Noname manuscript No.
(will be inserted by the editor)

Relational Data Factorization

Sergey Paramonov · Matthijs van Leeuwen · Luc De Raedt

Received: date / Accepted: date

Abstract Motivated by an analogy with matrix factorization, we introduce the problem of factorizing relational data. In matrix factorization, one is given a matrix and has to factorize it as a product of other matrices. In relational data factorization (ReDF), the task is to factorize a given relation as a conjunctive query over other relations, i.e., as a combination of natural join operations. Given a conjunctive query and the input relation, the problem is to compute the *extensions* of the relations used in the query. Thus, relational data factorization is a relational analog of matrix factorization; it is also a form *inverse* querying as one has to compute the relations in the query from the result of the query. The result of relational data factorization is neither necessarily unique nor required to be a lossless decomposition of the original relation. Therefore, constraints can be imposed on the desired factorization and a scoring function is used to determine its quality (often similarity to the original data). Relational data factorization is thus a constraint satisfaction and optimization problem. We show how answer set programming can be used for solving relational data factorization problems.

Keywords Answer Set Programming · Inductive Logic Programming · Pattern Mining · Relational Data · Factorization · Data Mining · Declarative Modeling

Sergey Paramonov
Machine Learning, Department of Computer Science, KU Leuven, Leuven, Belgium
E-mail: sergey.paramonov@cs.kuleuven.be

Matthijs van Leeuwen
Machine Learning, Department of Computer Science, KU Leuven, Leuven, Belgium and
Leiden Institute of Advanced Computer Science, Leiden University, Leiden, the Netherlands
E-mail: m.van.leeuwen@liacs.leidenuniv.nl

Luc De Raedt
Machine Learning, Department of Computer Science, KU Leuven, Leuven, Belgium
E-mail: luc.deraedt@cs.kuleuven.be

1 Introduction

The fields of data mining and machine learning have contributed numerous effective and highly optimized algorithms for analyzing data. However, this focus on efficiency and scalability has come at the cost of generality. Indeed, while the algorithms are highly effective, their application range is often very restricted, and the algorithms are typically hard to change and adapt even to small variations on the problem definition. This observation has led to an interest in declarative methods for data mining and machine learning in which the focus lies on the use of expressive models that can capture a wide range of different problem settings and that can then be solved using off-the-shelf constraint solving technology; see Guns et al (2013a); De Raedt (2012); Arimura et al (2012); De Raedt (2015).

Motivated by this quest for more general and generic data analysis approaches, the present paper introduces the problem of relational data factorization (ReDF). ReDF is inspired by matrix factorization, one of the most popular techniques in machine learning and data mining for which many variants have been studied, such as non-negative, singular value and Boolean matrix factorization. In matrix factorization, one is given an $n \times m$ matrix $\mathbf{A}$, and the problem is to rewrite it as the product of some other matrices, e.g., the product of an $n \times k$ matrix $\mathbf{B}$ and $k \times m$ matrix $\mathbf{C}$ such that $\mathbf{A}_{i,j} = \sum_k \mathbf{B}_{i,k} \cdot \mathbf{C}_{k,j}$. In relational data factorization, one is given a relation (i.e., a set of tuples over the same attributes) and asked to to rewrite it in terms of other relations. Consider, for instance, a relation *sells(Company, Part, Project)*, stating that companies sell particular parts to particular projects. While it is well-known that ternary relations, in general, can not be rewritten as the join of three binary relations (Heath, 1971; Jones et al, 1996)[1], we might be interested in an approximation of the ternary relation. That is, we might approximate *sells(Company, Part, Project)* by the query *offers(Company,Part), needs(Project, Part), deliversto(Company, Project)* (we follow logic programming notation, where the same variable name denotes a natural join). The question is then how to determine the extensions for the relations *offers*, *needs*, and *delivers*. The found solution will generally be imperfect, so in ReDF we want to find the best approximation w.r.t. a scoring function and we allow the user to specify hard constraints. In the example these might specify, e.g., that only tuples in the target relation *sells* may be derivable from the query.

In this paper, we develop a modeling and solving approach for ReDF using answer set programming (ASP) (Brewka et al, 2011). This is realized by showing for a number of ReDF problems how they can be tackled with ASP. This leads to the identification of constraints and scoring functions, which we then abstract to an even higher-level declarative language. We show that the resulting ReDF framework is general and generic and is in line with the declarative modeling approach to machine learning and data mining as 1) it allows one to easily specify and solve a wide range of well-known data analysis problems (such as tiling, Boolean matrix factorization, discriminative pattern mining, matrix block diagonalization, etcetera), 2) it is effective for prototyping such tasks (as we show in our experiments), even though it

[1] Heath Theorem: a relation $R(x, y, z)$ satisfying a functional dependency $x \rightarrow y$ can always be losslessly decomposed into its projections $R_1 = \pi_{xy} R$ and $R_2 = \pi_{xz} R$; see (Jones et al, 1996, Table 5)

cannot yet compete with optimized special purpose algorithms in terms of efficiency, and 3) the constraints and optimization criteria are specified in a declarative and flexible manner. Translating problem definitions in the ReDF framework to ASP models is straightforward, and small changes in the problem definitions generally result in small changes in the model.

Relational data factorization is a form of relational learning. That is, it is a relational analog of matrix factorization and is therefore relevant to inductive logic programming (Muggleton and De Raedt, 1994; De Raedt, 2008) and can also be seen as a form of large-scale abduction (Denecker and Kakas, 2002). Moreover, the solution techniques that we adopt are based on answer set programming, which has also been adopted in some recent works and methods on inductive logic programming (Paramonov et al, 2015; Järvisalo, 2011). The implementation techniques we employ may also be used in more traditional inductive logic programming settings.

This paper is structured as follows. Section 2 introduces the formal ReDF framework. Section 3 introduces ASP. Section 4 shows how a wide range of data mining problems can be expressed as ReDF problems. Section 5 introduces some novel problems that the framework can express. Section 6 discusses the encoding of the problems into ASP, while Section 7 reports on the experimental evaluation. In Section 8 we discuss related work, and we formulate some conclusions and directions for future work in Section 9.

2 Relational Data Factorization

Before we formalize the ReDF problem and approach in its full generality, we illustrate Relational Data Factorization on the *sells(Company, Part, Project)* example from the Introduction.

2.1 An example

Assume we are given 1) a set of tuples for the *database* relation *sells(Company, Part, Project)*, 2) a definite *shape* clause defining the predicate *approx(Company, Part, Project)*, e.g.,

approx (Com, Pa, Proj) $\leftarrow$ *offers(Com, Pa), needs(Proj, Pa), deliversto(Com, Proj),*

which should approximate the database relation *sells(Company, Part, Project)* in terms of the (unknown) relations *offers(Company, Part)*, *needs(Project, Part)* and *deliversto(Com, Project)*, and 3) an *error* function $error(approx, sells)$ that measures how different the database predicate and its approximation are, e.g., the number of tuples that is one in relation but not in the other. Then, the goal is to find sets of tuples for the unknown relations that minimize the *error*.

In practice, it is usually impossible to find a perfect solution (with $error = 0$) to relational data factorization problems, in this example because of Heath's theorem (Heath, 1971) (as discussed in the Introduction). Therefore, it is often useful to impose further restrictions on the sets to be considered. One such constraint could specify that there is no *overcoverage*, i.e., that all tuples in *approx* must be in *sells*.

2.2 Problem statement

Using a logic programming formalism, we generalize the above example into the following ReDF problem statement.
Given:

- a dataset D: a set of ground facts for target predicate *db*;
- a factorization shape Q: $\mathit{approx}(\bar{T}) \leftarrow q_1(\bar{T}_1), \ldots, q_k(\bar{T}_k)$, where the q_i are factors and the $\bar{T}_i$ denote tuples of variables;
- a set of constraints C;
- an *error* function measuring difference between two predicates (i.e., between the corresponding sets of ground facts);

Find: the set of ground facts F for the factors q_i that minimizes $\mathit{error}(\mathit{approx}, \mathit{db})$ and for which $Q \cup F \cup D$ satisfies all constraints in C.

The factorization shape is a single non-recursive rule defining *approx*, the approximation of the target predicate *db*, where the predicates in the body are the factors. If a variable occurs in a body atom $\bar{T}_i$ and not in $\bar{T}$ (the head), then it is called *latent*. The task is to find a set F of ground facts defining the factors q_i. Furthermore, each such set F uniquely determines a set of facts for *approx*. Notice that if a predicate q_i is already known and defined, then the task simplifies.

As in matrix factorization, it is quite likely that a perfect solution, with $\mathit{error} = 0$, cannot be obtained. Consider the following example: $\mathit{db}(X, Y) \leftarrow p(X), q(Y)$ and dataset $D = \{\mathit{db}(a, c), \mathit{db}(b, d)\}$. Then it is impossible to perfectly reconstruct the target D. If $F = \{p(a), p(b), q(c), q(d)\}$, the resulting program overgeneralizes as it entails facts not in D: $\mathit{db}(a, d) \in \mathit{approx}$ and $\mathit{db}(a, d) \notin D$; if, on the other hand, there are facts in D that are not entailed in *approx*, one undergeneralizes (e.g., when $F = \emptyset$).

The scoring function in relational factorization measures the *error* between the predicates *approx* and *db*. Instead of minimizing *error*, however, in some cases it is more convenient to maximize *similarity*. Since these two perspectives can be trivially transformed from one to the other, we will use both without loss of generality.

2.3 Approach

To make this setup operational, we represent ReDF problems at two different levels. First, at the high level, we characterize typical constraints of interest that are employed across different models. Further, all problems are formulated using the template shown in Listing 1. Second, at the low level, the high-level constraints and encodings are formulated in ASP. The high-level constraints can in principle be automatically transformed into low-level ones.

Listing 1: Prototypical template of a high level problem encoding

Input: a set of facts D for the *db* predicate
Shape: $\mathit{approx}(\bar{T}) \leftarrow q_1(\bar{T}_1), \ldots, q_k(\bar{T}_k)$
Find: $q_1 \ldots q_k$

Satisfying: $C_1(approx, db) \wedge \ldots \wedge C_n(approx, db)$
Minimizing: $error(approx, db)$

We next illustrate this on the *sells* example. The high-level description from which we start is given in Listing 2.

Listing 2: Sells example encoding

Input: *sells(c1,pa1,proj1), sells(c2,pa1,proj2)*
Shape: $approx(C, Pa, Prj) \leftarrow$ *offers(C,Pa), needs(Prj,Pa), deliversto(C,Prj).*
Find: *offers(·), needs(·), deliversto(·)*
Minimizing: $error(approx, sells)$

Next, this high-level formulation can be encoded in and solved using the ASP program given in Listing 3 (here, an ASP program can be thought as a conjunction of logical rules, where implication is denoted by ":-").

Listing 3: Factorization of a ternary relation into three binary relations

```
1  %factorization shape
2  approx(Com,Pa,Proj) :- offers(Com,Pa), needs(Proj,Pa), deliversto(Com,Proj).
3  %relation generators
4  0 { offers(Com,Pa)        } 1 :- sells(Com,Pa,Proj).
5  0 { needs(Proj,Pa)        } 1 :- sells(Com,Pa,Proj).
6  0 { deliversto(Com,Proj) } 1 :- sells(Com,Pa,Proj).
7  %optimization function
8  overcoverage(Com,Pa,Proj)  :-      approx(Com,Pa,Proj), not sells(Com,Pa,Proj).
9  undercoverage(Com,Pa,Proj) :- not approx(Com,Pa,Proj),      sells(Com,Pa,Proj).
10 error(Com,Pa,Proj) :- undercoverage(Com,Pa,Proj).
11 error(Com,Pa,Proj) :- overcoverage(Com,Pa,Proj).
12 #minimize[error(Com,Pa,Proj)].
```

We introduce ASP in more detail below, but this model is easy to understand if one is familiar with the basics of logic programming. The ASP model basically defines the necessary predicates in ASP using a set of clauses. In addition, the rule in Line 4 encodes the constraint that whenever a tuple holds for *sells(Com, Pa, Proj)* there should be 0 or 1 corresponding tuples for the predicate *offers(Com, Pa)*. Furthermore, the minimize statement specifies that we are looking for a model (a set of ground facts or tuples) that minimizes the error. The encoding in Listing 3 together with a set of facts for *sells* can be given to an ASP solver such as clasp (Gebser et al, 2011b).

Observe that the relational data factorization approach we propose perfectly fits within the declarative modeling paradigm for machine learning and data mining (De Raedt, 2012). Indeed, the next sections will show that it naturally supports a wide range of popular and well-known factorization problems. Modeling different problems corresponds to specifying different constraints, shapes and optimization functions. By doing so, one obtains a deep understanding of the relationships among the many variations of factorization, and one can easily design, prototype and experiment with new variations of factorization problems. Furthermore, the models of factorization are in principle solver-independent and do not depend on a particular ASP solver implementation.

Notice that it would also be possible to use other constraint satisfaction and optimization approaches (such as, e.g., Integer Linear Programming), but given that we work within a relational framework, ASP is a natural choice. It is also declarative

and has the right expressiveness for the class of problems that we will study, many of which are NP-complete such as BMF; see Subsection 4.2.

Finally, let us mention that there are many factorization approaches in both linear algebra, databases, and even in logic. We provide a detailed discussion of their relationship to ReDF in Section 8.

3 Preliminaries: ASP essentials

We use the answer set programming (ASP) paradigm for solving relational data factorization problems. Contrary to the programming language Prolog, which is based on a proof-theoretic approach to answer queries, ASP follows a model generation approach. It has been shown to be effective for a wide range of constraint satisfaction problems (Gebser et al, 2012).

The remainder of this subsection introduces the essentials of ASP in a rather informal way. ASP is a rich (and technical) research area, so we do not focus on technical issues as these would complicate the presentation, but rather refer the interested reader to Gebser et al (2012); Eiter et al (2009); Leone et al (2002); Lifschitz (2008) for more details on this. For the actual implementation, we will use the clasp system (Gebser et al, 2012; Brewka et al, 2011).

Definition 1 (Disjunctive datalog program) A disjunctive datalog program is a finite set of rules of the form:

$$a_1 \vee a_2 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k, \textit{ not } c_1, \ldots, \textit{ not } c_h$$

where $a_1, \ldots, a_n, b_1, \ldots, b_k, c_1, \ldots c_h$ are atoms of a function-free first order language L. Each atom is an expression of the form $p(t_1, \ldots, t_n)$, where p is a predicate name and t_i is either a constant or a variable. We refer to the head of rule r as $H(r) = \{a_1, \ldots, a_n\}$ and to the body as $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_k\}$ is the positive part of the body and $B^-(r) = \{c_1, \ldots, c_h\}$ the negative.

If a disjunctive datalog program P has variables, then its semantics are considered to be the same as that of its grounded version, written as *ground(P)*, i.e. all variables are substituted with constants from the Herbrand Universe H_P (the constants occurring in the program). The semantics of a program with variables is defined by the semantics of the corresponding grounded version.

An interpretation I w.r.t. to a program P is a set of ground atoms of P. Let P be a positive disjunctive datalog program (i.e. without negation), then an interpretation I is called closed under P, if for every $r \in \textit{ground}(P)$ it holds that $H(r) \cap I \neq \emptyset$ whenever $B(r) \subseteq I$.

Definition 2 (Answer set of a positive program (Eiter et al, 2009)) An answer set of a positive program P is a minimal (under set inclusion) interpretation among all interpretations that are closed under P.

Definition 3 (Gelfond-Lifschitz reduct) A reduct of a ground program P w.r.t. an interpretation I, written as P^I, is a positive ground program P^I obtained by:

- removing all rules $r \in P$ for which $B^-(r) \cap I \neq \emptyset$;
- removing the literals "*not* a" from all remaining rules.

Intuitively, the reduct of a program is a program where all rules with bodies contradicting I are removed and in all non-contradicting all negative ones are ignored. The interpretation I is a guess as to what is true and what is false.

Definition 4 (An answer set of a disjunctive program) An answer set of a disjunctive program P is an interpretation I such that I is an answer set of positive ground program *ground*$(P)^I$.

Example 1 Consider the following disjunctive datalog program P.

$$a \vee c \leftarrow b. \qquad b \leftarrow a,\ \textit{not}\ c. \qquad a.$$

If we take the interpretation $I = \{a, b\}$ of P as candidate answer set, then the reduct P^I is

$$a \vee c \leftarrow b. \qquad b \leftarrow a. \qquad a.$$

and it is easily seen that I is a minimal interpretation closed under P^I, and therefore an answer set.

We also use a special form of disjunctive rules called *choice rules* Gebser et al (2012):

$$v_1\ \{a_1, a_2, \ldots a_n\}\ v_2 \leftarrow b_1, \ldots, b_k,\ \textit{not}\ c_1, \ldots,\ \textit{not}\ c_h$$

where v_1 and v_2 are integer constants. The semantics are as follows: if the body is satisfied, then the number of true atoms in $\{a_1, a_2 \ldots a_n\}$ is from v_1 to v_2.

An aggregate atom is an atom that has the following form: $l\#\{a_1, \ldots, a_n\}u$ where l and u are constant numbers, each a_i is a literal. The atom is true in an answer set A iff there are from l to u literals a_i that are true in A.

Another construct is *maximization* (Gebser et al, 2012; Leone et al, 2002) (*minimization* is defined analogously) stated as $\#maximize\{a_1 = k_1, \ldots, a_n = k_n\}$, where $a_1, \ldots, a_n$ are classic literals and $k_1, \ldots, k_n$ are integer constants (possibly negative). The semantics of this constraint are as follows: a model I is selected if the weighted sum of $[a_i] * k_i$ is maximal in I, where $[\cdot]$ are Iverson brackets, i.e. $[a]$ is equal to 1 iff a is true in I and 0 otherwise.

4 Application to Data Mining Problems

In this section we show that the ReDF framework generalizes a wide range of data mining tasks and provides a truly declarative modeling approach for relational data factorization. We introduce a range of constraints and optimization criteria that can be used in practice. The data mining tasks studied include tiling (Geerts et al, 2004), Boolean Matrix Factorization (BMF) (Miettinen et al, 2008), discriminative pattern mining (Knobbe and Ho, 2006), and block-diagonal matrix forms (Aykanat et al, 2002).

4.1 Tiling

Data mining has contributed numerous techniques for finding patterns in (Boolean) matrices. One fundamental approach is that of *tiling* (Geerts et al, 2004). A tile is a rectangular area in a Boolean matrix represented by set of rows and columns such that all values on the corresponding rows and columns in the matrix are equal to 1.

One is typically not interested in *any* tile, but in maximal tiles, i.e., tiles that cannot be extended. For instance, Figure 1 shows a binary dataset and two tiles. The first tile consists of the first and second column together with the first and second row. All entries for these rows and columns are 1s. Furthermore, it cannot be expanded as adding the third column or row would also include 0 values. The second tile consists of all three columns and the third row. Together these two tiles "cover" the whole dataset, that is, all cells with value 1 in the matrix belong to one of the tiles. The area of a set of tiles, denoted as $area(\mathcal{T}, D)$, is the number of cells (7 in our example) in the (union of the) tiles $\mathcal{T}$ occurring in the dataset D

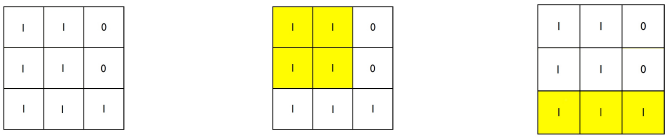


Fig. 1: Example of Boolean tiles and their coverage

Definition 5 (Maximum k-Tiling) Given a binary dataset D and a positive integer k, find a tiling $\mathcal{T}$ consisting of at most k tiles and maximizing $area(\mathcal{T}, D)$.

We now formalize tiling as a relational data factorization problem and then solve it using ASP. Rather than restricting ourselves to Boolean values as in the traditional formulation, we consider the relational case. The standard way of dealing with tables in attribute-value datasets was to expand them into a sparse Boolean matrix (with one Boolean for every attribute-value). In contrast, our formulation employs the attribute-value format directly.

Given a relation $db(Value, Attr, Transct)$, denoting that transaction *Transct* has *Value* for *Attr*, the task is to find a set of tiles that can be applied to the transactions to summarize the dataset *db*. Here, a tile is a set of attribute-value pairs.

	State	Age	Fuel	Type
t1	Fair	Old	Gas	Sport
t2	Good	New	Gas	Sport
t3	Fair	Old	Electric	HEV
t4	Good	New	Electric	HEV

Tile	Items
i1	Fair Old
i2	Gas Sport

Tile	InTransactions
i1	t1 t3
i2	t1 t2

Fig. 2: Relational tiling: two relational tiles (right) in a toy dataset (left) concerning cars.

In Figure 2, for example, we can see the initial dataset, in which *State* is an attribute and *Fair* and *Good* are values for this attribute. Moreover, the blue and green areas indicate two relational tiles occurring in particular sets of transactions.

The two example tiles can be expressed as

$$\textit{tile}(i_1, \textit{fair}, \textit{state}).\ \textit{tile}(i_1, \textit{old}, \textit{age}).\ \textit{in}(i_1, t_1).\ \textit{in}(i_1, t_3).$$
$$\textit{tile}(i_2, \textit{gas}, \textit{fuel}).\ \textit{tile}(i_2, \textit{sport}, \textit{type}).\ \textit{in}(i_2, t_1).\ \textit{in}(i_2, t_2).$$

where the first argument of each *tile* is the index of the tile, the second is the value of the attribute, and the third argument is the name of the attribute. When tile I is applied to a transaction T (i.e., it occurs in the transaction), this is denoted by $\textit{in}(I, T)$. We call a set of tiles a *tiling*.

We would like to factorize the initial dataset, represented as a set of $\textit{db}(\textit{fair}, \textit{state}, t_1)$, $\textit{db}(\textit{old}, \textit{age}, t_1)$, ..., using the following *shape* query:

$$\textit{approx}(\textit{Attr}, \textit{Value}, \textit{Transct}) \leftarrow \textit{tile}(\textit{Indx}, \textit{Value}, \textit{Attr}), \textit{in}(\textit{Indx}, \textit{Transct}). \quad (1)$$

To reason about the coverage of the shape, i.e., which transactions and attributes are covered in the dataset (indicated by color in Figure 2), we use the following definition:

$$\textit{covered}(\textit{Transct}, \textit{Attr}) \leftarrow \textit{approx}(\textit{Attr}, \textit{Value}, \textit{Transct}).$$

For instance, $\textit{covered}(t_1, \textit{age})$ holds because $\textit{tile}(i_1, \textit{old}, \textit{age})$ and $\textit{in}(i_1, t_1)$ hold.

To specify the maximum k-tiling problem, we need the following constraints.

`one-value-attribute`: for every attribute of a tile there is at most one value:

$$\leftarrow \textit{tile}(\textit{Indx}, \textit{Val}_1, \textit{Attr}), \textit{tile}(\textit{Indx}, \textit{Val}_2, \textit{Attr}), \textit{Val}_1 \neq \textit{Val}_2. \quad (2)$$

`no-tile-intersection`: tiles do not overlap in the same transaction

$$\leftarrow \textit{in}(I_1, T), \textit{in}(I_2, T), \textit{tile}(I_1, V, A), \textit{tile}(I_2, V, A), I_1 \neq I_2. \quad (3)$$

`no-overcoverage`: tiles cannot "overcover" the transaction, that is, they are only allowed to cover tuples that are in the dataset;

$$\leftarrow \textit{tile}(\textit{Indx}, \textit{Value}, \textit{Attr}), \textit{in}(\textit{Indx}, \textit{Transct}), \textit{not db}(\textit{Value}, \textit{Attr}, \textit{Transct}). \quad (4)$$

`number-of-patterns(K)`: there are at most k-tiles (numbered from 1 to k):

$$\textit{Indx} = 1 \vee \textit{Indx} = 2 \vee \ldots \textit{Indx} = k \leftarrow \textit{tile}(\textit{Indx}, \textit{Value}, \textit{Attr}).$$

Furthermore, the maximum k-tiling problem searches for the k tiles that maximize the area. This leads to an instance of the *similarity* score defined by

$$\texttt{coverage}: \quad \#\{(T, A) : \textit{covered}(T, A)\}. \quad (5)$$

The statement above correspond to the standard mathematical function optimization notation, that reads as follows: count (#) the cardinality of the set ($\{\cdot\}$) of tuples (T, A) such that (:) $\textit{covered}(T, A)$ holds. When we translate this statements into ASP formulation we have to use special syntax of ASP (*#maximize*) to capture this mathematical formulation.

We specify the high-level model for maximum k-tiling in Listing 4.

Listing 4: Maximum k-Tiling ReDF Model

Input: dataset *db* and constant K
Shape: *approx*(*Attr*, *Value*, *Transct*) $\leftarrow$ *tile*(*Indx*, *Value*, *Attr*), *in*(*Indx*, *Transct*)
Find: the set of ground facts *tile*$(\cdot)$, *in*$(\cdot)$
Satisfying:`no-tile-intersection` $\wedge$ `no-overcoverage`
$\wedge$ `number-of-patterns(K)` $\wedge$ `one-value-attribute`
Maximizing: `coverage`

To illustrate the advantages of our declarative and modular approach, let us consider a small variation of the tiling task, in which tiles may overlap.

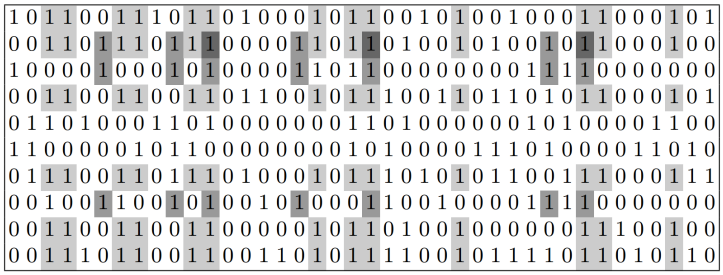


Fig. 3: Example of a 0/1 database with a tiling consisting of two overlapping tiles (darkest shaded area corresponds to the intersection of the two tiles), due to Geerts et al (2004)

Overlapping tiling Figure 3, taken from Geerts et al (2004), illustrates a Boolean dataset with two overlapping tiles. We investigate and present two new variations of maximum k-tiling: overlapping and noisy tiling. The first investigates the global pattern mining task, when the overall coverage is optimized, allowing overlaps between tiles. The second investigates the task when, in k-maximum tiling, a tile can have a number of mismatches as covering a transaction. It is straightforward to change the assumption in our ReDF framework (and the corresponding ASP implementation). For the first task, it only involves replacing the constraint `no-tile-intersection` by the following constraint.

`overlapping-tiles(N)`: two tiles in one transaction can intersect only on at most N attributes:

$$\leftarrow \mathit{in}(I_1, T), \mathit{in}(I_2, T), \mathit{tile}(I_1, V, A_1), \mathit{tile}(I_2, V, A_2), I_1 \neq I_2, \#\{A_1 = A_2\} > N.$$

To model the variation that tolerates some noise in the data, we can replace constraint `no-overcoverage` by

`noisy-overcoverage(N)`: every tile I can overcover at most N attributes in every transaction T where it occurs:

$$\leftarrow \mathit{tile}(I, V, A), \mathit{in}(I, T), \mathit{not\ db}(V, A, T), \#\{A\} > N.$$

Both variations show that a slight change of the formulation of property of a solution leads to a small change in the modeling and to a small change in the implementation.

4.2 The Discrete Basis Problem (DBP) and Boolean Matrix Factorization (BMF)

BMF has been extensively studied by Miettinen (2012), resulting in the well-known ASSO algorithm. Let us now show how it can be expressed as ReDF problem. As a starting point we take the same shape (Eq. 1) as in the tiling example in Subsection 4.1. However, we need to change the constraints to reflect the different properties of the desired solutions: tiles may now overlap, since one is not interested in tiles per se, but in good coverage of the dataset. That is why we remove the `no-tile-intersection` and `no-overcoverage` constraints, and introduce a notion of 'overcoverage' instead, by means of the following definition:

$$\textit{overcovered}(T, A) \leftarrow \textit{approx}(V, A, T), \textit{not db}(V, A, T).$$

In the Discrete Basis Problem, the scoring function maximizes the number of covered elements, while minimizing the overcovered ones. The latter term can be simply defined as:

`overcoverage`: $\#\{(T, A) : \textit{overcovered}(T, A)\}$.

We specify the high-level DBP model in Listing 5.

Listing 5: ReDF Model for the Discrete Basis Problem

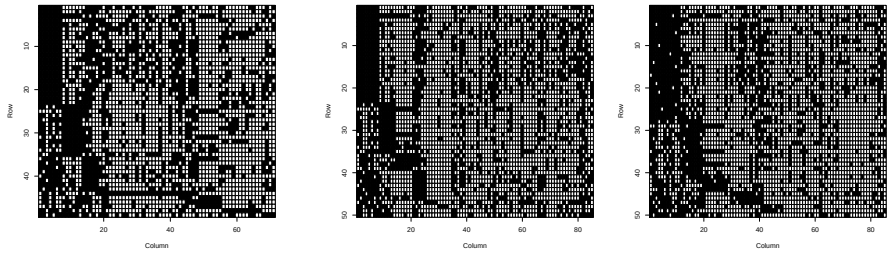
Input: dataset *db* and constants K, α
Shape: $\textit{approx}(\textit{Attr}, \textit{Transct}) \leftarrow \textit{tile}(\textit{Indx}, \textit{Attr}), \textit{in}(\textit{Indx}, \textit{Transct})$
Find: the set of ground facts $\textit{tile}(\cdot), \textit{in}(\cdot)$
Satisfying: `number-of-patterns(K)`
Maximizing: `coverage` $- \alpha \times$ `overcoverage`

This formulation mimics *The Discrete Basis Problem* (Miettinen et al, 2008). That is, K plays the role of the basis size and α mimics the bias towards rewarding covering and penalizing overcovering (the flags `--bonus-covered` and `--penalty-overcovered` in ASSO).

It is well-known that tiling and *Boolean matrix factorization* (BMF) are closely related (Miettinen, 2012). Hence, let us also briefly show how BMF can be realized in our framework. It corresponds to an instance of DBP where only binary values (true and false) are possible and the `no-overcoverage` constraint applies. Hence, it is required that the factorization undercovers the initial dataset, i.e., if there is a 0 in a position in the original dataset, then there cannot be a 1 in the approximation. Therefore, the optimization criterion of DBP is further simplified and we obtain the following BMF model, without overcovering, in Listing 6.

Listing 6: BMF without overcovering

Input: dataset *db* and constant K
Shape: $\textit{approx}(\textit{Attr}, \textit{Transct}) \leftarrow \textit{tile}(\textit{Indx}, \textit{Attr}), \textit{in}(\textit{Indx}, \textit{Transct})$
Find: the set of ground facts $\textit{tile}(\cdot), \textit{in}(\cdot)$
Satisfying: `number-of-patterns(K)` $\wedge$ `no-overcoverage`
Maximizing: `coverage`



(a) Regular **(b)** With penalties **(c)** With penalties and noise

Fig. 4: Re-arranging a matrix in block-diagonal form (Animals dataset): (a) regular, (b) with penalties, (c) with noisy blocks and penalties

4.3 Discriminative k-pattern set mining

A common supervised data mining task is that of discriminative pattern set mining (Knobbe and Ho, 2006). Let $db(Value, Attr, Transct)$ be a categorical dataset, $positive(T)$ ($negative(T)$) be the set of positive (negative) transactions, and k the number of tiles. Then, the task is to find extensions of the relations *tile*(*Indx*,*Value*,*Attr*) and *in*(*Indx*,*Transct*) such that positive and negative transactions are discriminated. A standard interpretation is to find tiles that cover as many positive and as few negative ones as possible (Liu et al, 1998). The only required change in the model concerns the scoring function (and assigning some weight to the errors):

$$\#\{T : covered(T), positive(T)\} - \alpha\#\{T : covered(T), negative(T)\}, \tag{6}$$

where α is a constant that represents the weights for the errors made. It is typically a domain specific parameter (the cost of covering a negative example by a rule, i.e., the false positive cost or a weight of a negative example). Let us denote the coverage of the positive transactions as $\texttt{coverage}^+$ (left set term in Eq. 6) and the coverage of negative as $\texttt{coverage}^-$ (right set term in Eq. 6).

Given that we have no `no-overcoverage` constraint and negative transactions can be covered, the optimization criterion is given by

$$similarity(T) = \texttt{coverage}^+ - \alpha \times \texttt{coverage}^-.$$

This corresponds to the high-level model in Listing 7.

Listing 7: ReDF Discriminative Patter Set Mining Model

Input: dataset *db* and constants K, α
Shape: $approx(Attr, Value, Transct) \leftarrow tile(Indx, Value, Attr), in(Indx, Transct)$
Find: the set of ground facts $tile(\cdot), in(\cdot)$
Satisfying: `number-of-patterns(K)` $\wedge$ `no-overcoverage` $\wedge$ `one-value-attribute`
Maximizing: $\texttt{coverage}^+ - \alpha \times \texttt{coverage}^-$

4.4 Block-diagonal matrix form

Aykanat et al (2002) introduced the problem of and an algorithm for permuting the rows and columns of a sparse matrix into block diagonal form. They relate this prob-

lem to other combinatorial and classical linear algebra problems. The underlying block-diagonal structure of a matrix can be used to parallelize certain matrix computations. An illustration of block-diagonalization (several variants) of the Animals dataset is depicted in Figure 4.

We reduce it to a form of tiling. The shape query is the same as in tiling but the constraints are different: if a tile I_1 has an attribute A, then a tile I_2 cannot use the same attribute. A similar constraint is imposed on the *in* predicate and transactions stating that each item A can belong to only one tile

`item-blocking`: $\leftarrow \mathit{tile}(I_1, A), \mathit{tile}(I_2, A), I_1 \neq I_2.$

Only one tile can occur in a transaction T.

`transaction-blocking`: $\leftarrow \mathit{in}(I_1, T), \mathit{in}(I_2, T), I_1 \neq I_2.$

We also modify the optimization criterion to take into account elements not covered by a tile but blocked by this tile. Every tile that selects attributes and transactions prohibits other tiles to use these attributes and transactions by means of the `item-blocking` and `transaction-blocking` constraints. We penalize excessive usage of attributes and transactions by a single tile. We do this to improve the block form of the matrix, since in this task we are not just interested in a tiling with maximal coverage, but in a tiling that maximizes the number of elements on the diagonal and minimizes the number of elements everywhere else. To enforce this we introduce two functions:

`item-penalty`: $\#\{(T, A) : \mathit{approx}(T, A'),\ \mathit{not\ covered}(T, A)\}$

`transt-penalty`: $\#\{(T, A) : \mathit{approx}(T', A),\ \mathit{not\ covered}(T, A)\}$

Then, the whole problem is formulated in Listing 8.

Listing 8: ReDF Block-Diagonal Model

Input: dataset *db* and constants N, α, β
Shape: $\mathit{approx}(\mathit{Attr}, \mathit{Value}, \mathit{Transct}) \leftarrow \mathit{tile}(\mathit{Indx}, \mathit{Value}, \mathit{Attr}), \mathit{in}(\mathit{Indx}, \mathit{Transct})$
Find: the set of ground facts $\mathit{tile}(\cdot), \mathit{in}(\cdot)$
Satisfying: `transaction-blocking` $\wedge$ `item-blocking`
`one-value-attribute` $\wedge$ `noisy-overcoverage(N)` $\wedge$ `no-tile-intersection`
Maximizing: `coverage` $- \alpha \times$ `transt-penalty` $- \beta \times$ `item-penalty`

If we omit `item-penalty` and `transt-penalty`, we obtain the standard optimization function for tiling. In the experimental section we evaluate the effect of the presence of this penalty.

5 Beyond Classic Problems

So far we have focused on matrix-like representations of the data, in which the dataset was represented by instances of $\mathit{db}(T, A, V)$, for a transactions T having a value V for an attribute A. This representation is independent of the number of attributes and values, it allows one to easily specify constraints over all attributes and to access the data using the predicate *db* only. We will now show that it is also possible to use other, purely relational representations, such as the *sells* example from the Introduction.

Section 2 already provided the *sells* example for decomposing a ternary relation into three binary ones. In the shape for the *sells* example in Listing 3 there is no latent variable: there are only attributes from the original dataset. Since there is no latent variable, there is no "pattern" to be found for which the optimization criterion needs to be optimized, which allowed us to use a simple error function using only one type of atom.

However, latent variables can also be useful in a purely relational setting. Let us illustrate this on an example inspired by the ArXiv community analysis example of Gopalan and Blei (2013). Assume we are given a relation *publishedIn* with attributes *Author*, *University*, and *Venue*, specifying that an author belonging to a particular university publishes in a particular venue. Furthermore, assume we want to factorize this relation into the relation *approx (A,U,V)* by introducing a latent attribute *Topic*, denoted as T. The latent topic variable clusters authors, universities and venues together in such a way that their join results in publications.

We obtain the following high-level model in Listing 9, where α is the constant that indicates the relative cost of overcovering an element and the integer constant k is the number of value that the latent variable (T) can take:

$$\textit{approx}(A, U, V) \leftarrow \textit{interestedIn}(A, T), \textit{specializedIn}(U, T), \textit{inField}(V, T).$$

Listing 9: ReDF Purely Relational

Input: dataset *db* and constants K, α
Shape: $\textit{approx}(A, U, V) \leftarrow \textit{interestedIn}(A, T), \textit{specializedIn}(U, T), \textit{inField}(V, T)$.
Find: the set of ground facts $\textit{interestedIn}(\cdot), \textit{specializedIn}(\cdot), \textit{inField}(\cdot)$
Satisfying: `number-of-patterns(K)`
Maximize: `coverage` $- \alpha \times$ `overcoverage`

The corresponding model without latent variables would be different only in the decomposition shape, i.e., it would look like

$$\textit{approx}(A, U, V) \leftarrow \textit{worksAt(A,U), publishesAt(A,V), knownAt(U,V)}.$$

Discriminative relational learning In the spirit of discriminative pattern mining, described in Subsection 4.3, we can also do discriminative learning in the purely relational setting. To do so, we assume that the relation has an extra argument Co-Author and we would like to discriminate the dataset by a particular Co-Author c^+, i.e.,

$$\begin{aligned} &\texttt{coverage}^+(A, U, V) \leftarrow \textit{approx}(A, U, V), \textit{publishedIn}(A, U, V, c^+). \\ &\texttt{coverage}^-(A, U, V) \leftarrow \textit{approx}(A, U, V), \textit{publishedIn}(A, U, V, C), C \neq c^+. \end{aligned} \tag{7}$$

Then, the optimization criterion remains the same as in Subsection 4.3. Intuitively, if we have only information about an author of a paper (together with his or her university affiliation and a venue), we use this to 'predict' his or her co-author using the patterns we obtain in this discriminative setting.

6 Implementation

This section describes how ReDF models can be implemented in ASP. We do this for the basic problem of tiling, as well as for the purely relational data factorization presented before. Implementations of the other variations are included in Appendix C. Our primary implementation is written in clasp, can be used with the clasp system (Gebser et al, 2012; Brewka et al, 2011) and will be made available online upon acceptance of this manuscript.

6.1 General computation methods: greedy and sampling approaches.

In all described problems, the goal is to find k patterns or tiles, where a pattern is interpreted as a set of facts corresponding to a particular value of the latent variable. We will follow an iterative approach to finding these patterns, in which the discovery of the next pattern or tile will be encoded in ASP. We will consider both a *greedy* and a *sampling* algorithm for realizing this. The sampling approach is intended for better scalability and will be evaluated in Section 7.1.

Greedy model. The greedy approach is described in Algorithm 1. Essentially, when the next best *pattern* has been computed (where *pattern* is a set of facts associated with the *pattern* identifier, e.g., in tiling a pattern is a set of transactions and attributes), it is added to the current set of *patterns*. The specific part for each tile is represented by `executeProgram` and is encoded separately in ASP. Note that this greedy, iterative approach to finding k patterns is very common in pattern mining. Theoretical bounds on the solution quality of the greedy approach have been studied in the context of the maximum k-set coverage problem (Hochbaum and Pathria, 1998; Feige, 1996); more details can be found in Appendix F.

Algorithm 1: Greedy execution model

input : *data* is the dataset
output: *patterns* is the set of patterns
patterns $\leftarrow \emptyset$;
for $i \in [1, k]$ **do**
  pattern $\leftarrow$ `executeProgram`(*data*, *patterns*, i);
  patterns $\leftarrow \{pattern\} \cup patterns$;

Column sampling execution model. To improve scalability, we employ a sampling approach. Interestingly, our approach is different from most existing sampling techniques in data mining: instead of sampling a rows or patterns, we sample columns. Algorithm 2 presents the column sampling approach we propose. The key difference with the greedy approach is that instead of determining the next best pattern on the *overall* dataset in each iteration, this approach samples N subsets of the data and determines the next best pattern for all of these subsets. The best among these is then fixed, and the process is repeated. We empirically evaluate the effects of sampling on the quality of the computed patterns and on the runtime in the experiment section. Quality bounds for this type of greedy search have also been analyzed previously (Hochbaum and Pathria, 1998); for more details we refer to Appendix F.

Algorithm 2: Column sampling execution model

```
input  : data is the dataset
input  : N is the number of samples
input  : α is the relative size of a sample
output: patterns is the set of patterns
patterns ← ∅;
for i ∈ [1, k] do
    maxPattern ← ∅ ;
    for j ∈ [1, N] do
        sample ← getColumnSample(data, α);
        pattern ← executeProgram(sample, patterns, i);
        if score(pattern) > score(maxPattern) then
            maxPattern ← pattern ;
    patterns ← {maxPattern} ∪ patterns ;
```

Listing 10: Greedy maximum k-tiling formalization in answer set programming

```
1  %one-value-attribute; it generates at most one value per attribute
2  0 { tile(currentI, Value, Attribute) : valid(Attribute, Value) } 1 :− col(Attribute).
3  %no-overcoverage
4  over_covered(currentI,T) :− not db(Value, Attribute, T), tile(currentI, Value, Attribute), transaction(T).
5  %no-tile-intersection
6  intersect(T) :− currentI != Index, tile(currentI, Value, Attribute), tile(Index, Value, Attribute), in(Index,T).
7  %defines presence of tiles in transactions
8  in(currentI,Transct) :− transaction(Transct), not over_covered(currentI, Transct), not intersect(Transct).
9  %defines coverage function
10 covered(Transct, Attribute) :− in(Index,Transct), tile(Index, Value, Attribute).
11 #maximize[covered(Transct, Attribute)].
```

6.2 Data mining problems expressed in the framework

The maximum k-tiling problem can be encoded in answer set programming as indicated in Listing 10. The code implements the greedy model, i.e., Algorithm 1, for the maximum k-tiling problem with a fixed number of tiles (Geerts et al, 2004). It assumes we have already found an optimal tiling for $n-1$ tiles, and indicates how to find the n-th tile to cover the largest area. The n-th tile is called *currentI* in the listing. Further, we have information about the names of the attributes and the possible values for each attribute through predicates *col*(*Attr*) and *valid*(*Attr*, *Value*). That is, *col*(A) is an unary predicate that encodes possible column indices, and *valid*(A, V) is a binary predicate that encodes which possible values V can occur in column A.

Let us explain the code in Listing 10. The constraint in Line 2 generates at most one value for each attribute. The constraints in Lines 4 and 6 compute the transactions where the current tile cannot occur, i.e., `intersect(T)` is the set of all transactions where the current tile overlaps with another tile and the current tile cannot cover these transactions. Similarly, `overcovered(currentI,T)` is the set of transactions that cannot be covered because there is an element in the current tile, with fixed index `currentI`, that is not present in transaction `T`. The constraint in Line 8 states that if the tile does not violate the overcovering and intersection constraints in a transaction, it occurs in the transaction. Line 10 defines the coverage and the optimization constraint in Line 11 enforces the selection of the best model.

Theorem 1 (Correctness of the greedy ASP tiling encoding) *The ASP program $\mathcal{P}$ defined by the Listing 10 computes the k-th largest tile w.r.t. the scoring function* `coverage` *(5) as extensions of the predicates tile$(k,\cdot,\cdot)$ and in$(k,\cdot)$ in its answer set $\mathcal{A}$, provided that the dataset is represented extensionally through the predicates db, valid, and col and the $k-1$ already found tiles are represented extensionally through the predicates tile$(I,\cdot,\cdot)$ and in$(I,\cdot)$ for $I \in [1,k-1]$.*

For the proof, see Appendix B. The clasp encodings for the other models are sketched in Appendix C.

6.3 Purely relational data factorization

In Section 5 we presented a factorization of the *publishedIn* relation into three binary relations. It constitutes a proof-of-concept prototype model in ASP and could be improved by, e.g., incorporating heuristics.

The general structure of the ASP encoding is similar to the *sells* example in Listing 3: we indicate here only a possible optimization for the relation generators. We use the left-to-right order of the atoms in the schema (replicated below) while generating candidates for the factorization.

Listing 11: Generators for the model without a latent variable into three binary relations

```
1 0 { works_at(A,U)     } 1 :- published_in(A,U,V).
2 0 { publishes_at(A,V) } 1 :- published_in(A,U,V), works_at(A,U).
3 0 { known_at(U,V)     } 1 :- published_in(A,U,V), works_at(A,U), publishes_at(A,V).
```

Implementation differences. When we generalize the factorization encoding with two relations to three relations, we observe a slight implementation difference between them. Factorization with the two relation shapes can be naturally implemented using the core ASP generate-and-test paradigm. Once we have guessed an extension for a certain value of the latent variable, we propagate it to the second relation and test against the constraints. This strategy is often deployed in specialized algorithms (Geerts et al, 2004; Miettinen et al, 2008). For a multiple relation shape we guess an extension of one relation, then we constrain the possible values we generate for the second value (e.g., see Line 2 in Listing 11). In general, we can search for one at a time using a greedy strategy (as in tiling). Theoretically, we can simultaneously search for values of a latent variable by replacing the fixed latent parameter by a variable and searching over the latent parameter as well. The work of Guns et al (2013b) provides evidence that this approach does not scale well, unless special propagators are introduced into the solver. This technique would allow extending the method to other shapes with more than three relations.

7 Experiments

The main goal of this section is to evaluate whether ReDF problems can be solved using a generic solver. In particular, we focus on solving the problem formulations as we specified them in ASP. We investigate whether the problems can be solved, and for

a number of tasks compare the results and runtimes to those obtained by specialized algorithms. Since we here use generic problem formulations and generic solvers that have neither been designed nor optimized for the tasks under consideration, we cannot expect the approach to be as efficient as specialized algorithms. However, what is more important is that we demonstrate that all tasks formalized and prototyped using the ReDF framework can be solved using a unified approach.

Experimental setup and datasets. The ASP engine we use is 64-bit clingo (clasp with the gringo grounder) version 3.0.5 with the parameter `--heuristic=Vmtf` (see Appendix A for details on the parameters) and all experiments are executed on a 64-bit Ubuntu machine with Intel Core i5-3570 CPU @ 3.40GHz x 4 and 8GB memory, except for Maximum k-tiling on Chess and Mushrooms datasets where Intel Xeon CPU with 128GB of memory (all single-threaded) has been used due to high memory requirements. For most experiments we use the datasets summarized in Table 1, which all but one originate from the UCI Machine Learning repository (Bache and Lichman, 2013). The *Animals (with Attributes)* dataset was taken from Osherson et al (1991). For the purely relational factorization task, the data and experiment results are described separately in the corresponding subsection.

In Subsection 7.1 we show how ReDF formulations of existing data mining tasks (from Section 4) can be solved using the implementation presented in Section 6, afterwards in Subsection 7.2 we show the results of the purely relational data factorization task. The ASP solver parameters used in the experiments and a breakdown of individual solving steps and their runtimes determined by the meta-experiment are presented in Appendix A.

7.1 Solving existing tasks

Maximum k-Tiling in Categorical Data We first consider the Maximum k-Tiling problem from Section 4.1 and present timing and coverage results in Table 2 obtained on all datasets from Table 1.

In all cases the problem specification given in Listing 10 was used to greedily mine $k = 25$ tiles. Since the problem becomes more constrained as the number of tiles increases, runtime decreases for each additional tile mined. We therefore report total runtime and coverage for different values of k, i.e., for different total numbers of tiles. Only $k = 10$ tiles were mined on Chess and Mushroom due to long runtimes.

Effect of sampling As we can see from Table 2a, runtimes are quite long on datasets like Mushroom. To address this issue, we use the sampling procedure of

Table 1: Dataset properties. For each dataset, we specify whether the attributes have Boolean or categorical domains, the number of tuples and attributes, and the average number of distinct values per attribute

Dataset	Attributes	# tuples	# attributes	Avg # values per attribute
Animals	Boolean	50	85	2
Solar flare	categorical	1 389	11	3.3
Tic-tac-toe	categorical	958	10	2.9
Nursery	categorical	12 960	8	3.4
Voting	categorical	435	17	3.0
Chess (Kr-vs-Kp)	categorical	3 196	36	2.1
Mushroom	categorical	8 124	22	5.6

Table 2: Maximum k-Tiling

(a) Runtime

Dataset	Number of tiles (k) 5	10	15	20	25
Animals	36s	1m4s	1m21s	1m32s	1m36s
Solar flare	6s	10s	13s	16s	18s
Tic-tac-toe	22s	31s	33s	34s	35s
Nursery	4m19s	6m32s	7m32s	7m56s	8m13s
Voting	52s	1m28s	1m42s	1m46s	1m49s
Chess	17h03m	22h31m	-	-	-
Mushroom	13h09m	19h44m	-	-	-

(b) Coverage

Number of tiles (k) 5	10	15	20	25
0.327	0.472	0.573	0.649	0.709
0.416	0.565	0.655	0.721	0.751
0.251	0.449	0.623	0.784	0.907
0.269	0.454	0.634	0.773	0.905
0.399	0.553	0.662	0.749	0.810
0.483	0.618	-	-	-
0.476	0.586	-	-	-

Algorithm 2 with the following parameters: $\alpha = 0.4$ and $N = 20$, i.e., 40% of all attributes were selected uniformly at random for each sample and 20 samples were used. Intuitively, the larger the sample size and the more samples, the better we approximate the exact result.

With the given parameters, we attain an order of magnitude improvement in runtime: instead of 19 hours with the regular algorithm, using sampling it takes only one hour to compute 10 tiles as indicated in Figure 5a. The effect of using sampling on coverage can be seen in Figure 5b: the first tiles that are mined have lower coverage than when sampling is not used, but after a while the difference in coverage with LTM-k remains more or less constant and even slightly decreases. LTM-k is the original, specialized tiling algorithm, to which we compare next.

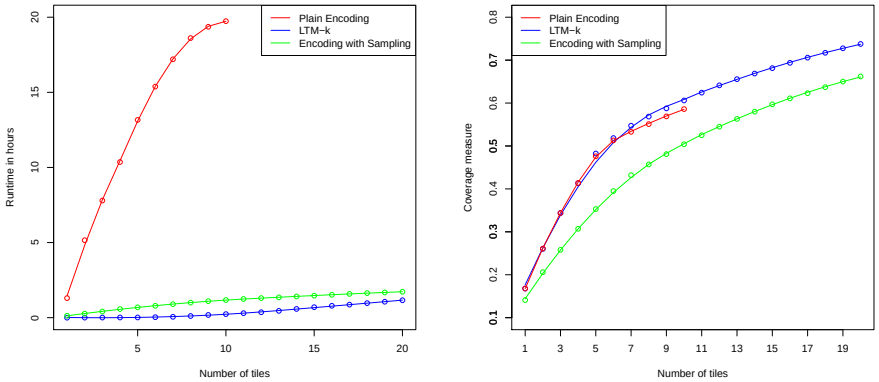
Comparison to a specialized algorithm We now compare the performance of the ASP-based implementation of LTM-k greedy strategy to that of a specialized implementation[2]. Figures 5a and 5b present both runtime and coverage comparisons obtained on Mushroom, both for our approach (with and without sampling) and the specialized miner.

Without sampling, we can see that our approach gives the same results in terms of the coverage as the LTM-k algorithm. This is as expected though, since both LTM-k and our approach guarantee to find an optimal solution in each iteration. The slight difference between the two coverage curves in Figure 5b is caused by the fact that multiple tiles can have the same (maximum) area, and some choice between those has to be made. Although these choices are typically made deterministically, the different implementations make decisions based on different criteria, resulting in slightly different tilings.

Unfortunately, the ASP solver is not as efficient as the specialized miner as can be seen in Figure 5a, and the generality of the approach comes at the cost of longer runtimes. However, as already discussed, using a sampling approach can substantially decrease the runtime. Experiments on other datasets showed similar behavior to that depicted here.

Overlapping tiling To evaluate the overlapping tiling task from Subsection 4.1, we apply the model in Listing 12 (ASP encoding in Appendix C) to the five smaller datasets from Table 1. We experiment with two levels of overlap, i.e., parameter N is set to either 1 or 2: tiles can intersect on at most one or two attribute(s). As the results

[2] `http://people.mmci.uni-saarland.de/~jilles/prj/tiling/`



(a) Runtime **(b)** Coverage

Fig. 5: Tiling comparison (runtime, coverage) with LTM-k (Mushroom dataset)

Table 3: Maximum k-Tiling with overlap. The maximal allowed overlap is limited by parameter N

(a) Runtime

		Number of tiles (k)				
Dataset	**N**	**5**	**10**	**15**	**20**	**25**
Animals	1	1m10s	2m28s	3m46s	4m24s	4m47
	2	1m39s	4m10s	6m26s	7m40s	8m10s
Solar flare	1	8s	13s	17s	21s	24s
	2	8s	15s	20s	25s	29s
Tic-tac-toe	1	24s	41s	49s	52s	53s
	2	23s	43s	51s	55s	56s
Nursery	1	5m00s	8m19s	10m10s	10m48s	11m12s
	2	5m43s	9m32s	11m9s	11m50s	12m12s
Voting	1	1m10s	2m19s	2m53s	3m8s	3m15s
	2	1m39s	3m34s	4m35s	5m9s	5m33s

(b) Coverage

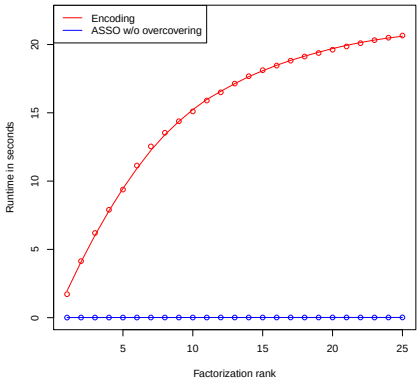
Number of tiles (k)				
5	**10**	**15**	**20**	**25**
0.327	0.475	0.583	0.663	0.722
0.332	0.482	0.592	0.675	0.742
0.433	0.595	0.684	0.734	0.756
0.452	0.602	0.685	0.731	0.755
0.253	0.451	0.626	0.781	0.898
0.253	0.451	0.626	0.781	0.898
0.268	0.454	0.633	0.772	0.905
0.268	0.454	0.633	0.772	0.905
0.403	0.558	0.675	0.765	0.828
0.409	0.571	0.683	0.762	0.819

in Table 3 show, allowing limited overlap can lead to a small increase in coverage, but runtimes also increase due to the costly aggregate operation in Line 1 of Listing 12.

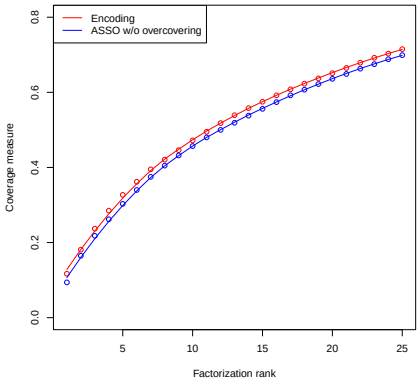
However, what is important to emphasize here is that only a small change in the problem formalization is sufficient to allow for overlap in the tilings, while the solver can still solve the problem without any further changes. And although the runtimes are longer when more overlap is allowed, the difference with the basic, non-overlapping setting is moderate.

Boolean matrix factorization (BMF) We perform Boolean matrix factorization (Section 4.2) by applying the formalization of Listing 13 and compare the results to those obtained by ASSO[3] (Miettinen, 2012) with the no-overcoverage flag (`-P1000`). The factorization rank k is incremented by one in each iteration, and meanwhile coverage gain and runtime are measured. The results for Animals are presented in Figure 6 and show that coverage is almost identical to that obtained by ASSO. Again, this is unsurprising, as our implementation follows the same solving strategy. However, runtimes are several times higher, which is due to the usage of a general solver that is not optimized for this type of task. Results obtained on other datasets are very similar and are therefore not presented here.

[3] `http://www.mpi-inf.mpg.de/~pmiettin/src/DBP-progs/`

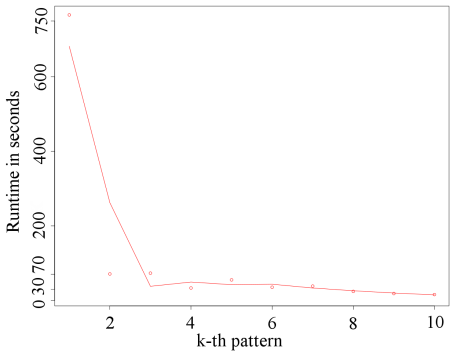


(a) Runtime



(b) Coverage

Fig. 6: Boolean matrix factorization on datasets Animals. Runtime and coverage are depicted for different factorization ranks.



(a) Runtime (in s) to mine k-th discriminative pattern on Chess dataset ($\alpha = 1$, i.e., positive and negative tuples are weighted equally)

(b) Discriminative mining coverage on Chess and Tic-tac-toe datasets ($\alpha = 1$, i.e., positive and negative tuples are weighted equally)

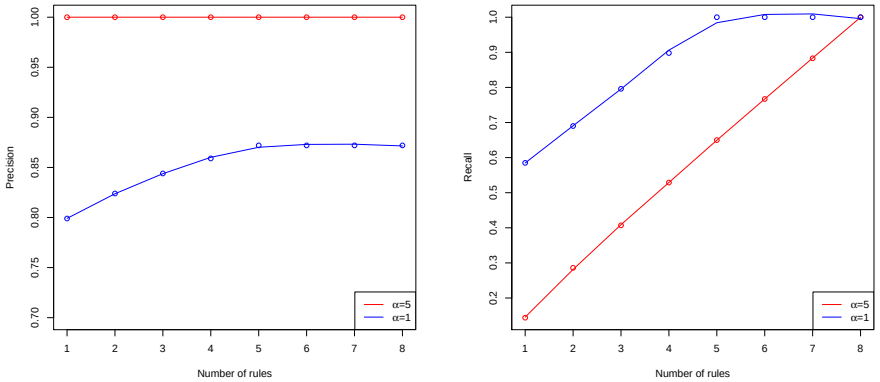
	Tic-tac-toe ($k = 5$)	Chess ($k = 10$)
Covered −	92 (27.7%)	160 (7%)
Covered +	626 (100%)	864 (95.5%)
Difference	534	704
Runtime	0.52s	18m48s

Fig. 7: Discriminative pattern set mining summary: runtime (left) and coverage (right)

Discriminative pattern set mining Here we demonstrate how the discriminative k-pattern mining model from Section 4.3 can be solved. For this we use Chess and Tic-tac-toe from Table 1, each of which has a binary class label indicating whether a game was won or not and can therefore be naturally used for this task.

We apply the encoding from Listing 14 to both datasets, set $\alpha = 1$ to weigh positive and negative tuples equally, and summarize the results in Figure 7b. The results show that five patterns suffice to cover all positive examples of Tic-tac-toe, hence mining more than five patterns would be useless. 92 of the 718 covered tuples are negative, i.e., 12.8%, while 34.7% of the tuples in the complete dataset is negative. For Tic-tac-toe, the time needed to solve this task is very limited: about half a second.

Figure 7a shows the runtime needed to iteratively find subsequent patterns in the Chess dataset. Interestingly, it seems that the problem becomes substantially easier (computationally) once the first few patterns have been found: the runtime per pattern



(a) Precision **(b)** Recall

Fig. 8: Discriminative pattern set mining (Tic-tac-toe dataset): precision (left) and recall (right) for different α, i.e., for varying weights of covering negative transactions

drops heavily. This confirms that the search space shrinks when the problem becomes more constrained, i.e., the number of answer sets decreases with the addition of more constraints.

We next show the influence of the α parameter, i.e., the relative weight of covering positive and negative tuples in the optimization criterion. By increasing α, the 'penalty' for covering a negative tuple is increased and the algorithm can be forced to select more conservative rules. We investigate the effect of this parameter by measuring and comparing precision and recall of the obtained pattern sets for $\alpha = 1$ and $\alpha = 5$. Figure 8 shows that precision goes to 1 when α is increased, while recall is decreased but this can be compensated by mining a larger number of patterns[4].

This task differs from the previous one in its optimization criterion: positive coverage penalized by negative coverage allows for fast inference and discovery of the optimal solution, which results in shorter runtimes than for tiling.

Matrix block-diagonal form We apply three versions of the encoding to the Animals dataset (Osherson et al, 1991). The results presented in Figure 4 demonstrate that the Animals dataset can be re-arranged into block-diagonal form using our proposed framework. The runtime in all experiments are on the order of seconds. Parameters used in the experiments were $\alpha = \frac{3}{20}$ and $\beta = \frac{1}{20}$. Figure 4c demonstrates the model from Section 4.4, with the same α, β and $N = 1$. The low-level encoding of this model is given in Listing 15.

7.2 Purely relational data factorization

In Section 5 we described how to model the factorization of *publishedIn(Author, University, Venue)* into three binary relations with a latent variable *Topic*. We now evaluate whether the standard ASP solver can solve this task. Unfortunately, we cannot

[4] Appendix D presents the data points of Figure 8 as a traditional precision-recall plot.

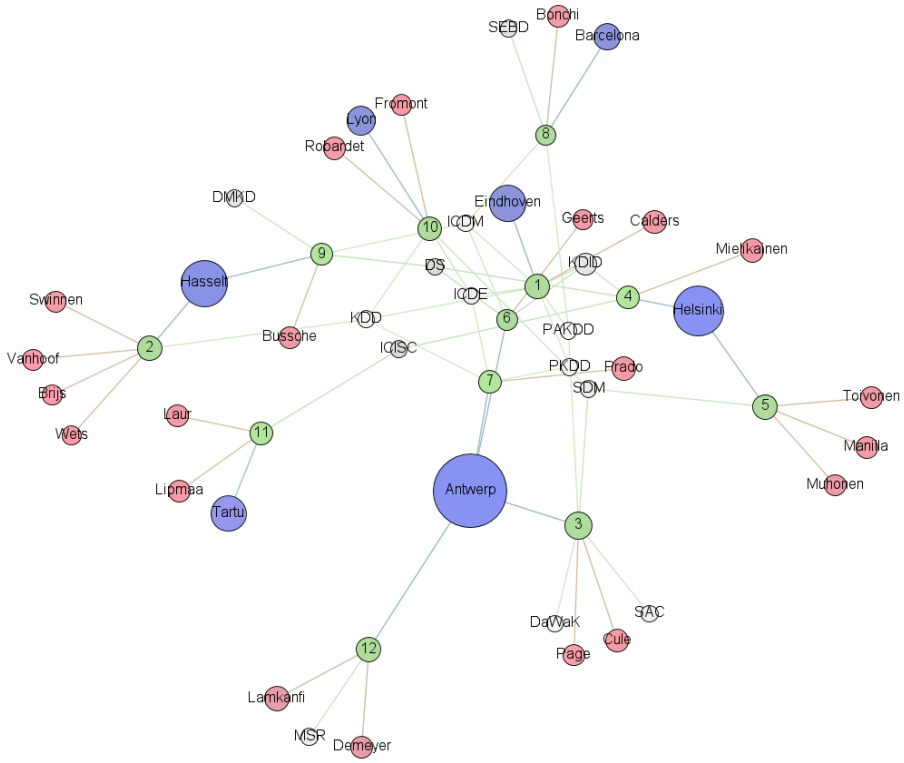


Fig. 9: Clustering in topics by ReDF. Red nodes represent co-authors, blue their university cities, white nodes venues and green topics that bind them together. If there is an edge between a topic and a node, then there is a corresponding element in the relation (i.e., *interestedIn*, *specializedIn* or *inField*)

expect a generic solver to handle enormous datasets such as the one from ArXiv as described by Gopalan and Blei (2013). Instead, we demonstrate a proof-of-concept of solving the model in Listing 16 on a moderate dataset.

We constructed a dataset for a well-known colleague from the data mining community: Bart Goethals (Antwerp University). We collected his publication list from Microsoft Academic Search[5] and extracted for each paper the publication venue, and all co-authors together with their corresponding affiliations (i.e., the last known affiliation for each author in this list of papers). Each unique combination of venue, co-author, and affiliation resulted in a tuple in the *publishedIn* relation. The complete dataset contains 57 tuples over 19 universities, 38 authors, and 15 venues.

Intuitively, if a set of *authors* from a set of *universities* publish in a set of *venues*, then there must be an underlying research *topic* that unites them. Hence, by factorizing the relation into three separate relations, we cluster each of the entity types into a (fixed) number of topics, as indicated by the value of the latent *Topic* variable.

The results for factorization using $K = 12$ topics and $\alpha = \frac{1}{2}$ are presented in Figure 9, including co-authors (red), universities (blue), publication venues (purple),

[5] http://academic.research.microsoft.com/Author/2266478/bart-goethals

and topics (green). To determine the number of topics K, we tracked the optimization criterion while increasing K and stopped when this no longer improved.

Since the task is of an exploratory character, we can only qualitatively evaluate the results. We observe that all data mining venues are located together in the center, connected to the same topics. SEBD, an Italian database conference, stands apart, and there is also a separate block for database and computing venues DaWaK and SAC. Manual inspection of the results indicates the topics (or clusters) to be coherent and meaningful: they represent different affiliations and groups of co-authors that Bart Goethals has collaborated with. For example, topic 5 contains the SDM conference, the University of Helsinki, and three co-authors specialized in Data Mining. Hence, this topic could be described as "Data mining collaboration with the University of Helsinki", which makes perfect sense as Bart Goethals was previously a researcher in Helsinki.

Not all authors are represented in the factorization. How much of the *publishedIn* dataset is covered depends on the number of topics K (which was chosen as described before). The higher the cardinality of the pattern set, the larger the total coverage. The *covered* elements positively contribute to coverage, whereas the *overcovered* elements contribute negatively. This implies that each pattern is chosen such that the number of *covered* and *overcovered* elements are balanced and the optimization criterion is maximized. In general, covering all authors with few patterns would lead to significant overcovering of the original dataset, while introducing too many patterns would create clusters with only one author (which is clearly undesirable, since these clusters would not be meaningful).

The decompositions, as the one depicted in Figure 9, could serve as a basis for new analyses. For example, we might visualize the intersection of common (latent) topics shared by two researchers. We outline possible examples in Appendix E.

Relational factorization without a latent variable. In Section 5, we also described a factorization that does not use any latent variables (analog to the *sells* example in Listing 3 from the introduction section). We evaluate this model using Listing 11 on the same dataset as used in the previous experiment, i.e., the co-author relation *publishedIn(Author, Uni, Venue)* for Bart Goethals.

In general, factorizations do not perfectly match the original relation (i.e., *error* $\neq 0$), but in this particular case the system found a lossless solution. It is easy to see that this will not always be possible though. For instance, let us assume we keep multiple affiliations per author in the dataset. For example, apart from a fact *p(bonchi,barcelona,pakdd)*, there may be another fact *p(bonchi, pisa, pakdd)* in *publishedIn*. Although the same factorization would be found by the solver, the found solution would be imperfect as the latter fact is not represented in the factorized relation.

Solving this task was computationally easy, since there is no latent variable to iterate over: the runtime was only $0.01s$. Table 4 presents a summary.

Relational discriminative learning Here we investigate discriminative learning in the purely relational setting, for which we use the discriminative optimization criteria described in Section 5. For this experiment we collected DBLP data for two well-known

Table 4: Experimental summary for pure relational factorizations from Subsection 5

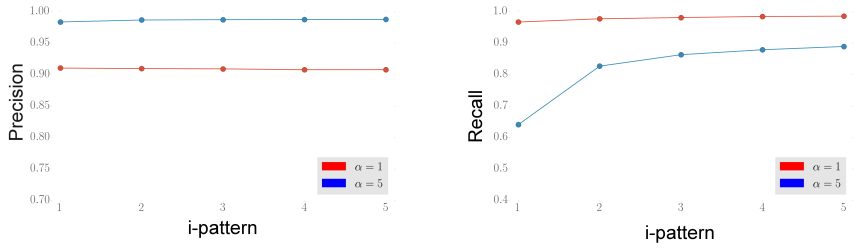
With a latent variable	**#Transactions**	**Overall runtime**	**Avg runtime**	**#Topics**	**Avg atoms per topic**
	58	14s	1.1s	12	5.4
Without a latent variable	**#Transactions**	**Overall runtime**	**Correct**	**#Incorrect**	**Avg factor size**
	58	0.01s	58	0	45

researchers in the field of data mining: Jiawei Han and Philip S. Yu. In this example all publications belonging to either researcher are considered a class. Since DBLP data does not have authors affiliations, we replace this attribute with the publication date converted to a categorical variable $M \in \{old, recent, new\}$, using the following rule: if the date is later than 2010, it is represented as "new"; if it is between 2005 and 2010, then it is "recent"; otherwise it is "old". The complete dataset contains around 6000 ground atoms of the following form: *paper(Author,Age,Venue,Han\Yu)*. The goal is to predict whether the paper is co-authored by Han or Yu based on author, venue and age using discriminative rules as defined in Eq. 7. In this experiment the shape is

$$approx(A, M, V) \leftarrow author(A, T), paperAge(M, T), venue(V, M),$$

where T is a latent variable. As for the previous discriminative experiment, in Figure 10 we present an overview of the dependency of precision and recall on the number of patterns and α, the penalty for covering the incorrect class. Runtimes are similar as in Figure 7a. ASP finds an optimal solution in half an hour and then spends a substantial amount of time to prove it is indeed the best solution. We therefore used a time limit of one hour per pattern speedup the computation. This limit was only reached in the computation of the first pattern, for both values of α.

In Figure 10 we see that, depending on the number of patterns and penalty for covering the wrong class (α), we can obtain a classifier with different precision and recall[6].



(a) Precision **(b)** Recall

Fig. 10: Discriminative learning in the purely relational setting; precision (left) and recall (right) for different α, i.e., for two weights of covering negative atoms.

7.3 Runtime discussion

In this section we have seen a number of experiments that solve ReDF problems using generic solving technology, i.e., answer set programming. As we can see in Figures

[6] Appendix D presents the data points of Figure 10 as a traditional precision-recall plot.

5a and 6, specialized algorithms are substantially faster than ASP. On datasets of moderate size, however, generic solvers obtain reasonable runtimes, as indicated by the results in Tables 2a, 3a, and 7b, and Figures 6 and 7a. For the purely relational data factorization task from Section 5 we present a summary of the experiments in Table 4. In these experiments, computation time ranged from several seconds to few minutes.

8 Related Work

Our work is related to 1) previous work on generalizing problem definitions and solutions in factorization, 2) existing forms of relational decomposition, and 3) approaches in inductive and abductive logic programming, and 4) the use of declarative languages and solvers for data mining.

8.1 General models for pattern mining

Our work can be related to a number of approaches that have generalized some of the tasks addressed in Section 5. Lu et al (2008) used BMF as a basis for defining several data mining tasks and modeled them using integer linear programming. While Lu et al (2008) also used a general purpose solver, it is restricted to Boolean matrix products involving only two Boolean matrices. In a similar manner Li (2005) defined a *General Model for Clustering Binary Data*, using matrix factorization to model several well-known clustering methods. The framework supports only one possible factorization shape, a lower-level modeling language, and requires complete partitions as well as specialized algorithms for different problems. In our approach, the shape of the factorization is separated from the constraints and optimization criterion.

Biskup et al (2004); Fan et al (2012) investigated inverse querying and the problem of solving relational equations $e_1(D) = e_2(D)$ exactly under several assumptions, that could be used to compute exact solutions to a restricted form of ReDF. However, this approach does not seem to allow for approximations and the use of loss functions.

8.2 Decomposition of databases, tensors, and real-valued matrices

ReDF is related to several forms of *relational decomposition*, a term that has been heavily overloaded in the literature. Hence, it is imperative that we present an overview and contrast existing paradigms to our own work. Moreover, ReDF is also related to decomposition methods for real-valued matrices.

Relational decomposition in database theory. Ever since the seminal paper by Codd (1970), the decomposition of relations has been an important theme in database research (Koehler, 2007; Date, 2006). Key properties of this form of relational decomposition are (Elmasri and Navathe, 2010): 1) a relational schema together with its constraints, e.g., functional dependencies, is assumed given; 2) decomposition is

never based on the data (extension), but only on the schema (intension); 3) decomposition is always lossless, i.e., factorization is always exact for any possible extension, and never an approximation. An interesting exception is *Relational Decomposition via Partial Relations* (Berzal et al, 2002), where one is looking for partially satisfied dependencies in the data and then uses these partial dependencies to derive a normal form. It does take into account data, but only to mine additional schema constraints in the decomposition.

Relational decomposition in tensor calculus. Kim and Candan (2011) extend classical tensor factorization, CP decomposition, to deal with datasets composed of several relations, i.e., CP is generalized to multi-relational datasets. This requires adding relational algebra operations to CP. Key differences are: the data consists of several tables, with a schema to join them at the end; the shape is always the same and a tensor is decomposed into a sum of terms having the same structure; the optimization function is fixed; no user constraints are supported.

Decomposition of real-valued matrices. Let us start with SVD (Singular Value Decomposition) (Golub and Van Loan, 1996), the best-known method in this area, which gives an optimal rank-k decomposition of a real-valued matrix A into a composition of three matrices $U\Sigma V^T$, where U and V are orthogonal real-valued matrices and Σ is a diagonal non-negative matrix with singular values of A. One of the key problems with SVD in the context of relational and Boolean factorization is that U and V may contain negative values, which make interpretation in the relational setting problematic. To overcome this issue NNMF (Non-Negative Matrix Factorization) has been introduced (Paatero and Tapper, 1994). Still, there are two key issues with the usage of NNMF and SVD for relational and Boolean data.

First of all, for a Boolean matrix A there is no clear relation between its real value rank, denoted $\text{rank}_{\mathcal{R}}(A)$, and its Boolean rank, denoted $\text{rank}_{\mathcal{B}}(A)$, ($\text{rank}_{\mathcal{R}_{\geq 0}}(A)$ denotes the non-negative rank). We know that the following inequalities hold (Miettinen, 2009):

$$\begin{aligned} &\text{rank}_{\mathcal{R}}(A) \leq \text{rank}_{\mathcal{R}_{\geq 0}}(A) \\ &\text{rank}_{\mathcal{B}}(A) \leq \text{rank}_{\mathcal{R}_{\geq 0}}(A) \end{aligned} \tag{8}$$

Furthermore, there are examples where $\text{rank}_{\mathcal{R}}(A) = n$ and $\text{rank}_{\mathcal{B}}(A) = \log(n)$ (where A is $n \times n$ matrix) (Miettinen, 2009), which implies that there are cases where Boolean factorization could be preferable over real-valued matrix factorization, i.e., there are cases where we can obtain a smaller decomposition if we use discrete rather than real-valued methods. Also, if we look at approximate ranks (when the approximation is set within ϵ), Equation 8 above does not hold, i.e., there is no clear connection between NNMF and Boolean ranks in the approximate case.

Secondly, existing real-valued matrix factorization methods do not support multiple relations in the decomposition shape and extra constraints in the decomposition, which is at the core of the ReDF method. Furthermore, the constraints used in our method are hard constraints over discrete values. The latter problem has been addressed by Collective Matrix Factorization (Singh and Gordon, 2008), which allows to handle multiple relations and optimization criteria. However, at its core the method relies on stochastic optimization over reals, which leads to the problems discussed at the beginning of the section.

Finally, as ReDF is defined over discrete values in the presence of the hard constraints, all the problems described above (rank inequalities, optimization over reals, uninterpretable values, etc) apply to the comparison of ReDF with real-valued matrix factorizations as well.

8.3 Relational learning

ReDF is also related to some well known techniques in inductive logic programming and statistical relational learning and even to abductive reasoning.

Several frameworks for *abduction* have been introduced over the years (Denecker and Kakas, 2002; Flach and Kakas, 2000). In abduction, the goal is to find a (minimal) hypothesis in the form of a set of ground facts that explains an observation. Abductive reasoning uses a rich background theory as well as integrity constraints; it also uses a set of clauses defining the predicate in the observation. The differences with ReDF are that ReDF uses a much simpler shape definition and no real background theory. On the other hand, abductive reasoning proceeds in a purely logical manner, and typically does neither take into account multiple facts in the observation nor does it use complex optimization functions. There also exist similarities between ReDF and fuzzy abduction (Vojtás, 1999; Miyata et al, 1995), but we differ in the core assumptions we make: all rules and constraints in our setting are deterministic, as well as the evidence that needs to be derived. Also, ReDF has the shape constraint, which allows to derive only specific explanations in a form of a factorization.

Meta-interpretive learning (Muggleton et al, 2015) uses templates together with a kind of abductive reasoning to find a set of rules and facts in a typical inductive logic programming setting. While it can use much richer templates and background theory, it uses neither constraints nor optimisation functions like ReDF does.

Kok and Domingos (2007) introduced a probabilistic framework based on Markov logic together with the EM principle to realize statistical predicate invention. This captures what the authors call *multiple relational clustering* and addresses essentially the same task as the *infinite relational model* of Kemp et al (2006). Statistical predicate invention shares several ideas with our approach: it employs a kind of query or schema to denote the kind of factorization one wants and also imposes some hard constraints on the possible solutions. On the other hand, its optimization criterion is built-in and based on the maximum likelihood principle, the framework seems restricted to a kind of block modeling approach, essentially clustering the different rows and columns into different blocks, and the approach is inherently probabilistic.

8.4 Declarative data mining

The idea of using generic solvers and languages for data mining is not new and has been investigated by, for instance, Guns et al (2011, 2013a); Métivier et al (2012), who used various constraint programming languages for modeling and solving itemset mining problems. The use of ASP for frequent item set and graph mining were investigated in Järvisalo (2011) and Paramonov et al (2015). Furthermore, the use

of integer linear programming is quite popular in data mining and machine learning; e.g., Chang et al (2008). While the choice of a particular framework for modeling and solving may lead to both different models and performances, it should be possible to use alternative frameworks, such as constraint programming or integer programming, for modeling and solving ReDF problems.

Aftrati et al (2012) extended the typical structure of the mining problem using three-level graphs that represent a chain of relations in the multi-relational setting: authors writing papers, and papers being about certain topics. The goal is to find the subgraphs that satisfy particular constraints and optimization criteria. E.g., an author is an authority if the number of topics he has written papers on is maximal. They provide various interesting discovery tasks and solve them using integer programming.

9 Conclusions

The key contribution of this paper is the introduction of the framework of relational data factorization, which was shown to be relevant for modeling, prototyping, and experimentation purposes.

On the modeling side, we have formulated several well-known data mining tasks in terms of ReDF, which allowed us to identify commonalities and differences between these data mining tasks. One advantage of the framework is that small changes in the problem definition typically lead to small changes in the model. Furthermore, ReDF allowed us to model new types of relational data mining problems.

We have not only modeled problems, but also demonstrated that these models can be easily translated into concrete executable ASP encodings. The experiments have shown the feasibility of the approach, especially for prototyping, and especially with the sampling technique. The runtimes were typically not comparable with highly optimized and much more specific implementations that are typically used in data mining. Still they could be run on reasonable datasets of modest size (e.g., Mushroom and Chess have approximately $185\,000$ and $115\,000$ non-empty elements respectively).

Directions for future research include investigating the use of alternative solvers (such as constraint or integer programming), the study of heuristics and local search, and the expansion of the range of tasks to which ReDF can be applied. For example, a general ReDF framework is needed to factorize evidence for probabilistic lifted inference, where the shape of the factorization crucially affects the overall performance of the algorithm (Van den Broeck and Darwiche, 2013).

Acknowledgments. We would like to thank Marc Denecker, Tias Guns, Benjamin Negrevergne, Siegfried Nijssen, and Behrouz Babaki for their help and assistance, and last but not least the ICON project (FP7-ICT-2011-C) and FWO for funding this work.

References

Aftrati F, Das G, Gionis A, Mannila H, Mielikäinen T, Tsaparas P (2012) Mining chains of relations. In: Holmes DE, Jain LC (eds) Data Mining: Foundations

and Intelligent Paradigms, Intelligent Systems Reference Library, vol 24, Springer Berlin Heidelberg, pp 217–246

Arimura H, Medina R, Petit JM (eds) (2012) Proceedings of the IEEE ICDM Workshop on Declarative Pattern Mining

Aykanat C, Pinar A, Catalyurek ÜV (2002) Permuting sparse rectangular matrices into block-diagonal form. SIAM Journal on Scientific Computing 25:1860–1879

Bache K, Lichman M (2013) UCI machine learning repository. URL `http://archive.ics.uci.edu/ml`

Berzal F, Cubero JC, Cuenca F, Medina JM (2002) Relational decomposition through partial functional dependencies. Data Knowl Eng 43(2):207–234

Biskup J, Paredaens J, Schwentick T, den Bussche JV (2004) Solving equations in the relational algebra. SIAM Journal on Computing 33(5):1052–1066

Brewka G, Eiter T, Truszczyński M (2011) Answer set programming at a glance. Communications of the ACM 54(12):92–103

Chang MW, Ratinov LA, Rizzolo N, Roth D (2008) Learning and inference with constraints. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, pp 1513–1518

Codd EF (1970) A relational model of data for large shared data banks. Commun ACM 13(6):377–387

Date CJ (2006) Date on Database: Writings 2000-2006. Apress, Berkely, CA, USA

De Raedt L (2008) Logical and relational learning. Cognitive Technologies, Springer

De Raedt L (2012) Declarative modeling for machine learning and data mining. In: The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, pp 2–3

De Raedt L (2015) Languages for learning and mining. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA., pp 4107–4111

Denecker M, Kakas A (2002) Abduction in logic programming. In: Kakas A, Sadri F (eds) Computational Logic: Logic Programming and Beyond, Lecture Notes in Computer Science, vol 2407, Springer Berlin Heidelberg, pp 402–436

Eiter T, Ianni G, Krennwallner T (2009) Answer set programming: A primer. In: 5th International Reasoning Web Summer School (RW 2009), Brixen/Bressanone, Italy, August 30 – September 4, 2009, Springer, LNCS, vol 5689

Elmasri R, Navathe SB (2010) Fundamentals of Database Systems, Sixth Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Fan W, Geerts F, Zheng L (2012) View determinacy for preserving selected information in data transformations. Information Systems 37(1):1–12

Feige U (1996) A threshold of ln n for approximating set cover. In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, STOC '96, pp 314–318

Flach P, Kakas A (2000) 4:1–33

Fürnkranz J, Scheffer T, Spiliopoulou M (eds) (2006) Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, September 18-22, 2006, Proceedings, Lecture Notes in Computer Science, vol 4213, Springer

Gebser M, Kaminski R, Kaufmann B, Schaub T, Schneider M, Ziller S (2011a) A portfolio solver for answer set programming: Preliminary report. In: Delgrande J, Faber W (eds) Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11), Springer-Verlag, Lecture Notes in Artificial Intelligence, vol 6645, pp 352–357

Gebser M, Kaufmann B, Kaminski R, Ostrowski M, Schaub T, Schneider M (2011b) Potassco: The potsdam answer set solving collection. AI Communications 24(2):107–124

Gebser M, Kaminski R, Kaufmann B, Schaub T (2012) Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers

Gebser M, Kaufmann B, Romero J, Otero R, Schaub T, Wanko P (2013) Domain-specific heuristics in answer set programming. In: desJardins M, Littman ML (eds) Association for the Advancement of Artificial Intelligence, AAAI Press

Geerts F, Goethals B, Mielikäinen T (2004) Tiling databases. In: Discovery Science, Springer, pp 278–289

Golub GH, Van Loan CF (1996) Matrix Computations (3rd Ed.). Johns Hopkins University Press, Baltimore, MD, USA

Gopalan PK, Blei DM (2013) Efficient discovery of overlapping communities in massive networks. Proceedings of the National Academy of Sciences 110(36):14,534–14,539

Guns T, Nijssen S, De Raedt L (2011) Itemset mining: A constraint programming perspective. Artif Intell 175(12-13):1951–1983

Guns T, Dries A, Tack G, Nijssen S, De Raedt L (2013a) Miningzinc: A modeling language for constraint-based mining. In: International Joint Conference on Artificial Intelligence, Beijing, China

Guns T, Nijssen S, De Raedt L (2013b) k-pattern set mining under constraints. Knowledge and Data Engineering, IEEE Transactions on 25(2):402–418

Guns T, Nijssen S, De Raedt L (2013c) k-pattern set mining under constraints. IEEE Transactions on Knowledge and Data Engineering 25(2):402–418

Heath IJ (1971) Unacceptable file operations in a relational data base. In: Proceedings of the 1971 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control, ACM, New York, NY, USA, SIGFIDET '71, pp 19–33

Hochbaum DS, Pathria A (1998) Analysis of the greedy approach in problems of maximum k-coverage

Järvisalo M (2011) Itemset mining as a challenge application for answer set enumeration. In: Logic Programming and Non-Monotonic Reasoning, pp 304–310

Jones TH, Song IY, Park EK (1996) Ternary relationship decomposition and higher normal form structures derived from entity relationship conceptual modeling. In: Proceedings of the 1996 ACM 24th Annual Conference on Computer Science, ACM, New York, NY, USA, CSC '96, pp 96–104

Kemp C, Tenenbaum JB, Griffiths TL, Yamada T, Ueda N (2006) Learning systems of concepts with an infinite relational model. In: Proceedings of the 21th National Conference on Artificial Intelligence, AAAI Press, pp 381–388

Kim M, Candan KS (2011) Approximate tensor decomposition within a tensor-relational algebraic framework. In: Proceedings of the 20th ACM International

Conference on Information and Knowledge Management, ACM, New York, NY, USA, CIKM '11, pp 1737–1742

Knobbe AJ, Ho EKY (2006) Pattern teams. In: Fürnkranz et al (2006), pp 577–584

Koehler H (2007) Domination normal form: Decomposing relational database schemas. In: Proceedings of the Thirtieth Australasian Conference on Computer Science - Volume 62, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, ACSC '07, pp 79–85

Kok S, Domingos P (2007) Statistical predicate invention. In: Proceedings of The 24th International Conference on Machine Learning, pp 433–440

Leone N, Pfeifer G, Faber W, Eiter T, Gottlob G, Perri S, Scarcello F (2002) The dlv system for knowledge representation and reasoning. ACM Transactions on Computational Logic 7:499–562

Li T (2005) A general model for clustering binary data. In: ACM SIGKDD, ACM, New York, NY, USA, pp 188–197

Lifschitz V (2008) What is answer set programming? In: Association for the Advancement of Artificial Intelligence, pp 1594–1597

Liu B, Hsu W, Ma Y (1998) Integrating classification and association rule mining. In: ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp 80–86

Lu H, Vaidya J, Atluri V (2008) Optimal boolean matrix decomposition: Application to role engineering. In: IEEE 24th ICDE, pp 297–306

Métivier JP, Boizumault P, Crémilleux B, Khiari M, Loudni S (2012) A constraint language for declarative pattern discovery. In: Ossowski S, Lecca P (eds) Proceedings of the ACM Symposium on Applied Computing, pp 119–125

Miettinen P (2009) Matrix decomposition methods for data mining: computational complexity and algorithms. Department of Computer Science, series of publications A, report A-2009-4, University of Helsinki 2009 (Ph.D. thesis, monograph)

Miettinen P (2012) Dynamic boolean matrix factorizations. In: Zaki MJ, Siebes A, Yu JX, Goethals B, Webb GI, Wu X (eds) Proceedings of International Conference on Data Mining, IEEE Computer Society, pp 519–528

Miettinen P, Mielikäinen T, Gionis A, Das G, Mannila H (2008) The discrete basis problem. IEEE Transactions on Knowledge and Data Engineering 20(10):1348–1362

Miyata Y, Furuhashi T, Uchikawa Y (1995) A study on fuzzy abductive inference. In: Proceedings of 1995 IEEE International Conference on Fuzzy Systems, Citeseer, vol 1, pp 337–342

Muggleton S, De Raedt L (1994) Inductive logic programming: Theory and methods. J Log Program 19/20:629–679

Muggleton SH, Lin D, Tamaddoni-Nezhad A (2015) Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. Machine Learning 100(1):49–73

Osherson D, Stern J, Wilkie O, Stob M, Smith E (1991) Default probability. Cognitive Science 15(2):251–269

Paatero P, Tapper U (1994) Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. Environmetrics 5(2):111–126

Paramonov S, van Leeuwen M, Denecker M, De Raedt L (2015) An exercise in declarative modeling for relational query mining. In: International Conference on Inductive Logic Programming, ILP, Kyoto, 20-22 August 2015, Springer

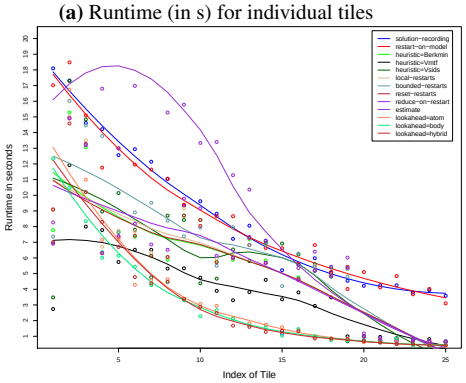
Singh AP, Gordon GJ (2008) Relational learning via collective matrix factorization. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 650–658

Van den Broeck G, Darwiche A (2013) On the complexity and approximation of binary evidence in lifted inference. In: The Neural Information Processing Systems, pp 2868–2876

Vojtás P (1999) Fuzzy logic abduction. In: Proceedings of the EUSFLAT-ESTYLF Joint Conference, Palma de Mallorca, Spain, September 22-25, 1999, pp 319–322

A Evaluating solver performance

(a) Runtime (in s) for individual tiles



(b) Runtime (in s) for N tiles

	Number of tiles				
Search option	**5**	**10**	**15**	**20**	**25**
solution-recording	76	131	168	193	213
restart-on-model	75	127	164	191	211
heuristic=Berkmin	50	86	112	133	137
heuristic=Vmtf	**36**	**64**	**83**	**95**	**100**
heuristic=Vsids	54	88	119	138	140
local-restarts	49	87	113	134	138
bounded-restarts	59	101	132	154	158
reset-restarts	48	85	111	132	136
reduce-on-restart	48	89	115	136	139
estimate	86	169	213	237	241
lookahead=atom	**51**	**71**	**81**	**86**	**88**
lookahead=body	**43**	**63**	**72**	**76**	**79**
lookahead=hybrid	**48**	**68**	**77**	**81**	**84**

Fig. 11: Determining the best ASP solver parameters for tiling with the model of Listing 10: (a) the runtime (in s) needed for mining each subsequent individual tile, (b) runtime summary (in s) for first 25 tiles

This appendix briefly evaluates the performance of the solver that was used for the experiments. We first describe how we tuned the system and then continue with an analysis of the solving process.

Solver tuning. The clingo system (Gebser et al, 2011b) has a variety of parameters that affect runtime. We performed a series of preliminary experiments to determine the parameters that were used for all other experiments in this section. These tuning experiments mostly concerned maximum k-tiling on datasets of moderate size, but some test experiments on other tasks did not reveal any discrepancies.

Runtime was measured for a large number of search options while fixing the dataset and task. Figure 11 shows the results obtained for maximum k-tiling on the Animals dataset. Search options "heuristic=Vmtf" and the "lookahead" options result in shorter runtimes, but the "lookahead" options do not scale well; with these options the system is unable to handle bigger datasets like Mushroom and Chess. Therefore, all remaining experiments in this section have been performed with "heuristics=Vmtf". None of the other combinations of parameters gave any substantial improvement in runtime.

Grounding-solving analysis Besides measuring the time needed for the solver to provide an answer, it is also useful to look at the time needed for the individual execution steps. In case of answer set programming, solving a problem consists of two main steps: 1) grounding and 2) solving (or, alternatively, searching).

Table 5 presents results obtained for the maximum k-tiling problem, i.e., the time needed to compute a single tile, averaged over the first 15 tiles, split into grounding and solving time. For many ASP programs

Table 5: Maximum k-tiling: grounding and solving – avg time per tile (s), and their time ratio

Dataset	Grounding	Solving	Ratio – Solving/Grounding
Nursery	2.173	38.185	18
Voting	0.052	8.350	161
Animals	0.020	2.728	136
Tic-tac-toe	0.124	1.969	16
Flare	0.225	0.575	3

the grounding step is the bottleneck, but the results clearly demonstrate that this is not the case for our ReDF tasks: the ratio between solving and grounding time is generally large. (Due to long runtimes this experiment was not performed for Chess and Mushroom.)

The large ratios can be explained by the fact that we deal with optimization problems, as is usual in data mining. In the presence of an exponential number of answer sets and a binary predicate as the optimization criterion, it is natural to expect the solving part to be the computationally most intensive step.

The results suggest that, if we would like to speed-up ReDF in ASP, we would have to focus on making the general solving process more efficient, possibly by using properties of the ReDF framework. Specialized mining algorithms, e.g., for tiling, demonstrate that faster solving is possible, but translating these efficient mechanisms to an ASP solver, which works completely differently, seems far from trivial.

Alternatives to manual parameter tuning. claspfolio is a portfolio system for clasp (Gebser et al, 2011a). However, our experiments indicated that none of the precomputed models of claspfolio applied to the problems considered in this work and the system switches to the default solver. *hclasp* (Gebser et al, 2013) introduces heuristics into the reasoning process of clasp. Since we consider a wide class of problems, it requires a nontrivial decision procedure to find the right heuristics for a particular task. We regard it as a possible direction for future research.

B Correctness proof of encoding 10

We present the proof of Theorem 1 (Section 6).

Theorem (Correctness of the ASP tiling encoding) *The ASP program $\mathcal{P}$ defined by the listing 10 computes the k-th largest tile w.r.t. the scoring function* `coverage` *(5) as extensions of the predicates tile$(k,\cdot,\cdot)$ and in$(k,\cdot)$ in its answer set $\mathcal{A}$, provided that the dataset is represented extensionally through the predicates db, valid, and col and the $k-1$ already found tiles are represented extensionally through the predicates tile$(I,\cdot,\cdot)$ and in$(I,\cdot)$ for $I \in \overline{1,k-1}$.*

Proof We prove the theorem by means of mathematical induction. The encoding here implicitly refers to the encoding in Listing 10.

We first prove the statement for the case when $k = 1$, i.e., the program $\mathcal{P}$ computes the first largest tile by the coverage function in Eq. 5.

Step 1: k = 1

First of all, the answer set $\mathcal{A}$ always exists, since there is no cyclic dependency involving negation and an "empty" answer set w.r.t. to the predicates *tile* and *in* can be selected as a possible answer set.

Secondly, observe that the extension of the predicate *intersect*$(\cdot)$ is empty, since there is no other tile computed before and the rule in Line 6 never applies. Then, let us show that $\mathcal{A}$ satisfies the other two conditions from the constraints *Cs* and it is the largest tile w.r.t. to coverage.

Assume that the following constraint from *Cs* is violated

$$\leftarrow \mathit{tile}(\mathit{Indx}, \mathit{Value}_1, \mathit{Attr}), \mathit{tile}(\mathit{Indx}, \mathit{Value}_2, \mathit{Attr}), \mathit{Value}_1 \neq \mathit{Value}_2.$$

Then there exist v_1, v_2, a such that $v_1 \neq v_2$, $\mathit{tile}(1, v_1, a) \in \mathcal{A}$, $\mathit{tile}(1, v_2, a) \in \mathcal{A}$. Then the body of the rule in Line 2 is satisfied, since a is a column in the dataset. Then, the head must be satisfied: v_1 and v_2 are valid values, so we derive that

$$0\ \{\mathit{tile}(1, v_1, a), \mathit{tile}(1, v_2, a)\}\ 1,$$

which contradicts the assumption that $v_1 \neq v_2$, $\mathit{tile}(1, v_1, a) \in \mathcal{A}$, $\mathit{tile}(1, v_2, a) \in \mathcal{A}$. Then, since v_1, v_2, a were fixed but arbitrary the constraint in Eq. 2 is satisfied by $\mathcal{A}$.

Assume that the following constraint from *Cs* is violated by $\mathcal{A}$

$$\leftarrow \textit{tile}(\textit{Indx}, \textit{Value}, \textit{Attr}), \textit{in}(\textit{Indx}, \textit{Transct}),\ \textit{not db}(\textit{Value}, \textit{Attr}, \textit{Transct}).$$

There exist v, a, t such that $\textit{tile}(1, v, a) \in \mathcal{A}$ and $\textit{in}(1, t) \in \mathcal{A}$, but $db(v, a, t) \notin \mathcal{A}$. Then the rule in Line 4 applies, since the body of it satisfied by the assignment $\theta = \{T \mapsto t, Value \mapsto v, Attribute \mapsto a\}$ (and guess is equal to 1). Then, *overcover(t)* $\in \mathcal{A}$. Then the body of the rule in Line 8 is not satisfied and $\textit{in}(1, t) \notin \mathcal{A}$. Then, we derived contradiction between the facts that $\textit{in}(1, t) \in \mathcal{A}$ and $\textit{in}(1, t) \notin \mathcal{A}$, therefore, $\mathcal{A}$ satisfies the constraint in Eq, 4.

Assume that $\mathcal{A}$ is not a maximal answer set by the coverage function, than there is an answer set $\mathcal{A}'$ such that it satisfies all the constraints from *Cs* and the cardinality of the extension of the predicate *covered* is higher than in $\mathcal{A}$. If so, then $\mathcal{A}$ violates the optimization criterion in Line 11. Then, $\mathcal{A}$ is maximal w.r.t. the coverage function.

Step 2: k-1. Assume, we have proven the theorem for the values up to $k - 1$.

Step 3: k. Let us show that theorem holds for the k-th tile, for $k > 1$. The proof is very similar to the reasoning presented above. We assume that $\mathcal{A}$ violates some of the contraints from *Cs* and demonstrate "reductio ad absurdum". Here we demonstrate the only different reasoning part from the previous case of the step one.

Assume that "intersection" constraint in Eq. 3 from *Cs* is violated:

$$\leftarrow \textit{in}(I_1, T), \textit{in}(I_2, T), \textit{tile}(I_1, V, A), \textit{tile}(I_2, V, A), I_1 \neq I_2.$$

Then, there exist i, v, t, a such that $i \neq k$ and $\textit{in}(i, t) \in \mathcal{A}$, $\textit{in}(k, t) \in \mathcal{A}$, $\textit{tile}(i, v, a) \in \mathcal{A}$, $\textit{tile}(k, v, a) \in \mathcal{A}$. Then the body of the rule in Line 6 is satisfied, hence *intersect(t)* $\in \mathcal{A}$. Then, the body of the "in"-rule in Line 8 is not satisfied and $\textit{in}(k, t) \notin \mathcal{A}$. We obtain contradiction between two facts: our assumption that $\textit{in}(k, t) \in \mathcal{A}$ and the conclusion that $\textit{in}(k, t) \notin \mathcal{A}$. □

C Additional ASP encodings

Overlapping tiling. Constraint `overlapping-tiles(N)` is encoded in ASP as indicated in Listing 12 (we indicate only the new lines in the listing). Let us explain how this new constraint works. The rule in Line 1 for a transaction T computes the attributes *Attr* on which the current tile intersects with the previous tiles. The constraint in Line 2 defines the transaction where the current tile cannot occur in the following way: if the number of attributes where the current tile intersect with the other is higher than *overlap_level*, then the tile cannot occur in this transaction.

Listing 12: Overlapping tiling

```
1 intersect_N(T,Attr) :− guess != Indx, tile(currentI, V, Attr), tile(Indx, V, Attr), in(Indx,T).
2 intersect(T) :− overlap_level #count{ intersect_N(T,Indx) : col(Indx) }, transaction(T).
```

We see that a slight change of the formulation of property of a solution leads to a small change in the modeling and also to a small change in the implementation.

BMF. Boolean data is a particular case of attribute-value data, and corresponds to the situation when values can be either true or false. It is straightforward to realize the encoding of tiling for BMF by deleting all the arguments in Listing 10 corresponding to values; we encode only the true facts for *db(T,I)*. Also, as described in Subsection 4.2, we remove the rules corresponding to the intersection of tiles. The ASP code is in Listing 13.

Listing 13: Boolean Matrix Factorization

```
1 0 { tile(currentI,I) } 1 :− item(I).
2 over_cover(currentI, T) :− not db(T,I), tile(currentI,I), transaction(T).
3 in(currentI,T) :− not over_cover(currentI,T), transaction(T).
4 covered(T,I) :− tile(currentI,I), in(currentI,T), db(T,I).
5 #maximize[covered(T,I)].
```

The structure and functions of the constraints is the same as in Listing 10. Except for the modification described above, where we remove "value" argument from the predicates in the encoding and non-intersection constraint.

Discriminative k-pattern set mining. In case of discriminate k-pattern set mining, we need to change the optimization criterion and evaluation function to capture positive and negative transactions. The changed optimization criterion 6 can be straightforwardly implemented in ASP, see Listing 14.

Listing 14: Discriminative k-pattern set mining

```
1 in(currentI,T) :− transaction(T), not over_covered(currentI, T).
2 covered_plus(T)  :− in(Indx,T), tile(Indx, Value, Attribute), positive(T).
3 covered_minus(T) :− in(Indx,T), tile(Indx, Value, Attribute), negative(T).
4 #maximize[covered_plus(T) = 1, covered_minus(T) = −α].
```

This encoding distinguishes between the positively and negatively covered transactions, and weights negative covered transactions by α.

Matrix block-diagonal form. The encoding mainly mimics the tiling encoding. We indicate here the constraints that are different in Listing 15.

Listing 15: Encoding block diagonal form

```
1  %generate possible tile using only not used items
2  0 { tile(currentI,I) } 1 :− item(I), not usedI(I).
3  %coverage takes into account noise level
4  over_covered(I,T) :− not db(T,I), code(currentI,I), transaction(T).
5  too_noisy(T)      :− noise_level #count{ over_covered(I,T) : item(I) }, transaction(T).
6  covered(T,I) :− in(C,T), code(C,I).
7  %current tile can occur only in not used transactions
8  in(currentI,T) :− transaction(T), not too_noisy(T), not usedT(T).
9  %penalty for used transactions
10 leftT(T,I) :− in(currentI,T), db(T,I), not covered(T,I).
11 %penalty for used columns
12 leftI(T,I) :− code(currentI,I), db(T,I), not covered(T,I).
13 %optimization takes into account penalties
14 #maximize[ covered(T,I)=1, leftT(T,I)=α, leftI(T,I)=−β].
```

The constraints are similar to the described above in tiling, overlapping tiling and discriminative k-pattern set mining. Rules in Lines 10 and 12 introduce the notation of "not used" but blocked by the tile transactions and items. In the optimization criterion 14 each of these elements is penalized.

Purely relational factorization with a latent variable. The factorization shape is defined as

$$\mathit{approx}(A,U,V) \leftarrow \mathit{interestedIn}(A,\mathit{Topic}),\mathit{specializedIn}(U,\mathit{Topic}),\mathit{field}(V,\mathit{Topic}).$$

The candidates are generated by the following rules:

Listing 16: Generator rules

```
1 0 { interestedIn(A,T)  } 1 :− p(A,_,_), topic(T).
2 0 { specializedIn(U,T) } 1 :− p(A,U,_), interestedIn(A,T), topic(T).
3 inField(V,T) :− p(A,U,V), interestedIn(A,T), specializedIn(U,T), topic(T).
4 approx(Author, Uni, Venue) :− interestedIn(Author, G), specializedIn(Uni, G), inField(Venue, G).
```

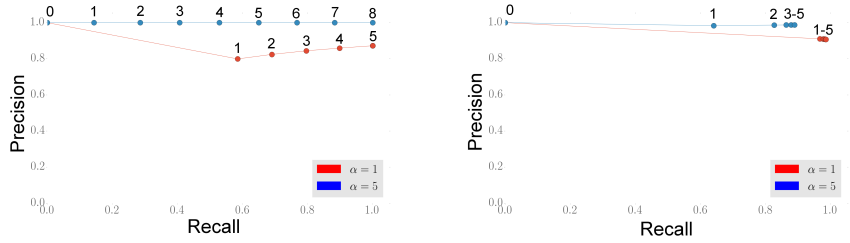
The idea is that we first consider for every author whether he is interested in the topic or not, and then this process is repeated for every university (having a corresponding author interested in the topic), and finally, it is determined whether the venue belongs to the topic based on the relations *interestedIn* and *publishedIn*. The optimization criterion is essentially the same as that of BMF.

Listing 17: Optimization criterion

```
1  covered(A,U,V) :− approx(A,U,V),    p(A,U,V).
2  overcovered(A,U,V) :− approx(A,U,V), not p(A,U,V).
3  #maximize[covered(A,U,T) = 1, overcovered(A,U,T) = −α].
```

D Precision-Recall Curves

Figure 12 presents the precision-recall curves corresponding to Figures 8 and 10. Figure 12a corresponds to the panel in Figure 8 and Figure 12b corresponds to the panel in Figure 10. Note that these precision-recall curves have been obtained by varying the number of rules.



(a) Tic-tac-toe discriminative pattern set mining **(b)** Purely relational discriminative mining

Fig. 12: Precision recall curves for discriminative learning for different α, i.e., for two weights of covering negative atoms. For different number of learned rules, indicated on each point (starting from zero).

E Visualizations for an Application of Purely Relational ReDF

Figures 13 and 14 visualize an example application of purely relational factorization. Figure 13 shows the intersection of latent topics depicted in Figure 9 with publication data of Floris Geerts, while Figure 14 shows the intersection of the same latent topics with the publication data of Pauli Miettinen.

The figures show that we can relate not only particular data points such as venues and co-authors between scientists, but also common (latent) topics. They are identified and visualized in an interpretable manner. The intersection includes the publication records (indicated in red) of two authors (the other author is indicated in pink, if he/she belongs to co-authors of the original decomposition). If for a latent variable, there are a co-author, a university and a venue that belong to both authors, then the latent topic is labeled (in blue) as common.

F Maximum k Set Coverage Analysis of Greedy Strategy

We here elaborate on the bounds of the greedy and sampling strategies as described in Algorithms 1 and 2, when applied to the problems we consider in this work.

In the greedy case of Algorithm 1, the bound analysis is similar to LTM-k and based on the analysis of the greedy approach to general maximum k-coverage (Hochbaum and Pathria, 1998; Feige, 1996). Greedy search optimizes its result in each iteration, given the solutions from the previous iterations, until k solutions have been found. Hence, this algorithm follows the general maximum k-coverage strategy and the factor $\frac{greedy}{optimal} \leq 1 - \frac{1}{e}$ applies.

The key assumption highlighted by Hochbaum and Pathria (1998) is that for a set selected in the i^{th} iteration, no data point can be counted twice, i.e., in two different sets, towards the maximal weight summary. That is, if a data point is present in the sets selected in previous iterations, then the total weight should be the same as if we excluded this data point from one of the sets; the solutions should be non-overlapping. This is the case for the problems we consider (tiling, discriminative rule mining, tiling-based matrix diagonalization, etc).

Furthermore, the maximum k-set cover bound is very general and in practice, the greedy solution is way closer to the optimum than suggested by the bound. For example, Guns et al (2013c) presented a constraint model of optimal tiling and based on this it was possible to compute the optimal k-tiling for $k = 2$ for the smallest datasets and for $k = 3$ for two datasets (a time-out was set to 6 hours). The average

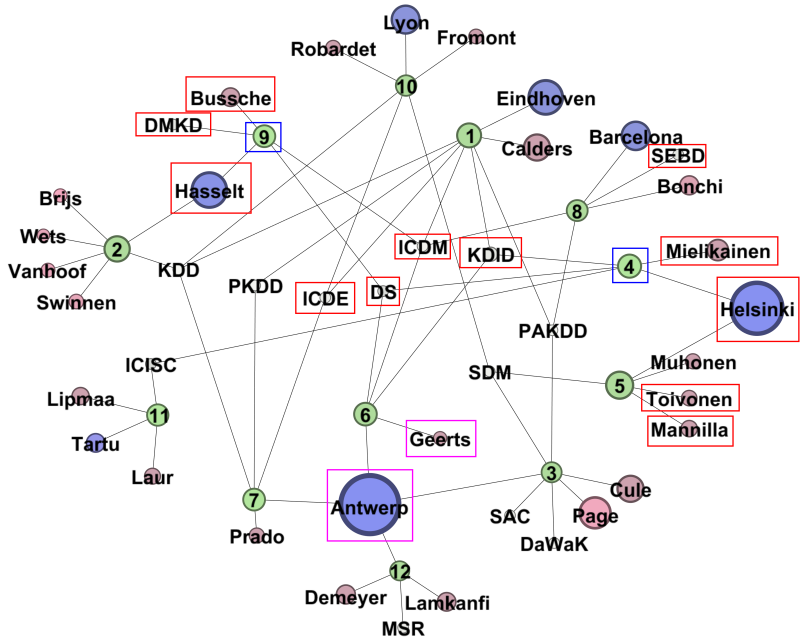


Fig. 13: An application of purely relational factorization: intersection of publication data (in red) and latent topics (in blue) of Bart Goethals with publications of his co-author Floris Geerts (highlighted in pink) in the graph. A latent topic belongs to an intersection if both have the same co-author, university and venue in the publication records.

difference between optimal and actual area across all datasets was around 1%, while the bound suggests that it might deviate up to 36.7%.

The sampling strategy of Algorithm 2 follows the approximate greedy scheme: on each iteration it greedily finds an approximation of a 'greedily optimal' solution. As has been shown by Hochbaum and Pathria (1998), the approximate greedy schema with approximation parameter β (i.e., a solution found at i-th iteration is within a β constant factor of the greedy optimal one at i-th iteration) is within $1 - \frac{1}{e^\beta}$ from the optimal solution.

α, the parameter controlling the number of columns to sample, influences both runtime and coverage, i.e., solution quality. It is, however, extremely hard—if not impossible—to analytically derive the dependency between sample size α and approximation factor β, given the generality of the method. Many existing methods used to obtain sampling bounds rely on a) (model) distributions, or its forms, on the parameters and/or values (observed and/or latent) in the data, b) the structure of the constraints, c) the form of the optimization function. In our case, we consider a very generic sampling method and therefore determining an exact analytical form of the dependency between α and β is beyond the scope of this paper.

It is possible, however, to empirically estimate the parameter β for a particular choice of α. We use the following approach: we estimate the average β for a particular value of α as the mean of n iterations. In the following the upper index denotes the iteration: $\hat{\beta} = \frac{\sum_i \beta^i}{n}$. Then, to estimate β^i for an arbitrary iteration i, we assume that we have computed an optimal solution G^i greedily and an approximation S^i via sampling as $\hat{\beta^i} = \frac{S^i}{G^i}$.

An empirical analysis using this approach can be found in Appendix G.

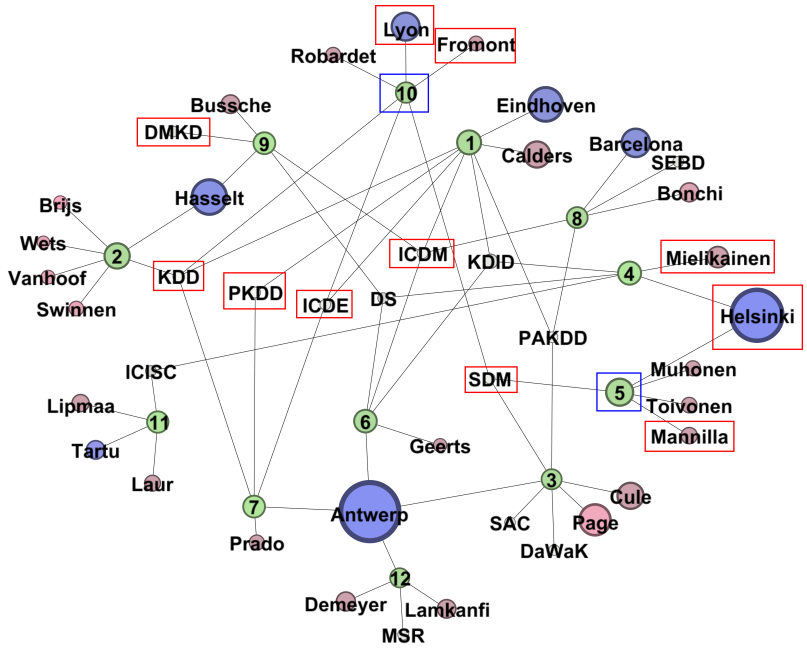


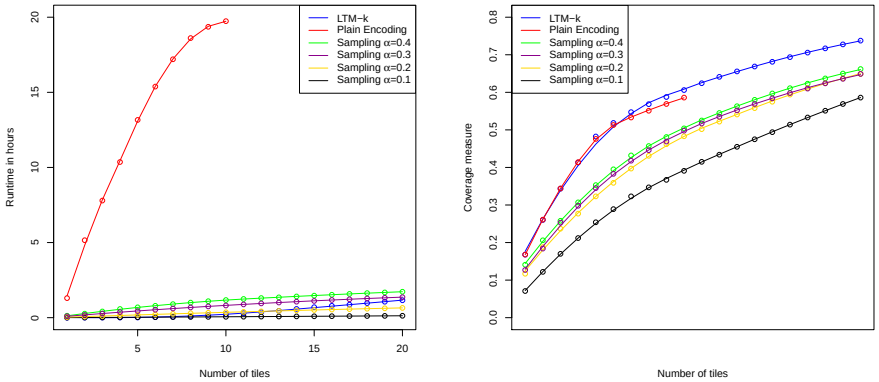
Fig. 14: An application of purely relational factorization: intersection of publication data (in red) and of latent topics (in blue) of Bart Goethals with publications of Pauli Miettinen (not his co-author). A latent topic belongs to an intersection if both have the same co-author, university and venue in the publication records.

G Additional Tiling Experiments with Sampling

Here we present the estimation of β given corresponding values of α, as visualized in Figures 15a and 15b. The calculation method is described in Appendix F. We computed the values for β in Table 6 based on the data from Figure 15b.

Table 6: An empiricial reconstruction of β, given the values of α

α	0.1	0.2	0.3	0.4
β	0.55	0.73	0.76	0.79



(a) Runtime **(b)** Coverage

Fig. 15: Tiling comparison (runtime, coverage) with LTM-k (Mushroom dataset)