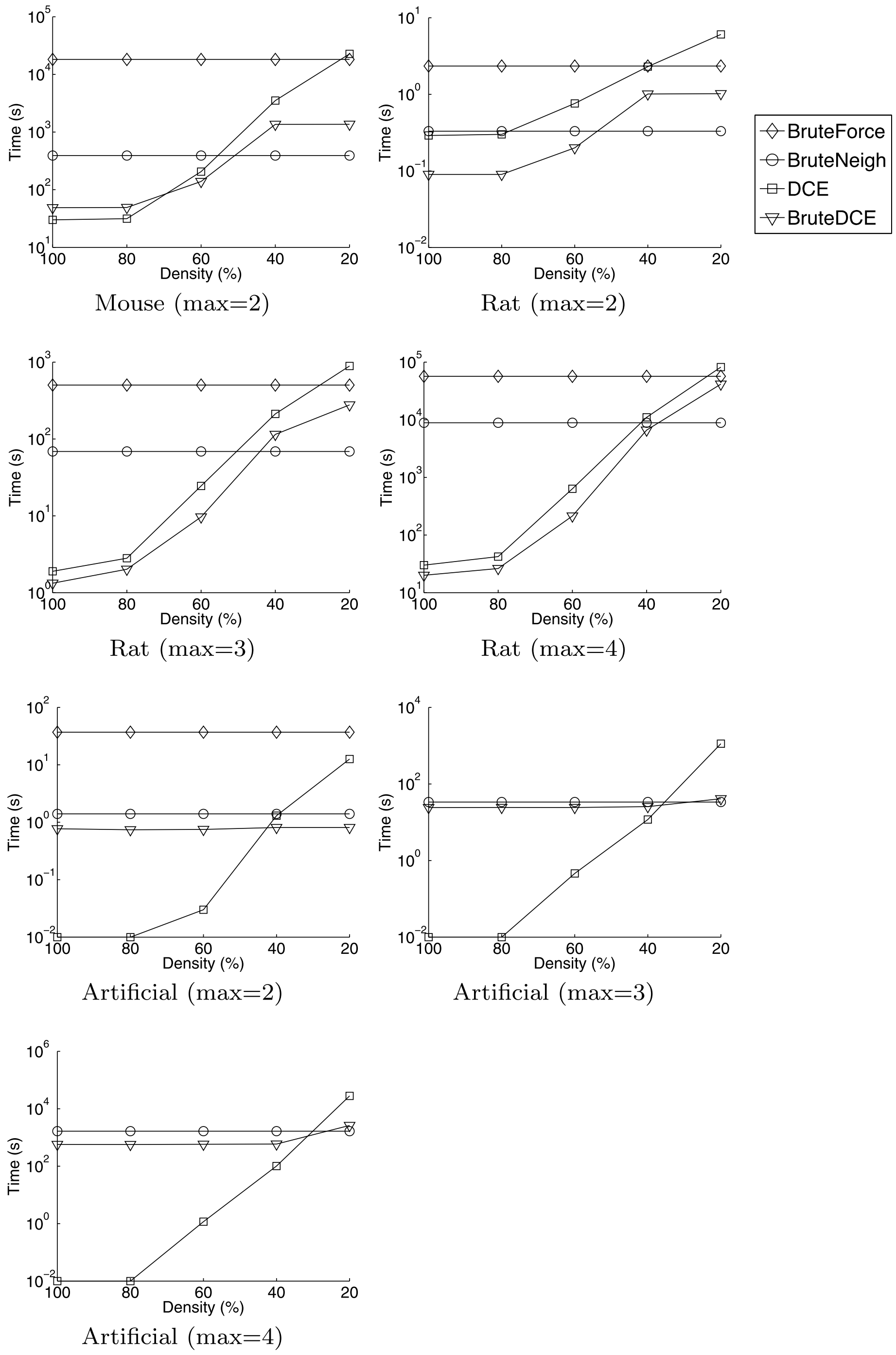


Fig. 5 Runtime comparison of DCE and other set enumeration strategies in dependence of the minimum density threshold. The time axis is logarithmic. See text for details

Senders:	155, 162, 169
Recipients:	114, 155, 165, 169
Weeks:	103, 108, 118, 120

Sender	Recipient	No. of emails in week			
		103	108	118	120
155	114	≥10	≥10	≥10	≥10
155	155	0	0	0	0
155	165	≥10	≥10	≥10	8
155	169	≥10	≥10	≥10	8
162	114	≥10	≥10	≥10	≥10
162	155	≥10	≥10	≥10	≥10
162	165	≥10	≥10	≥10	≥10
162	169	≥10	≥10	≥10	≥10
169	114	≥10	≥10	≥10	≥10
169	155	≥10	≥10	≥10	≥10
169	165	8	≥10	≥10	8
169	169	0	0	0	0

Fig. 6 Top-ranking cluster for email traffic data

time stamp of emails. From this, we generated a three-way tensor as follows. We mapped each time stamp to the corresponding week and then determined the number of emails a certain sender sent to a certain recipient in a certain week. This yielded a $120 \times 143 \times 123$ tensor with 6995 non-zero elements. We were interested in groups of persons that regularly exchange a large number of emails. The individual frequency values ranged from 1 to 202, however 81% of them were lower than or equal to 10; we set elements with values greater than 10 to 10, in order to avoid results that are dominated by one or very few outlier elements and consequently do not reliably describe associations between sets of instances. After the pre-processing, we ran DCE with a density threshold of 80%. That means, for a cluster solution, each sender sends in each week on average at least 8 emails to each recipient.

Restricting the maximum number of instances per dimension for each cluster to 4, we obtained approximately 3.5×10^7 clusters in total. This seems to be a large number of clusters, but it is reasonably small compared to the number of cluster candidates for the tensor at hand, which is 2.0×10^{22} for the given maximum size constraint. Focusing on locally maximal patterns with at least two instances in each dimension, the size of the result set shrinks to 240675, and among them, there are only 142 clusters with at least three instances in each dimension. The top-ranking cluster (density: 82%, p-value: 4.7×10^{-20}) is shown in Fig. 6. It contains three senders, four recipients and four weeks. Senders and recipients are given by personal identifiers. Two persons appear as both senders and recipients, one person appears only as sender, and two persons only as recipients. The only zero elements of the cluster are due to the absence of self-emails for the two persons in the overlap. This cluster remains the top-ranking cluster even if we drop the maximum cardinality constraints for senders and recipients, which means that there do not exist such dense clusters involving more people.

7.5 Coexpression analysis

To illustrate dense cluster enumeration in the case of partially symmetric data, we analyzed coexpression networks from multiple gene expression experiments in yeast.

7.5.1 Data

We took the gene expression dataset from Gasch et al. (2000) and pre-processed it similarly as described in Hu et al. (2005): first, we selected the experiments with at least 6 individual measurements; then we calculated for each experiment the Pearson correlation coefficients between the expression profiles of all genes; if the correlation was positive and significant at a level of 10^{-5}, we connected the corresponding genes by an edge (weight 1). This resulted in 17 different coexpression networks on the same set of genes; there were 6152 genes in total, and each network contained 9237 edges on average. The networks can be represented as a three-dimensional tensor with the genes in the first two dimensions and the identifier of the experiment (i.e., the network type) in the third dimension. As the networks are undirected, the tensor is symmetric with respect to the first two dimensions.

7.5.2 Experimental setup

Our goal was to analyze the set of networks for cooccurring dense substructures. There exist several competitive approaches that deal with sets of unweighted networks. We compared our DCE approach with Cocain (COC) (Zeng et al. 2006) and Codense (COD) (Hu et al. 2005). Cocain is an enumerative method that detects frequent quasi-cliques. This criterion is stricter than our criterion, even if we require balance constraints (Sect. 5.2); in fact, each frequent quasi-clique is a balanced dense cluster, but not vice versa. The Codense algorithm is a non-enumerative approach that searches for dense subgraphs in a summary network and then extracts subsets with correlated edge occurrence across the original networks. Relational data mining approaches (Ji et al. 2006) are equivalent to DCE with density threshold 100%, so we did not consider them separately in our biological evaluation. The COC code was obtained from the original authors (Zeng et al. 2006), and COD was downloaded from http://zhoulab.usc.edu/CODENSE/. The minimum network frequency and the minimum number of genes per cluster were set in all approaches to 3 and 6, respectively; there were 4745 frequent edges involving 411 nodes. The p-value threshold required by COD was set to 0.01.

7.5.3 Evaluation measures

To evaluate clusters of genes, we performed a functional enrichment analysis with respect to Gene Ontology annotations. The Gene Ontology (Ashburner et al. 2000) provides a framework for functional categorization of genes; it is organized in a hierarchical way. We used the Expander tool (Shamir et al. 2005) to detect functional categories that are significantly overrepresented in the predicted clusters; for this, we used the default parameters with a p-value threshold of 0.05 after correction for multiple testing. In addition to the number of functionally enriched clusters, we report the average gene-wise reliability, the average pair-wise reliability, as well as the overall precision and recall. These measures are defined as follows. Given a cluster with one or several significantly enriched functional categories, genes that belong to the same enriched category are called *homogeneous*. Let hg_i be the size of the largest group of homogeneous genes in the ith cluster, and let g_i be the total number of genes in the cluster. Then, the gene-wise reliability of the cluster is given by

$$\frac{\mathrm{hg}_i}{g_i}.$$

Further, let hgp_i be the number of homogeneous gene pairs and gp_i the total number of gene pairs. The pair-wise reliability of the cluster is defined as

$$\frac{\mathrm{hgp}_i}{\mathrm{gp}_i}.$$

Compared to the gene-wise reliability, this measure takes into account all different enriched categories of a cluster. It can be seen as the probability that an arbitrary gene pair taken from the cluster is homogeneous. For each of the two reliability measures, we determine the average across all clusters. Finally, the precision and recall measures refer to unique (homogeneous) gene pairs across all clusters. That means, each gene pair is only counted once even if it occurs in more than one predicted cluster. Note that all methods applied in this comparison predict overlapping clusters. We report the number of homogeneous pairs relative to the number of all within-cluster pairs (precision) as well as the absolute number of recalled homogeneous pairs.

7.5.4 Results

Table 2 summarizes the results of DCE, COC, and COD for different density thresholds. For DCE, we also list the results with balance constraints (bal.) and with branching restrictions ($k = 1, 2$). These constrained DCE versions do not allow to do the local maximality check described in Sect. 5.1; instead, they return all clusters at leaf nodes of the search trees, which can produce larger cluster numbers.

For 100% density, DCE, DCE (bal.), and COC are all equivalent and therefore yield the same results. However, for lower density values DCE and DCE (bal.) are more flexible than the quasi-clique approach used by COC, so they achieve much higher recall, while precision and reliability remain in a comparable range. Interestingly, both for DCE and COC, the average cluster reliability with density threshold 85% is larger than with density threshold 100%. This can be explained by the fact that, at sufficiently high density levels, larger clusters are more likely to be biologically significant than small ones (note that the average number of genes per cluster is increased). On the other hand, a decreasing density threshold allows the clusters to include less related genes. Therefore, the overall precision of DCE was slightly reduced when going from 100% to 85% density. In contrast, COC, which applies the more rigid quasi-clique criterion, kept the precision level. The criterion required by COD is generally more restrictive and its search is not exhaustive, so the recall is lower. However, while the precision and reliability values are perfect for a density threshold of 95%, they are considerably behind the other approaches at 85% density.

For density thresholds below 85%, the number of solutions returned by DCE increases drastically, which comes along with an exponential increase of the runtime (the measurements were performed on a 2.8 GHz processor). The reason for that is the increasing flexibility of patterns, which leads to strongly overlapping solutions. This is even the case if we consider balanced clusters. Clearly, such an overwhelming set of results is not suitable for user inspection (also, the Expander tool for enrichment analysis failed); therefore, further criteria to restrict the search are needed. For comparison purposes, we again used the heuristic branching restriction introduced in Sect. 5.5 with values 1 and 2. With this, the performance was competitive with COC and COD. The branching restriction produced lower recall than the complete search (considering balanced clusters in both cases), but it can still compete with the recall values achieved by COC and COD. Furthermore, although our cluster criterion is less restrictive than the criteria for COC or COD, the clusters were biologically meaningful, achieving similar levels of reliability and precision. Beside that, DCE

Table 2 Comparative evaluation on coexpression data. Abbreviations: max. (maximum), avg. (average), rel. (reliability), bal. (balanced). The parameter k refers to the optional branching restriction of DCE. See text for details

	Density (%)	No. of clusters	No. of enriched clusters	Max. no. of genes	Avg. no. of genes	Gene-wise rel. (%)	Pair-wise rel. (%)	Precision (%)	No. of recalled pairs	Time (s)
DCE	100	53	52	9	6.7	95.2	92.6	84.3	215	2.9
	95	239	238	11	7.8	95.9	93.1	84.2	388	5.4
	90	1057	1048	13	8.6	95.6	92.9	81.7	642	25.7
	85	3269	3240	16	10.7	96.3	94.1	82.6	1041	179.2
	80	16982	n/a	18	11.8	n/a	n/a	n/a	n/a	2245.0
	75	95869	n/a	20	13.9	n/a	n/a	n/a	n/a	30011.2
DCE (bal.)	100	53	52	9	6.7	95.2	92.6	84.3	215	2.9
	95	425	416	9	6.5	96.3	94.9	83.3	219	5.4
	90	1288	1277	11	6.6	97.5	96.0	81.9	303	25.6
	85	3705	3684	11	7.0	98.0	96.9	82.6	409	179.7
	80	10697	n/a	13	7.2	n/a	n/a	n/a	n/a	2271.8
	75	24200	n/a	14	8.3	n/a	n/a	n/a	n/a	29968.1
DCE (bal., k=1)	100	17	16	9	6.7	92.1	90.4	83.6	117	0.6
	95	17	16	9	6.8	91.4	88.8	82.5	118	0.6
	90	28	27	11	6.8	92.1	88.6	81.4	162	0.7
	85	38	37	11	7.0	94.0	91.4	82.9	194	0.7
	80	66	64	12	7.1	93.8	91.0	80.9	284	0.9
	75	71	69	14	8.0	94.9	92.9	82.2	332	1.0
DCE (bal., k=2)	100	133	130	9	6.7	95.7	94.0	83.0	176	1.1
	95	136	133	9	6.8	95.8	94.2	83.3	185	1.4
	90	296	291	11	6.9	95.9	94.1	83.9	260	2.3
	85	590	584	11	7.4	97.3	96.1	82.6	338	4.2
	80	1247	1237	13	7.7	97.1	95.4	81.9	456	9.8
	75	2198	2192	14	8.9	97.7	96.0	82.8	521	23.5
COC	100	53	52	9	6.7	95.2	92.6	84.3	215	1.3
	95	53	52	9	6.7	95.2	92.6	84.3	215	1.3
	90	53	52	9	6.7	95.2	92.6	84.3	215	2.3
	85	109	108	10	8.2	97.2	95.4	85.0	260	7.2
	80	200	199	12	7.6	96.3	93.4	83.3	329	14.0
	75	520	512	13	8.2	95.7	93.2	82.9	474	54.2
COD	100	0	–	–	–	–	–	–	–	0.2
	95	3	3	11	9.7	100.0	100.0	100.0	80	1.6
	90	10	9	11	7.5	90.7	91.7	83.6	107	1.5
	85	9	8	10	7.9	84.5	81.3	76.1	140	1.6
	80	10	9	18	8.9	85.4	82.8	79.0	245	2.0
	75	8	7	21	11.2	85.6	84.3	80.9	314	1.6

is applicable to more general settings, namely data with an arbitrary number of dimensions, binary or weighted values, including symmetries or not.

8 Discussion

We presented a general framework for the systematic extraction of dense patterns from higher-order association data. It extends conventional relational set mining approaches (Cerf et al. 2008; Jaschke et al. 2006; Ji et al. 2006) and clique-related network analysis (Palla et al. 2005; Jiang and Pei 2009; Zeng et al. 2006). The proposed reverse search algorithm allows for an effective pruning of the search space without missing any solutions. Remarkably, the complexity of the delay between two consecutive solutions is in the order of the input size. This property distinguishes the reverse search approach from straightforward set enumeration algorithms and makes it applicable in cases where they are infeasible. However, for large datasets or low density thresholds, the number of solutions is prohibitive (even if only maximal solutions are considered); consequently, the search method does not scale well.

There are several remedies for this problem. The first possibility is to maintain the enumerative search, but add further constraints based on additional criteria, prior knowledge, or external data. Often, it is possible to define anti-monotonic constraints, which contribute actively to pruning of the search space (Zhu et al. 2007; Georgii et al. 2009a). Or, if relevant subsets are pre-specified for some dimensions (for instance, windows of consecutive time intervals), reverse search with respect to the other dimensions can be performed for each of these subsets individually. On the other hand, one can use the reverse search strategy, but apply heuristic criteria or sampling techniques to control the number and overlap of solutions; this allows to directly trade off the runtime and the completeness of the solution set, which we illustrated in the experiments with a simple branching heuristic. Even if it is not used for exhaustive exploration, the definition of the anti-monotonic search space has a value by itself, as solutions are visited with polynomial delay.

Finally, instead of applying the method to the whole dataset at once, it can be combined with different strategies of pre-partitioning or pre-aggregation of the data (Kolda and Bader 2007; Hu et al. 2005). Furthermore, the reverse search strategy is compatible with distributed computation, and its efficiency can be further improved by adapting the data structures and pruning techniques to the specific task at hand.

Acknowledgements We are very grateful to G. Rätsch for his encouragement and support and to the anonymous reviewers for their helpful comments. We thank K. Borgwardt for interesting discussions and for providing the email dataset; we thank Z. Zeng and J. Wang as well as H. Hu and X.J. Zhou for making their implementations available. This work was supported in part by TEKES 40141/07 (partly to EG), JSPS Kakenhi 21680025 and FIRST (partly to KT), and the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886 (partly to EG, KT, and BS). This publication only reflects the authors' views.

Appendix A: Proof of Lemma 1

We consider the case where the minimum degree instance belongs to the jth dimension. For convenience, we use the subarray representation of clusters, $U = (U_1, \ldots, U_n)$, and denote by v the corresponding element in U_j. First we show that $(U_1, \dots, U_{j-1}, U_j \setminus \{v\}, U_{j+1}, \dots, U_n)$ is a valid cluster where all index sets are non-empty. For that purpose, let us assume that $|U_j| = 1$. Then, $U_j = \{v\}$ and $\deg_U(v, j) = \sum_{k_i \in U_i} w_{k_1,\dots,k_n} =: T$, that means the degree of $v \in U_j$ is equal to the sum of all elements in the subtensor induced by U. Furthermore, T is positive because $\rho(U) > 0$. As U is non-trivial, there exists a $j' \in \{1, \dots, n\}$, $j' \neq j$, such that $|U_{j'}| > 1$. Let u' be an instance of minimum degree in $U_{j'}$, i.e., $u' \in \operatorname{argmin}_{u \in U_{j'}} \deg_U(u, j')$. Then, $T > \deg_U(u', j')$:

- for $\deg_U(u', j') > 0$: $T = \sum_{u \in U_{j'}} \deg_U(u, j') \geq |U_{j'}| \cdot \deg_U(u', j') > \deg_U(u', j')$;
- for $\deg_U(u', j') \leq 0$: obvious because $T > 0$.

So we have found a $u' \in U_{j'}$ with $\deg_U(u', j') < \deg_U(v, j)$, which contradicts the assumption of the lemma. Consequently, $|U_j| > 1$, and therefore $|U_j \setminus \{v\}| > 0$. Thus, the size of the reduced cluster is at least 1.

The second part of the lemma is shown by simple algebra:

$$\begin{aligned}
&\rho(U_1, \dots, U_{j-1}, U_j \setminus \{v\}, U_{j+1}, \dots, U_n) - \rho(U_1, \dots, U_n) \\
&\quad = \frac{1}{\operatorname{size}(U)} \left(\frac{|U_j|}{|U_j| - 1} \sum_{u \in U_j \setminus \{v\}} \deg_U(u, j) - \sum_{u \in U_j} \deg_U(u, j) \right) \\
&\quad = \frac{|U_j|}{(|U_j| - 1) \operatorname{size}(U)} \left(\frac{1}{|U_j|} \sum_{u \in U_j} \deg_U(u, j) - \deg_U(v, j) \right) \\
&\quad \geq 0.
\end{aligned}$$

The inequality holds due to the choice of v and the first part of the proof.

Appendix B: Proof of Lemma 2

1. By definition, u^* is a minimum degree instance in $\mathcal{U}$. Due to the non-negativity of W, the degree values of instances in $\mathcal{U}$ cannot decrease when adding the instance v. Thus, v is the minimum degree instance in $\mathcal{U} \cup \{v\}$.
2. The quantity $g_{\mathcal{U}}(u^*, v)$ is easily computed as follows:

$$g_{\mathcal{U}}(u^*, v) = \begin{cases} 0 & \text{if } d(u^*) = d(v), \\ \operatorname{size}(U)/(|U_{d(u^*)}| \cdot |U_{d(v)}|) & \text{otherwise.} \end{cases}$$

   Here, U is the subarray representation corresponding to $\mathcal{U}$, and $d(v)$ denotes the dimension to which v belongs. As the maximum value in W is 1 due to our normalization (see Sect. 4.1), $g_{\mathcal{U}}(u^*, v)$ corresponds to the maximum increase of the degree of u^* after addition of v. So if the degree of v exceeds the maximum possible degree of u^*, v cannot be minimum degree instance in $\mathcal{U} \cup \{v\}$.

Appendix C: Proof of Lemma 3

Let $\mathcal{U}$ be a non-trivial cluster that contains only non-isolated instances. Its ancestors are obtained by iterative reduction until a trivial cluster is reached. First, we show that a single reduction step cannot turn $\mathcal{U}$ into a cluster with (at least) two isolated instances in a dimension. Assume that this would be possible. Let $u^* \in \mathcal{U}$ denote the instance that is removed in the reduction step; further, let $u_1, u_2 \in \mathcal{U} \setminus \{u^*\}$ be the isolated instances in the resulting cluster, i.e., $\deg_{\mathcal{U}\setminus\{u^*\}}(u_1) = \deg_{\mathcal{U}\setminus\{u^*\}}(u_2) = 0$. The instances u_1 and u_2 belong to the same dimension, u^* belongs to a different dimension. Together with the non-negativity assumption for the data array, it follows that $\deg_{\mathcal{U}}(u^*) \geq \deg_{\mathcal{U}}(u_1) + \deg_{\mathcal{U}}(u_2)$. By the definition of $\mathcal{U}$, $\deg_{\mathcal{U}}(u_1) > 0$ and $\deg_{\mathcal{U}}(u_1) > 0$. Thus, $\deg_{\mathcal{U}}(u^*) > \deg_{\mathcal{U}}(u_1)$ and $\deg_{\mathcal{U}}(u^*) > \deg_{\mathcal{U}}(u_2)$, contradicting the fact that u^* is a minimum degree node. Second, also by using multiple reduction steps we cannot generate a cluster with (at least) two isolated instances from a cluster without isolated instances because isolated instances cannot be accumulated during reduction: whenever the reduction procedure generates a cluster where one or several dimensions contain an isolated instance, these instances have minimum degree and are consequently removed in the subsequent reduction steps; the elimination of an isolated instance does not affect the degree of other instances, so it cannot generate new isolated instances.

References

Acar, E., Aykut-Bingol, C., Bingol, H., Bro, R., & Yener, B. (2007). Multiway analysis of epilepsy tensors. *Bioinformatics*, *23*(13), i10–i18.

Acar, E., Çamtepe, S., & Yener, B. (2006). Collective sampling and analysis of high order tensors for chatroom communications. In *Intelligence and security informatics* (pp. 213–224). Berlin: Springer.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *VLDB '94: Proceedings of the 20th international conference on very large data bases* (pp. 487–499). San Mateo: Morgan Kaufmann.

Asahiro, Y., Iwama, K., Tamaki, H., & Tokuyama, T. (2000). Greedily finding a dense subgraph. *Journal of Algorithms*, *34*(2), 203–221.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., & Sherlock, G. (2000). Gene ontology: tool for the unification of biology. *Nature Genetics*, *25*(1), 25–29.

Avis, D., & Fukuda, K. (1996). Reverse search for enumeration. *Discrete Applied Mathematics*, *65*, 21–46.

Bader, G. D., & Hogue, C. W. (2003). An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, *4*, 2.

Baranzini, S. E., Mousavi, P., Rio, J., Caillier, S. J., Stillman, A., Villoslada, P., Wyatt, M. M., Comabella, M., Greller, L. D., Somogyi, R., Montalban, X., & Oksenberg, J. R. (2004). Transcription-based prediction of response to IFNβ using supervised computational methods. *PLoS Biology*, *3*(1), e2.

Beckmann, C. F., & Smith, S. M. (2005). Tensorial extensions of independent component analysis for multisubject FMRI analysis. *Neuroimage*, *25*(1), 294–311.

Bejerano, G., Friedman, N., & Tishby, N. (2004). Efficient exact p-value computation for small sample, sparse, and surprising categorical data. *Journal of Computational Biology*, *11*(5), 867–886.

Besson, J., Robardet, C., De Raedt, L., & Boulicaut, J. F. (2006). Mining bi-sets in numerical data. In *Lecture notes in computer science: Vol. 4747. KDID '06: Knowledge discovery in inductive databases, fifth international workshop* (pp. 11–23). Berlin: Springer.

Borgwardt, K. M., Kriegel, H. P., & Wackersreuther, P. (2006). Pattern mining in frequent dynamic subgraphs. In *ICDM '06: Proceedings of the sixth international conference on data mining* (pp. 818–822). Los Alamitos: IEEE Comput. Soc.

Cerf, L., Besson, J., Robardet, C., & Boulicaut, J. F. (2008). Data peeler: contraint-based closed pattern mining in n-ary relations. In *SDM '08: Proceedings of the SIAM international conference on data mining* (pp. 37–48).

Culhane, A. C., Schwarzl, T., Sultana, R., Picard, K. C., Picard, S. C., Lu, T. H., Franklin, K. R., French, S. J., Papenhausen, G., Correll, M., & Quackenbush, J. (2010). GeneSigDB—a curated database of gene expression signatures. *Nucleic Acids Research 38*(suppl_1), D716–D725.

Everett, L., Wang, L. S., & Hannenhalli, S. (2006). Dense subgraph computation via stochastic search: application to detect transcriptional modules. *Bioinformatics*, *22*(14), e117–e123.

Farkas, I. J., Abel, D., Palla, G., & Vicsek, T. (2007). Weighted network modules. *New Journal of Physics*, *9*, 180.

Gasch, A. P., Spellman, P. T., Kao, C. M., Carmel-Harel, O., Eisen, M. B., Storz, G., Botstein, D., & Brown, P. O. (2000). Genomic expression programs in the response of yeast cells to environmental changes. *Molecular Biology of the Cell*, *11*(12), 4241–4257.

Georgii, E., Dietmann, S., Uno, T., Pagel, P., & Tsuda, K. (2009a). Enumeration of condition-dependent dense modules in protein interaction networks. *Bioinformatics*, *25*(7), 933–940.

Georgii, E., Tsuda, K., & Schölkopf, B. (2009b). Multi-way set enumeration in real-valued tensors. In *DMMT '09: Proceedings of the second workshop on data mining using matrices and tensors* (pp. 32–41). New York: ACM.

Goldberg, L. A. (1992). Efficient algorithms for listing unlabeled graphs. *Journal of Algorithms*, *13*(1), 128–143.

Han, J., & Kamber, M. (2006). *The Morgan Kaufmann series data management systems*. *Data mining: concepts and techniques*. San Mateo: Morgan Kaufmann.

Haraguchi, M., & Okubo, Y. (2006). A method for pinpoint clustering of web pages with pseudo-clique search. In *Lecture notes in computer science: Vol. 3847*. *Federation over the Web* (pp. 59–78). Berlin: Springer.

Höppner, F., Klawonn, F., Kruse, R., & Runkler, T. (1999). *Fuzzy cluster analysis: methods for classification, data analysis and image recognition*. New York: Wiley.

Hu, H., Yan, X., Huang, Y., Han, J., & Zhou, X. J. (2005). Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, *21*(suppl_1), i213–i221.

Jaschke, R., Hotho, A., Schmitz, C., Ganter, B., & Stumme, G. (2006). TRIAS—an algorithm for mining iceberg tri-lattices. In *ICDM '06: Proceedings of the sixth international conference on data mining* (pp. 907–911). Los Alamitos: IEEE Comput. Soc.

Jegelka, S., Sra, S., & Banerjee, A. (2009). Approximation algorithms for tensor clustering. In *Algorithmic learning theory* (pp. 368–383).

Ji, L., Tan, K. L., & Tung, A. K. H. (2006). Mining frequent closed cubes in 3D datasets. In *VLDB '06: Proceedings of the thirty-second international conference on very large data bases* (pp. 811–822). VLDB Endowment/ACM, New York. http://portal.acm.org/citation.cfm?id=1164197, http://dblp.uni-trier.de/rec/bibtex/conf/vldb/JiTT06.

Jiang, D., & Pei, J. (2009). Mining frequent cross-graph quasi-cliques. *ACM Transactions on Knowledge Discovery Data*, *2*(4), 1–42.

Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T., & Ueda, N. (2006). Learning systems of concepts with an infinite relational model. In *AAAI '06: Proceedings of the twenty-first national conference on artificial intelligence* (pp. 381–388). Menlo Park: AAAI Press.

Klimt, B., & Yang, Y. (2004). The Enron corpus: a new dataset for email classification research. In *ECML '04: Proceedings of the 15th european conference on machine learning* (pp. 217–226). Berlin: Springer.

Kolda, T. G., & Bader, B. W. (2007). Tensor decompositions and applications. Technical Report SAND2007-6702, Sandia National Laboratories.

Kolda, T. G., Bader, B. W., & Kenny, J. P. (2005). Higher-order web link analysis using multilinear algebra. In *ICDM '05: Proceedings of the fifth IEEE international conference on data mining* (pp. 242–249). Los Alamitos: IEEE Comput. Soc.

Kolda, T. G., & Sun, J. (2008). Scalable tensor decompositions for multi-aspect data mining. In *ICDM '08: Proceedings of the eighth IEEE international conference on data mining* (pp. 363–372).

Koyutürk, M., Szpankowski, W., & Grama, A. (2007). Assessing significance of connectivity and conservation in protein interaction networks. *Journal of Computational Biology*, *14*(6), 747–764.

Madeira, S. C., & Oliveira, A. L. (2004). Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology of Bioinformatics*, *1*(1), 24–45.

Mishra, N., Ron, D., & Swaminathan, R. (2004). A new conceptual clustering framework. *Machine Learning*, *56*(1–3), 115–151.

Newman, M. E. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of United States of America*, *103*(23), 8577–8582.

Palla, G., Derenyi, I., Farkas, I., & Vicsek, T. (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, *435*(7043), 814–818.

Robardet, C. (2009). Constraint-based pattern mining in dynamic graphs. In *ICDM '09: Proceedings of the ninth IEEE international conference on data mining* (pp. 950–955). Los Alamitos: IEEE Comput. Soc.

Rymon, R. (1992). Search through systematic set enumeration. In *Proceedings of the third international conference on principles of knowledge representation and reasoning* (pp. 539–550).

Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review*, *1*(1), 27–64.

Shamir, R., Maron-Katz, A., Tanay, A., Linhart, C., Steinfeld, I., Sharan, R., Shiloh, Y., & Elkon, R. (2005). EXPANDER—an integrative program suite for microarray data analysis. *BMC Bioinformatics*, *6*(1), 232.

Spirin, V., & Mirny, L. A. (2003). Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences of United States of America*, *100*(21), 12123–12128.

Sun, J., Tao, D., & Faloutsos, C. (2006). Beyond streams and graphs: dynamic tensor analysis. In *KDD '06: Proceedings of the twelfth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 374–383). New York: ACM.

Tanay, A., Sharan, R., & Shamir, R. (2002). Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, *18*(Suppl 1), S136–S144.

Ulitsky, I., & Shamir, R. (2009). Identifying functional modules using expression profiles and confidence-scored protein interactions. *Bioinformatics*, *25*(9), 1158–1164.

Uno, T. (2007). An efficient algorithm for enumerating pseudo cliques. In *ISAAC '07: Algorithms and computation, eighteenth international symposium* (pp. 402–414).

Yan, C., Burleigh, J. G., & Eulenstein, O. (2005). Identifying optimal incomplete phylogenetic data sets from sequence databases. *Molecular Phylogenetics and Evolution*, *35*(3), 528–535.

Yan, X., & Han, J. (2002). gSpan: graph-based substructure pattern mining. In *ICDM '02: Proceedings of the second IEEE international conference on data mining* (pp. 721–724). Los Alamitos: IEEE Comput. Soc.

Yan, X., Zhou, X. J., & Han, J. (2005). Mining closed relational graphs with connectivity constraints. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 324–333). New York: ACM.

Zeng, Z., Wang, J., Zhou, L., & Karypis, G. (2006). Coherent closed quasi-clique discovery from large dense graph databases. In *KDD '06: Proceedings of the twelfth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 797–802). New York: ACM.

Zhao, L., & Zaki, M. J. (2005). TRICLUSTER: an effective algorithm for mining coherent clusters in 3D microarray data. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on management of data* (pp. 694–705). New York: ACM.

Zhu, F., Yan, X., Han, J., & Yu, P. S. (2007). gPrune: a constraint pushing framework for graph pattern mining. In *PAKDD '07: Proceedings of the eleventh Pacific-Asia conference on advances in knowledge discovery and data mining* (pp. 388–400). Berlin: Springer.