RC23565 (W0503-074) March 11, 2005
Computer Science

IBM Research Report

Redescriptions: Structure Theory and Algorithms

Laxmi Parida
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Naren Ramakrishna
Department of Computer Science
Virginia Tech
Blacksburg, VA

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Redescriptions: Structure Theory and Algorithms

Laxmi Parida
Computational Biology Center
IBM T J Watson Research Center
Yorktown Heights, USA
parida@us.ibm.com

Naren Ramakrishna
Department of Computer Science
Virginia Tech
Blacksburg, USA
naren@cs.vt.edu

Abstract

We present a new approach to mining redescriptions – patterns that identify subsets of data that afford multiple definitions. Redescription mining finds important applications in descriptor-rich datasets, such as in bioinformatics. The key contributions of this paper are (i) identifying the existence of a dichotomy law that the redescriptions follow (ii) definition of a notion of *irredundant* redescriptions underyling a dataset, (iii) an output-sensitive algorithm to mine the irredundant set of redescriptions in a specific form, both exact and approximate, (iv) identifying an important connection between biclusters and redescriptions, using which we can build a redescription mining algorithm around a biclustering algorithm.

Keywords: redescription, redescription mining, biclustering, data mining in biological domains.

1 Introduction

Redescription mining is a new data mining task introduced in [3]. As the name indicates, to redescribe something is to describe anew or to express the same concept in a different vocabulary. The input to redescription mining is a collection of sets such as shown in Fig. 1. Each bubble in this diagram denotes a meaningful grouping of objects (in this case, countries) according to some intensional definition. For instance, the colors green, red, cyan, and yellow (from right, counterclockwise) refer to the sets 'permanent members of the UN security council,' 'countries with a history of communism,' 'countries with land area $> 3,000,000$ square miles,' and 'popular tourist destinations in the Americas (North and South).' We will refer to such sets as *descriptors*. An example redescription for this dataset is then: 'Countries with land area $> 3,000,000$ square miles outside of the Americas' are the same as 'Permanent members of the UN security council who have a history of communism.' This redescription re-defines the set {Russia, China}. The goal of redesription mining is to find which subsets afford multiple definitions and to find these definitions. The underlying premise is that sets that can indeed be defined in (at least) two ways are likely to exhibit concerted behavior and are, hence, interesting.

Finding redescriptions is trivial if all the possible participating expressions are specified *a priori*. Instead we are given only a vocabulary of sets as in Fig. 1 but neither the way in which the sets can be

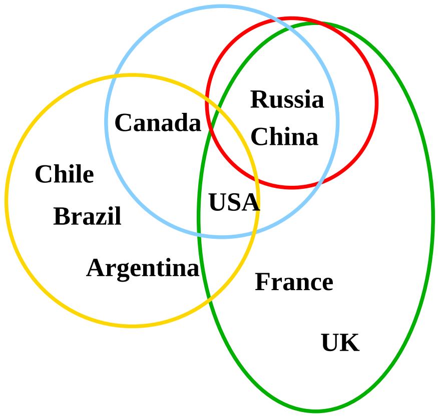

Figure 1: Example input to redescription mining.

combined into an expression nor the objects (countries) participating in the redescription are given. An algorithm is expected to automatically infer that we must subtract the yellow set from the blue set on one side but intersect the green and red sets on the other, to arrive at a redescription for two countries.

We can view redescription mining as a generalization of association rule mining [1, 4] wherein we enrich the pattern space from implications to equivalences. At the same time, however, redescription mining emphasizes constructive induction (i.e., the task of automatically creating new features for use in data mining) to an extent not previously studied by the rule mining community. It is this additional flexibility that both makes the patterns appealing and complicates the design of data mining algorithms.

Ref. [3] presents an approach (CARTwheels) to mining redescriptions by exploiting two important properties of binary decision trees. First, if the nodes in such a tree correspond to boolean membership variables of the given sets then we can interpret paths to represent set intersections, differences, or complements; unions of paths would correspond to disjunctions. Second, a partition of paths in the tree corresponds to a partition of objects. These two properties are employed in CARTwheels which grows two trees in opposite directions so that they are joined at the leaves. Essentially, one tree exposes a partition of objects via its choice of subsets and the other tree tries to grow to match this partition using a different choice of subsets. If partition correspondence is established, then paths that join can be read off as redescriptions. CARTwheels explores the space of possible tree matchings via an alternation process whereby trees are repeatedly re-grown to match the partitions exposed by the other tree. By suitably configuring this alternation, we can guarantee, with non-zero probability, that any redescription existing in the dataset would be found. However, CARTwheels has a tendency to re-find already mined redescriptions as it searches for potentially unexplored regions of the search space.

1.1 Contributions of this Paper

This paper presents new algorithms for mining redescriptions involving theoretical (a formal basis of redescriptions) as well as practical (algorithm implementation and case study) contributions:

1. We identify a dichotomy law that governs redescriptions in general
2. Given a collection of descriptors, we define a notion of an *irredundant* redescription underlying the dataset and present an output-sensitive algorithm to mine them. We explain how the irredundant redescription can help form a *basis* for the set of all possible redescriptions.
3. We highlight how finding redescriptions can exploit biclustering algorithms at its core; specifically, we show how we can mine in linear time boolean expressions satisfying a certain support threshold using a biclustering algorithm, and subsequently relate these expressions to arrive at redescriptions.
4. Our algorithm is able to redescribe in expressive and complete biases for the expressions, such as monotone CNF (conjunctive normal form) or DNF (disjunctive normal form), and mines both exact and approximate redescriptions.

The rest of the paper is organized as follows. Section 2 introduces background notation and terminology for use in this paper. It also presents a basic strategy for exploring the space of possible expressions in search of redescriptions. Section 3 relates the strategy to biclustering and introduces a concrete algorithm to mine exact redescriptions. This is extended in Section 5 to the task of mining approximate redescriptions.

2 Formalisms

Formally, the inputs to redescription mining are the universal set of objects $O = \{o_1, o_2, \ldots, o_n\}$, and a set (the vocabulary) $F = \{F_1, F_2, \ldots, F_m\}$ of proper subsets of O. The elements of F (called *features*) are assumed to form a covering of O ($\bigcup_i F_i = O$), but not necessarily a partition. For notational convenience, this information can be summarized in the $n \times m$ binary *dataset matrix* D (see Fig. 2) whose rows represent objects, columns represents the features, and the entry D_{ij} is 1 if object o_i is a member of feature F_j, and 0 otherwise. The reader will notice the immediate parallels between D and the traditional item-transaction modeling in association rule mining.

Definition 1 *(descriptor e, features $F(e)$, objects $O(e)$) A descriptor is a boolean expression on a set of features $V \subseteq F$. Given a descriptor e, we will denote the set of features involved in e by $F(e)$ and the set of objects it represents (for a presumed D) by $O(e)$.*

For ease of interpretability, notice that we have overloaded notation: F denotes the entire set of features, $F(e)$ denotes the subset of F that participates in e (similarly for O). Also, in writing boolean expressions, we will use boolean connectives ($\wedge$, $\vee$, $\neg$) as well as set constructors ($\cap$, $\cup$, $-$) interchangeably, but never together in the same expression. Example descriptors are F_3, $F_1 \cap F_4$, $\neg F_2 \vee F_3$, and $F_1 - (F_1 - F_4)$.

Two descriptors e_1 and e_2 defined over (resp.) V_1 and V_2 are distinct (denoted as $e_1 \neq e_2$), if one of the following holds: (1) $V_1 \neq V_2$, or (2) there exists some D for which $O(e_1) \neq O(e_2)$. Notice that this condition rules out tautologies. For example the descriptors $F_1 \cap F_4$ and $F_1 - (F_1 - F_4)$ are not distinct.

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
o_1	0	0	0	1	1	0	0	1
o_2	1	0	1	0	1	1	0	1
o_3	1	1	0	0	0	1	1	0
o_4	0	1	1	0	0	1	0	0
o_5	0	0	0	1	0	0	1	1

Figure 2: Example dataset matrix D.

Definition 2 *(redescriptions $R(e)$, $O(R(e))$ e' is a* redescription *of e, if and only if $O(e) = O(e')$ holds for the given D. $R(e)$ is the set of all redescriptions of e. $O(R(e)$ is defined to be $O(e)$.*

In the example dataset matrix D of Fig. 2, $(F_3 \cap F_1) \cup (F_4 - F_3)$ is a redescription of $(F_7 - F_6) \cup (F_5 - F_7)$, since they both induce the same set of objects: $\{o_1, o_2, o_5\}$. Furthermore, these expressions are also redescriptions of F_8. The reader will find it easy to verify the following two properties:

Lemma 1 *Given D, if $e_1, e_2 \neq e_1 \in R(e)$, then $(e_1 e_2), (e_1 \vee e_2) \in R(e)$.*

Lemma 2 *Redescription is reflexive, symmetric, and transitive: it induces a partition on a collection of descriptors on D.*

Clearly, the set of redescriptions of e, as defined by $R(e)$ contains redundant elements; the next task is to trim $R(e)$ to its bare essentials by identifying a 'basis' set of redescriptions for a descriptor e. As a first attempt to arrive at such a minimal set, we can reason whether this set would be parwise disjoint in its use of features, i.e., whether the following holds.

Conjecture 1 *Fixing a set of features can endow a unique (upto tautology) description of a set of objects.*

We answer in the negative using a counterxample. Given the D of Fig. 2, there are at least two distinct redescriptions ($e_1 \neq e_2$) such that $F(e_1) = F(e_2) = \{F_3, F_4\}$ and $O(e_1) = O(e_2) = \{o_1, o_2, o_4, o_5\}$: (1) $e_1 = F_3 \vee F_4$, and, (2) $e_2 = F_3 \oplus F_4 = (\neg F_3 \wedge F_4) \vee (F_3 \wedge \neg F_4)$. Redescription relationships are hence heavily data dependent. Conversely, note that if the values of both F_3 and F_4 are flipped for o_3 in D, e_1 and e_2 are no longer redescriptions of each other. While e_2 would continue to denote the set of objects $\{o_1, o_2, o_4, o_5\}$, the definition of e_1 would get expanded.

Definition 3 *(relaxation $X(e)$ of e, $e' \leq e$) Given descriptors e and e', defined on the features V and V' respectively, e' is a relaxation of e, denoted as $(e' \leq e)$, if $e \Rightarrow e'$ is a tautology. The collection of all the relaxations of e is denoted by $X(e)$.*

For example, descriptor F_1 is a relaxation of $F_1 \wedge F_2$, $F_1 \vee F_2$ is a relaxation of $F_1 \vee (F_2 \wedge F_3)$, and $F_1 \vee F_2$ is also a relaxation of F_1. It is easy to see the following:

Lemma 3 *Relaxation is reflexive, anti-symmetric, and transitive: it induces a partial order on a collection of descriptors on D.*

Lemma 4 *For each $e_2 \in X(e_1)$, $O(e_2) \supseteq O(e_1)$.*

Given a dataset D, note that a relaxation of e is not necessarily a redescription of e. Consider our running example: $F_1 \in X(F_1 \wedge F_2)$ but $F_1 \notin R(F_1 \wedge F_2)$ since $O(F_1) = \{o_2, o_3\} \supset \{o_3\} = O(F_1 \wedge F_2)$. On the other hand, $F_4 \in X(F_4 \wedge F_8)$ and $O(F_4) = O(F_4 \wedge F_8) = \{o_1, o_5\}$, hence $F_4 \in R(F_4 \wedge F_8)$. In general, therefore, we cannot express any prior relationship between $R(e)$ and $X(e)$. Consider the partial order P_X hinted at by Lemma 3: there is a directed edge from e_1 to e_2 in P_X if $e_1 \in X(e_2)$. If we choose to include 'true' and 'false' in the set of possible expressions, then P_X will also be a lattice as every pair of expressions will have a lowest upper bound as well as a greatest lower bound. Specifically, the lub of e_1 and e_2 is $e_1 \wedge e_2$ and their glb is $e_1 \vee e_2$. For instance, the glb of F_1 and $\neg F_1$ is 'true' and the lub of $F_1 \wedge F_2$ and $F_1 \wedge F_3$ is $F_1 \wedge F_2 \wedge F_3$. Notice that we can extend the lub and glb definitions to more than two expressions, by associativity.

Lemma 5 *If $(e' \in X(e)) \notin R(e)$, then $X(e') \cap R(e) = \phi$*

S ince $e' \in X(e)$ and $e' \notin R(e)$, $O(e') \supset O(e)$. For each $e'' \in X(e')$, $O(e'') \supseteq O(e')$. Thus for each $e'' \in X(e')$, $O(e'') \supset O(e)$, hence $e'' \notin R(e)$. □

Lemma 6 *If e is a relaxation of both e_1 and $(e_2 \notin R(e_1))$, then $e \notin R(e_1)$ and $e \notin R(e_2)$.*

S ince e is relaxation of e_1 and e_2, $O(e) \supseteq O(e_1) \cup O(e_2)$. Note that $O(e_1) \neq O(e_2)$. Thus $O(e) \neq O(e_1)$ and $O(e) \neq O(e_2)$, hence the result. □

2.1 Irredundant Representation of R(e)

We next address the question of describing $R(e)$ in the most concise manner, without any loss of information.

Definition 4 *(Frontier$(R(e))$) Frontier$(R(e)) \subseteq R(e)$ is defined as follows: (1) for each $e'' \in R(e)$, there is an $e' \in$ Frontier$(R(e))$ such that $e'' \in X(e')$ and (2) there is no $e'' \in R(e)$ such that some e' $\in$ Frontier$(R(e))$ is a relaxation of e''.*

Theorem 1 *Frontier$(R(e))$ is a singleton set.*

A ssume this is not true and there exist distinct $p > 1$ expressions $e_1, e_2, \ldots, e_p \in Frontier(R(e))$. Then by Lemma 1, $e_1 e_2 \ldots e_p \in R(e)$, $Frontier(R(e))$. Then $e_1, e_2, \ldots, e_p \notin Frontier(R(e))$. Hence the assumption is wrong and $p = 1$. □

This lone element of $Frontier(R(e))$ is written as $Fr(R(e))$.

To summarize, relaxation induces a partial order and redescription is a "terrain" on this lattice satisfying the following conditions: (1) By Lemma 5 if a node e' is not in $R(e)$, then no descendent of node e' can be in $R(e)$. (2) By Lemma 6 a node (e') that is not a $Fr(R(e))$ for some expression e in the partial order cannot have multiple parents from distinct description families. These suggest a very concise representation for a redescription $R(e)$ as $Fr(R(e))$ with the following algorithm to extract $R(e)$.

```
CompRedscrp(e)
{
  For each e' ∈ X1(e)
    If (e' ∉ S) then
      {output e'; CompRedscrp(e')}
}
```

$X_1(e)$ is a procedure that takes an expression e and returns the set of all possible e' such that $e' \leq e$ and there is no e'' with $e' \leq e'' \leq e$. Let S be the set of all $Fr(R(e))$ for the given data set D.

3 Mining Redescriptions

The above discussion explains how we might find $B(e)$ for a given e but overlooks the fact that, in the general setting of redescription mining, we are given neither e nor the set of objects $O(e)$. In this section, we relax these views and explain how to automatically identify redescribable e's along with their $O(e)$'s and $B(e)$'s.

3.1 Impossibility Results

We begin by making some impossibility statements in the context of mining redescriptions. The first statement asserts an impossibility about finding even a single descriptor for certain object sets, and the second statement asserts an impossibility about finding any redescriptions at all in a dataset.

Lemma 7 *If two rows (corresponding to o_i and o_j) of D are identical, there can be no descriptor involving o_i but not o_j.*

This is easy to see since no boolean expression can be constructed over D's columns that can discriminate between the two objects. The second impossibility result holds when D has at least as many rows as a truth table:

Theorem 2 *Given a $(n \times m)$ dataset D such that every possible m-tuple binary vector is a row in D, let e be a descriptor defined on D's columns. Then $R(e) = \phi$, i.e., no descriptor defined over the columns of D has a redescription.*

A ssume the contrary, i.e., there exists some $e' \neq e$ such that $O(e) = O(e')$. Then $Fr(R(e)) = Fr(R(e'))$, hence $V = F(e) = F(e')$. Next let D' be the dataset restricted to the $|V|$ columns and then D' has all possible $|V|$-tuples: thus $O(e) \neq O(e')$ since $e' \neq e$. Thus e' must be the same as e and subsequently $R(e) = \phi$. $\square$

3.2 Strong Possibility Result

We next show that even if one or few rows are absent in the dataset D, each expression e has a non-trivial redescription.

Theorem 3 *Given a $(n \times m)$ dataset D such that at least one of the m-tuple binary vector is absent in D. Then for each descriptor e defined on D's columns, $|R(e)| > 1$, i.e., every descriptor e defined over the columns of D has a redescription $e' \neq e$.*

C onsider some expression e with the support rows as $O(e)$. Let the absent m-tuples be denoted as A; these correspond to the missing collection of rows. Consider the expression e' with $O(e') = O(e) \cup A$. If $e' \neq e$, then we are done. Let $e' = e$, then there must exist at least another expression e'' distinct from e, e' such that one of the following holds: (1) $O(e'') = O(e) \cup A$ and $O(e') = O(e)$, or, (2) $O(e')$ $= O(e) \cup A$ and $O(e'') = O(e)$. (1) implies that $e'' \leq e$ and (2) implies that $e \leq e''$. Then e'' is a redescription of e. Hence the result. $\square$

3.3 Forms of expression e

Theorem 4 *(Dichotomy Law) Given a dataset D either no description e has a distinct redescription or* **all** *descriptions e on D have distinct redescriptions.*

L et D be an $n \times m$ matrix of elements. If $n = 2^m$ and all the rows are distinct, then by Thoerem 2, no expression e has a redescription. Otherwise, by Theorem 3 each expression e has a disinct redescription. Hence the result. $\square$

This bi-state phenomenon encourages us to consider some subset of expressions (or collection of rows) and the study of redescriptions of expressions becomes interesting.

In this paper we adopt the specification of the form of the expression e as a systematic approach to restriciting our study to a subset of the collection of expressions.

Recall that an *atomic* element involves a single feature or column label such as F_1 or $\neg F_1$. An expression is a *pure conjunction* if it is a conjunction of atomic elements and a *pure disjunction* if it is a disjunction of atomic elements. For example, let $e_1 = (F_1 \vee \neg F_2)$, $e_2 = (F_1 \wedge \neg F_3 \wedge F_4)$ and $e_3 = F_1 \vee (F_2 \wedge F_3)$. Then e_1 is a pure disjunction, e_2 is a pure conjunction, and e_3 is neither. If e is a pure conjunction then $\neg e$ is a pure disjunction and *vice-versa*. A CNF (conjunctive normal form) [**?**] expression e is made up of conjunctions of one or more pure disjunctions (each made up of one or more atomic elements). For instance, F_1, $\neg F_2$, $F_2 \vee F_3$, $F_2 \wedge F_5$, and $(F_1 \vee \neg F_4) \wedge (F_3 \vee \neg F_5 \vee F_8)$

	F_1	F_2	F_3	F_4
o_1	0	0	0	0
o_2	1	0	1	1
o_3	1	1	0	1
o_4	0	1	1	0
o_5	0	0	0	1

Figure 3: An example dataset matrix D to show that the Dichotomy Law does not hold for specific forms of expressions.

are CNF expressions but $F_1 \vee (F_2 \wedge F_3)$ is not (although it can be restated as one). The definition of DNF (disjunctive normal form) is similar.

Corollary 1 *If the expressions are in the CNF or DNF form, then the dichotomy law holds for the collection of descriptions.*

Since any arbitrary expression e can be written in the canonical CNF or DNF form, the dichotomy law holds. Hence we must look for further restriction in the forms of the expressions.

In this paper we will design algorithms for two kinds of restrictions on expressions (1) monotone forms [**?**] or (2) general expressions with a small number of variables. For each of the cases, we can have the expressions either in CNF or DNF forms. We make the following proposition.

Proposition 1 *If the expressions are in (1) monotone form or (2) use only some $p < m$ variables, then the dichotomy law does not hold.*

W e will prove this result using an example where the dichotomy law does not hold. Consider the dataset shown in Figure 3. We will show an expression e_1 with redescriptions and another expression e_2 with no redescriptions for this dataset in both the forms of expressions. Let $e_1 = F_1 + F_2$ with $O(e_1) = \{o_2, o_3, o_4\}$ and $e_2 = F_1 F_4$ with $O(e_2) = \{o_2, o_3\}$.

(Case 1) Let the expressions be in the monotone form. Then $O(e_1) = O(F_2 + F_3) = O(F_1 + F_3) = O(F_1 + F_2 + F_3)$. Thus $R(e_1) = \{F_2 + F_3,\ F_1 + F_3,\ F_1 + F_2 + F_3\}$. $R(e_2)$ is a singleton set i.e., e_2 has no redescriptions.

(Case 2) Let the expressions be in the general form with only two factors. Again $R(e_1) = \{F_2 + F_3,$ $F_1 + F_3\}$ and $R(e_2)$ is a singleton.

Notice that if no restriction is imposed on the form of expressions then $e_3 = F_1(F_2 + F_3)F_4$ with $O(e_3)$ $= O(e_2)$ in both the cases. $\square$

4 Mining Exact Redescriptions

We now present a general framework that given a $n \times m$ dataset D and support $k \leq n$, identifies all descriptors e such that $|O(e)| \geq k$. We focus on mining exact redescriptions here; the next section deals with approximate redescriptions.

Thus our basic mining approach has two steps:

1. Compute the $O(R(e))$'s for the e's in the predefined form and extract the $Fr(R(e))$.
2. Compute redescriptions of e from $Fr(R(e))$.

4.1 Computing O(R(e)), Fr(R(e))

Next, we claim that $Fr(R(e))$ is an expression that involves *all* the variables that play a role in $R(e)$.

Lemma 8 $F(Fr(R(e))) = \cup_{e' \in R(e)} F(e')$

L et there exist $e' \in R(e)$ such that

$$F(e') \setminus F(Fr(R(e))) \neq \phi$$

Then clearly $e' \notin X(e)$ which is a contradiction, hence the assumption must be wrong, thus for each $e' \in R(e)$, $F(e') \subseteq F(Fr(R(e)))$. Further, $Fr(R(e)) \in R(e)$, hence the result. $\square$

This result shows that if there is a mechanism for computing $O(e)$ where e involves as many variables as possible, it can be used for computing all the descriptions (and subsequently redescriptions). So we focus on computing biclusters. The rows in the bicluster correspond to $O(R(e))$. , thus when the bicluster is maximal, then $F(R(e))$ can be derived from the columns.

4.1.1 From Biclusters to Redescriptions

Given D, a bicluster is a non-empty collection of rows O and a non-empty collection of columns F such that for a fixed $j \in F$, $D[i][j] = c_j$ for a constant c_j and for each $i \in O$ and there does not exist $i' \notin O$ with $D[i][j] = c_j$ for each $j \in F$.

The bicluster is maximal (or also called a closed itemset [4]) there does not exist $j' \notin F$ with $D[i][j'] = c'_j$ for each $i \in O$ and some fixed c'_j. These conditions define the 'constant columns' type of biclusters (see [2] for different flavors of biclusters used in the bioinformatics community).

The bicluster is minimal if for each $j \in F$, the collection of rows O and the collection of columns $F \setminus \{j\}$ is no longer a bicluster.

A generic biclustering algorithm is shown in Figure 4 which is capable of computing both maximal and minimal biclusters. In detail, the algorithm can be understood as follows. For this description assume that *depth* is set to $(m + 1)$, *compPOSET* is set to $FALSE$, B and E are each set to 1. The algorithm systematically generates and maintains $\mathcal{S}$, a polymorphic array of (two) sets where $\mathcal{S}[1]$ is a set of row numbers and $\mathcal{S}[2]$ is a set of column labels which are to be intrepreted as an expression that is a conjunction of these column labels. Depending on how the algorithm is invoked, $\mathcal{S}$ will either represent a maximal or a minimal bicluster. A maximal bicluster ($\mathcal{S}$ is such that no other term can be added to $\mathcal{S}[2]$ without changing the size of $\mathcal{S}[1]$. A minimal bicluster $\mathcal{S}$ is such that no other term can be removed from $\mathcal{S}[2]$ without changing the size of $\mathcal{S}[1]$. Notice that when computing maximal biclusters, $\mathcal{S}[2]$ will be a single set but could possibly be a collection of sets when computing minimal bicluster. Given D and a support k, the maximal or minimal biclusters are computed recursively by ordered search. The algorithm uses a data structure $\mathcal{T}$, such as a tree, to store the sets $\mathcal{S}[1]$ and $\mathcal{S}[2]$ computed in lines (2.4) and (2.7) respectively, so that the query of line (3.3) can be answered in $\log n$ time. Also, the following lemma is straightforward to verify.

Lemma 9 *Let $\overline{\mathcal{S}}$ be defined as follows: (1) $\overline{\mathcal{S}}[1] = U - \mathcal{S}[]$ and (2) $\overline{\mathcal{S}}[2] = \{\overline{f} \mid f \in \mathcal{S}[2]\}$. $\mathcal{S}$ is a minimal disjunction form if and only if $\overline{\mathcal{S}}$ is a minimal conjunction form.*

The *depth* parameter is used to terminate the call when the required amount of factors (columns) in the bicluster have been collected. The *compPOSET* parameter is used to compute the connectivity in the partial order of the $O(-)$ sets. When B is set to 0, then the negation of each column is used, when E is set to 1 the column is used as-is, hence when B=E=1, then no column is negated and when B=E=0, each column is negated.

Time Complexity. Here we give the worst case analysis of the *Generic* algorithm. Let C be all the biclusters computed. In the case of maximal biclusters, the time taken by this algorithm is $O(L \log n + mn)$ where $L = \sum_{c \in C} |c|$. In the case of minimal biclusters, the time taken by this algorithm is $O(|C| \log n + mn)$. Also, in each of the cases *Generic* is invoked only a constant number (≤ 2) of times.

4.1.2 Expressions (e) in monotone CNF form

Since the forms are monotone, no negation of a variable (column) is permitted.

1. Find all minimal monotone pure disjunctions in D, by performing the following two substeps:
   (a) Find all minimal pure conjunctions in D using the biclustering algorithm in the minimal mode; Generic($\mathcal{S}[], m, k, MINIMAL, compPOSET = FALSE, B = 0, E = 0, 0, depth = m + 1$)

   (b) Extract all minimal monotone pure disjunctions by negating each of these computed minimal conjunctions (see Lemma 9). Let the number of disjunctions computed be A in number.
2. Augment matrix D with the results of the last step. For each minimal disjunction form $\overline{\mathcal{S}}$, introduce a new column c in D with

$$D[i, c] = \begin{cases} 1 & \text{if } i \in \overline{\mathcal{S}} \\ 0 & \text{otherwise} \end{cases}$$

   Let the number of minimal disjunction forms in D be a. Thus the augmented D' is of size $n \times (m + A)$. Next, find all the monotone conjunction terms as maximal biclusters in D'.
   Generic($\mathcal{S}[]$,m,k,$MAXIMAL$,$compPOSET = TRUE$,$B = 1$,$E = 1$,0,$depth = m + 1$)

Generic($\mathcal{S}[], j, k, flag, compPOSET, B, E, d, depth$)

(0) If ($j \leq 0$) exit; If ($d \geq depth$) exit

(1) $l_b \leftarrow B$, $l_e \leftarrow E$

(2) For $\ell \leftarrow l_b, l_e$

  Ancestor$_\ell$ $\leftarrow$ TRUE

  //==== $\mathcal{S}_\ell[1] \leftarrow \{i \mid D[(i \in \mathcal{S}), j] = \ell\}$

  For each $i \in \mathcal{S}[1]$

    If $D[i,j] = \ell$ $\mathcal{S}_\ell[1] \leftarrow \mathcal{S}_\ell[1] \cup \{i\}$

      Else Ancestor$_\ell$ $\leftarrow$ FALSE

  If ($\ell = 0$) $f \leftarrow \overline{F}_j$ Else $f \leftarrow F_j$

  $\mathcal{S}_\ell[2] \leftarrow \mathcal{S}[2] \cup \{f\}$

(3) For $\ell \leftarrow l_b, l_e$ //==== 1st and 2nd child of traversal

  If $|S_\ell[1]| \geq k$ {

    If $\mathcal{S}_\ell[1]$ exists in $\mathcal{T}$ as $\mathcal{S}^p[1]$ {

      If (compPOSET) S is added to parent list of S^p

      If Ancestor$_\ell$= TRUE { //==== $\mathcal{S}[2] \supset \mathcal{S}^p[2]$ holds

        If $flag$=$Maximal$ Update $\mathcal{S}^p[2] \leftarrow \mathcal{S}^p[2] \cup \mathcal{S}_\ell[2]$

        Generic($\mathcal{S}_\ell[], j-1, k, flag, compPOSET, B, E, (d+1), depth$)

      Else // [*If Ancestor$_\ell$= TRUE*] ==== $\mathcal{S}[2] \subset \mathcal{S}^p[2]$ holds

        If $flag$=$Minimal$ Add $\mathcal{S}_\ell$ to $\mathcal{T}$ as $\mathcal{S}^q$

        //==== terminating traversal

      } // [*If Ancestor$_\ell$= TRUE*]

    Else //[*If $\mathcal{S}_\ell[1]$ exists*]

      Add $\mathcal{S}_\ell$ to $\mathcal{T}$ as $\mathcal{S}^q$

      If (compPOSET) S is assigned parent of S_ℓ

      Generic($\mathcal{S}_\ell[], j-1, k, flag, compPOSET, B, E, (d+1), depth$)

    } //[*If $\mathcal{S}_\ell[1]$ exists*]

  } //[*If $|S_\ell[1]| \geq k$*]

(4) Generic($\mathcal{S}[]$, j-1, $k, flag, compPOSET, B, E, d, depth$) //==== 3rd child of traversal

Figure 4: Sketch of the algorithm: This is a biclustering algorithm that multiplexes three tasks (1) computing minimal biclusters (2) computing maximal biclusters and (3) generating the connectivity structure (partial order) of the biclusters.

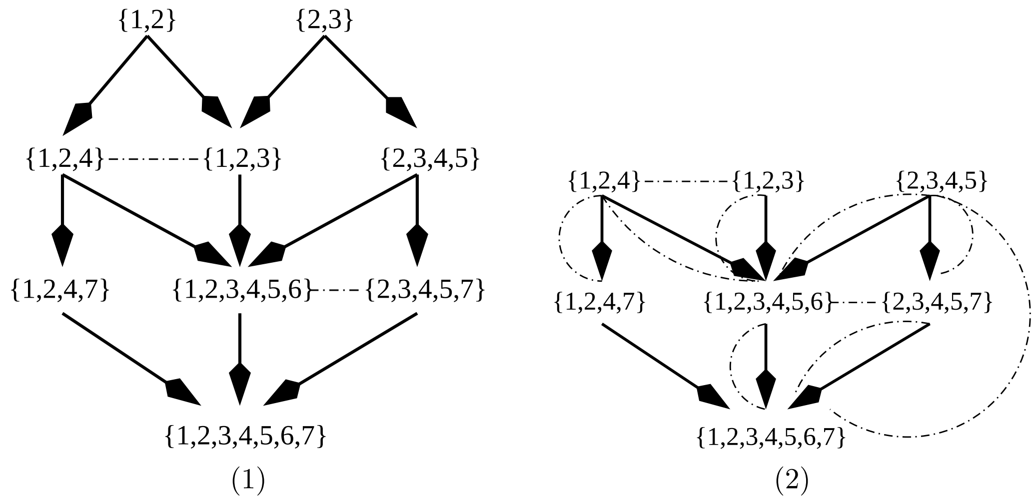

Figure 5: An example to show the steps involved when $\xi < 1$. Let $k = 3$ and $\xi = 0.5$ with $n = 7$ (7 rows). Then $k' = 2$. (1) shows the redescription graph G'_R using k'. The special edges labeled with J_ξ are shown as undirected dashed lines between the vertices: they correspond to straddling sets S_1 and S_2 with $J(S_1, S_2) \geq \xi$. The collection of rows corresponding to the biclusters are shown here as nodes in the graph. The directed solid edges correspond to the connectivity in the partial order. (2) shows the G_R when the nodes with support $< (k = 3)$ have been removed. Further, the edges with label J_ξ are shown with undirected dashed edges: these new edges correspond to the case when a set S_1 is contained in S_2 with $J(S_1, S_2) \geq \xi$.

4.1.3 Expressions (e) in monotone DNF form

There are two ways to computing the DNF form. The first is to compute the CNF forms and then derive the DNF forms for each CNF form.

The other approach is to switch the order of calls to the routine *Generic*: first compute maximal conjunctions and next compute the minimal disjunctions. Due to space constraints the details will be presented in the full version of the paper.

4.1.4 Expressions (e) with a f number of factors

In this case we can use the columns as-is as well as in the negated forms, hence B is set to 0 and E is set to 1. To limit the number of factors, *depth* is set to f.

Notice that we no longer have to impose the monotone condition. For the CNF form, the steps are identical to the monotone CNF form with the following changes in the calls. In Step 1(a) the following call is made: Generic($\mathcal{S}[], m, k, MINIMAL, compPOSET = FALSE, B = 0, E = 1, 0, depth = f$) and in Step 2 the following call is made: Generic($\mathcal{S}[]$,m,k,$MAXIMAL$,$compPOSET = TRUE$,$B = 0$,$E = 1$,0,$depth = f$).

4.2 Computing Redescriptions R(e)

Observe that if a set $S[1]$ is generated such that $S[1] \subset S'[1]$, then $S'[1]$ must have been generated at an ancestor node of the implicit tree of the recursive calls of *Generic*. Thus the connectivity of the partial order is constructed during in the final call of *Generic*. Let this partial order be termed the redescription graph and is denoted as G_R.

The redescription is computed using the algorithm presented in Section 2.1. Checking for existence of an $Fr(R(e))$ in the algorithm is simplified by traversing G_R. This is to be interpreted for $G_R(V, E)$ as follows: if node v_1 is a parent of node v_2, then the directed edge $v_1 v_2 \in E$.

5 Mining Approximate Redescriptions

Given two sets O_1 and O_2 the Jaccard's coefficient J of the two is given by

$$J(O_1, O_2) = \frac{|O_1 \cap O_2|}{|O_1 \cup O_2|} = \frac{|O_1 \cap O_2|}{|O_1 \cap O_2| + |O_1 - O_2| + |O_2 - O_1|}$$

When O_1 and O_2 are identical, then $J = 1.0$. In practice, it is useful to talk about O_1, O_2 that are nearly equal but not necessarily exactly, i.e., the two sets have a Jaccard's coefficient $\xi < 1$.

Next we define approximate redescriptions in terms of the Jaccard's coefficient.

Definition 5 *(approximate redescriptions $R_\xi(e)$, $O(R_\xi(e))$) Let $0 < \xi \leq 1.0$. e' is an ξ approximate redescription of e, if and only if $J(O(e), O(e')) \geq \xi$ holds for the given D. $R_\xi(e)$ is the set of all ξ approximate redescriptions of e. $O(R_\xi(e))$ is defined to be $\cup_{e' \in R_\xi(e)} O(e')$.*

The pseudocode of the algorithm is as follows. Assume a given D and some $0 < \xi \leq 1.0$. Notice that when $\xi = 1.0$, the redescriptions are exact and are computed as discussed in the last section.

```
Let R be the set of all R_1's
  CompApproxRedscrp(e, ξ)
  {
    Let O(R_ξ(e)) ← O(R_1(e))
    For each R_1(e') ∈ R
      If (J(O(R(e)), O(R_1(e'))) ≥ ξ) then
        O(R_ξ(e)) ← O(R_ξ(e)) ∪ O(R_1(e'))
  }
```

In practice however, this algorithm can be made very efficient using the partial order connectivity of the exact approximations. This is discussed in the following section.

5.1 Approximate Redescriptions & Partial Order

The task here is to identify each pair of sets O_1 and O_2 such that $J(O_1, O_2) \geq \xi$. We exploit the redescription graph G_R to cut down on the number of comparisons between sets. There are two cases:

(1) $O_1 \subset O_2$ and (2) $(O_1 \setminus O_2) \neq \phi$ or $(O_2 \setminus O_1) \neq \phi$, i.e., the two sets straddle. The first case is easier to detect since the possible candidates are along a path on the redescription graph G_R. The second case is not so straight forward.

We observe that given a support k, two straddling sets O_1, O_2 are such that $|O_1 \cap O_2| \geq k'$ for some k' whose value depends on k. If the redescription graph G_R is augmented with these sets (that have a support k') to obtain G'_R, then each pair of straddling sets O_1, O_2, with $J(O_1, O_2) \geq \xi$ has a common parent in G'_R. Again, it is easy to check $J(O_1, O_2)$ on G_R by simply keeping track of the cardinalities of the sets. For example if $v_1v_2 \in E$ with $|O_1| = k_1, |O_2| = k_2$ then it is easy to see that $J(O_1, O_2) = k_1/(k_2 - k_1)$. And when $v_0v_1, v_0v_2 \in E$ then $J(O_1, O_2) = k_0/(k_1 + k_2 - 2k_0)$ where k_0, k_1, k_2 have the usual meaning.

To summarize, the algorithm with a given Jaccard's coefficient $\xi \leq 1$ and a support k is as follows:

1. We first estimate k', a lower bound on $|O_1 \cap O_2|$, where $|O_1|, |O_2| \geq k$ with $J(O_1, O_2) \geq \xi$. It is easy to see that
$$k' = \frac{2k\xi}{1+\xi}$$
2. In Step 1 of the last section, we use support k' (instead of k) and obtain all the expressions e in the required form and the corresponding sets $O(e)$ termed O_i's. Construct the redesription graph $G'_R(V, E)$ in Step 2.
3. A special edge with label J_ξ is introduced in the graph as follows: If $v_0v_1, v_0v_2 \in E$ and $J(O_1, O_2) = k_0/(k_1 + k_2 - 2k_0) \geq \xi$, then the special edge between v_1 and v_2 is introduced. Further, when for each child v_{11} of v_1 and each child v_{21} of v_2, where v_1v_2 has the J_ξ labeled edge, if $J(O_{11}, O_{21}) = k_0/(k_{11} + k_{21} - 2k_0) \geq \xi$, then the special edge between v_{11} and v_{21} is introduced. This process is continued till no special edges can be added. This is the case when O_1 and O_2 straddle.
4. Next G'_R is transformed to G_R by removing all the vertices that correspond to sets O with $|O| < k$.
5. Edge with label J_ξ is introduced in G_R as follows: If $v_1v_2 \in E$ and $J(O_1, O_2) = k_1/(k_2 - k_1)\ \xi$, then the special edge with label J_ξ between v_1 and v_2 is introduced.

   Next, for all possible v_1v_2 with label J_ξ, and $v_2v_3 \in E$ with $J(O_2, O_3) = k_2/(k_3 - k_2) \geq \xi$, then the special edge between v_1 and v_3 with label J_ξ is introduced.

   This is the case when $O_1 \subset O_2$ holds.
6. Compute the exact redescriptions as discussed in the last section.
7. Reading off the redesrciptions from G_R: Each pair of nodes v_1v_2 with edge label J_ξ is a redescription of each other with Jaccard's coefficient ξ.

References

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, pages 487–499, Sep 1994.

[2] S.C. MAdeira and A.L. Oliveira. Biclustering Algorithms for Biological Data Analysis: A Survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, Vol. 1(1):pages 24–45, Jan 2004.

[3] N. Ramakrishnan, D. Kumar, B. Mishra, M. Potts, and R.F. Helm. Turning CARTwheels: An Alternating Algorithm for Mining Redescriptions. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*, pages 266–275, Aug 2004.

[4] M.J. Zaki. Mining Non-Redundant Association Rules. *Data Mining and Knowledge Discovery*, Vol. 9(3):pages 223–248, Nov 2004.