Int J Data Sci Anal (2016) 1:61–76
DOI 10.1007/s41060-016-0004-3

REGULAR PAPER

P-N-RMiner: a generic framework for mining interesting structured relational patterns

Jefrey Lijffijt[1,2] · **Eirini Spyropoulou**[2] · **Bo Kang**[1,2] · **Tijl De Bie**[1,2]

Received: 7 December 2015 / Accepted: 16 January 2016 / Published online: 3 February 2016

Abstract Methods for local pattern mining are fragmented along two dimensions: the *pattern syntax*, and the *data types* on which they are applicable. Pattern syntaxes include subgroups, n-sets, itemsets, and many more; common data types include binary, categorical, and real-valued. Recent research on relational pattern mining has shown how the aforementioned pattern syntaxes can be unified in a single framework. However, a unified model to deal with various data types is lacking, certainly for more complexly structured types such as real numbers, time of day—which is circular—, geographical location, terms from a taxonomy, etc. We introduce P-N-RMiner, a generic tool for mining interesting local patterns in (relational) data with structured attributes. We show how to handle the attribute structures in a generic manner, by modelling them as partial orders. We also derive an information-theoretic subjective interestingness measure for such patterns and present an algorithm to efficiently enumerate the patterns. We find that (1) P-N-RMiner finds patterns that are substantially more informative, (2) the new interestingness measure cannot be approximated using existing methods, and (3) we can leverage the partial orders to speed up enumeration.

Keywords Data mining · Pattern mining · Information theory · Subjective interestingness · Relational data · Structured attributes

✉ Jefrey Lijffijt
jefrey.lijffijt@ugent.be

[1] Data Science Lab, Ghent University, Ghent, Belgium

[2] Intelligent Systems Lab, University of Bristol, Bristol, UK

1 Introduction

Exploratory data mining (EDM) tools enable businesses and scientists to explore their data and find previously unknown patterns, which in turn helps them learn about reality, innovate, and gain a competitive edge. An important obstacle for the adoption of EDM techniques in general, and local pattern mining approaches in particular, is their limited flexibility in terms of the data types to which they can be applied, e.g., only tabular data, and the types of patterns they can generate, e.g., itemsets. In reality, however, data are often complexly structured (e.g., relational databases), and additionally there is often structure among the different values data attributes may attain, i.e., attribute values can be ordinal, interval, taxonomy terms, and more.

Local pattern mining has traditionally been rooted in categorical or even binary data, including algorithms for frequent itemset mining and variants [2], n-set mining [7], subgroup discovery [14], and multi-relational pattern mining [15,24]. Some of these local pattern mining approaches have been extended in various ways to include ordinal, real-valued, or other data structures. For example, extensions of itemset mining to real-valued data have led to approaches akin to biclustering, and subgroup discovery methods exist that allow discovery of rules based on attribute-value inequalities.

However, that work is fragmented and often ad hoc, in the sense that other kinds of structure (taxonomy terms, time-of-day intervals on a circular 24-hour clock, geographical regions on the globe, etc.) may not be approachable in the same way and may necessitate fundamentally different approaches. The purpose of this paper is to provide an elegant and encompassing framework to deal with attributes of any of the structured types listed above and more, and this in a relational setting, i.e., applicable to data as it resides in relational databases. To illustrate the breadth and nature of the contributions, we provide two motivating examples.

Example 1 Consider a dataset of Foursquare[1] check-in times of a number of users. Such a dataset has the potential of elucidating lifestyle patterns shared by a number of Foursquare users. To formalise and then find such patterns, it is tempting to specify a time resolution and discretise the data. However, it is unclear which discretisation level to use, and whether to take it uniform throughout the day. In fact, the optimal discretisation could vary for different lifestyle patterns.

An alternative approach could be to take the mean and possibly higher-order statistics of the check-in times for each user and find patterns in this summary description. This approach would suffer from two problems: first, computing averages of circular quantities is ambiguous (e.g. is the mean of 6 am and 6 pm midnight or noon?), and second, it ignores much of the information in the data.

The method developed in this paper, when applied to this data, deems as most interesting a pattern that reveals that 1.6% of all users check in frequently in the 6am-7am interval and again in the 10.10–10.50 am interval. Here, the interval sizes are tuned automatically to maximise interestingness, and the intervals can be of varying size even within a pattern.

While this example illustrates how the contribution in this paper advances the state-of-the-art even for a single relation (between users and check-in times), the second example shows the full power on data in a relational database.

Example 2 Consider a relational database of users, who have rated books with an integer from 1 to 5, and where the books are tagged with a number of genres organised in a taxonomy. Applied to this dataset, the method proposed in this paper identifies interesting patterns in the form of sets of books that have been rated by the same set of users in a similar way (say, in the interval from 3 up to 5), which may all belong to a particular set of genres (e.g., fantasy and action).

This second example illustrates the ability of the proposed method to identify patterns that span several types of entities (users, ratings, books, genres), including structured entity types such as ordinal values or values organised in a taxonomy.

The work in this paper is most easily explained as an extension of the N-RMiner algorithm for mining local patterns in relational databases [23], towards structured entity types. However, given the generality of the N-RMiner pattern syntax, this immediately results in a method that includes itemset mining, n-set mining, and subgroup discovery for structured data types as special cases. To do this, we overcome the following challenges.

[1] https://foursquare.com/.

- We formalise the problem and a matching pattern syntax, in a manner as generic as possible (Sect. 2). To achieve this, we adopt an abstract formalisation in terms of a partial order over the structured values. For example, with the time-of-day and book ratings, the partial order is over the intervals, where one is ‘smaller’ than another if it is included in it. For taxonomy terms, one taxonomy term is smaller than another if it is a specialisation of it.
- We formalise the interestingness of such patterns under the Information Theoretic framework for subjective interestingness [9,11]. This is a non-trivial contribution over the approach applicable for the N-RMiner pattern syntax, because the presence of entities is no longer independent (Sect. 3).
- We provide an algorithm for efficiently enumerating all such patterns (Sect. 4). This is a non-trivial extension of the algorithmic approach used in N-RMiner that is applicable due to the additional structure in the search space. However, we also prove that under the algorithmic framework used here (due to [1]), no algorithm can exist that uses only a polynomial number of steps per output. This result is new but also applies to earlier works.

2 Problem formalisation

2.1 Notation

We formalise a relational database as follows. Let E denote the set of *entities*, that is all possible values of all attributes, and $t : E \to \{1, \ldots, k\}$ a function that gives the *type of an entity* (assuming k types). We write $\mathcal{R}$ to denote the set of all *relationship instances* in the database, while $R \subseteq \{1, \ldots, k\} \times \{1, \ldots, k\}$ denotes the set of tuples of entity types whose entities may have relationships, according to the schema of the database. The elements of R will be referred to as the *relationship types*. A *relational database* is then a tuple $\mathcal{D} = (E, t, \mathcal{R}, R, \succeq)$, where $\succeq$ will be introduced below.

As an example, consider the schema illustrated in Figure 1. There are four entity types: *User* (1), *Check-in times* (2), *Profession* (3), and *Age* (4). The numbering is arbitrary. The

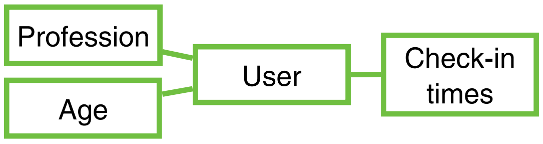

Fig. 1 Example schema of users and check-in times. Additionally, we know the age and profession of the users. There are three relationship types: there are relationships between (1) users and check-in times, (2) users and ages, and (3) users and professions

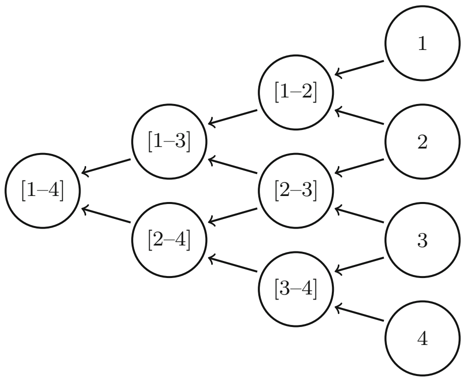

Fig. 2 Partial order of all intervals that are supersets of {1}, {2}, {3}, and {4}. The partial order corresponds to the superset relation

set of entities E contains all possible values for all types, and t is a function that returns the type of that entity. The set of relationships is $R = \{(1, 2), (1, 3), (1, 4)\}$ and finally $\mathcal{R}$ contains all the instances of such relationships.

In this example, *Age*, *Check-in times*, and *Profession* could all be structured attributes; the values of *Age* are numerical, *Check-in times* are numerical but without full order, and *Profession* has hierarchical structure. One could be interested in finding patterns in such data not only including an exact age such as 32, but also intervals such as [25–35]. The set of all such intervals can be modelled as a partial order. An example of such a partial order is given in Fig. 2.

To model such structure, we consider one additional element in the data model: a partial order $\succeq$ that represents implication of relationships across entities of the same type. That is, $e \succeq f$ means that if any entity g is related to f, i.e., $(f, g) \in \mathcal{R}$, then g is also related to e: $\forall e, f, g \in E : e \succeq f \wedge (g, f) \in \mathcal{R} \Rightarrow (g, e) \in \mathcal{R}$. Only implications between entities of the same type are allowed: $e \succeq f \Rightarrow t(e) = t(f)$.

For example, in Fig. 2, we have $[1–2] \succeq 1$, $[1–3] \succeq [1–2]$, etc. This means that if an entity is connected to 1 it is also connected to [1–2], [1–3], and [1–4]. For notational convenience, we assume that $\mathcal{R}$ contains both the relationship instances present in the database, as well as all relationship instances implied by $\succeq$. In practice, we need not store these implied edges explicitly; details on this are presented in Sect. 5.

2.2 Pattern syntax

Our general aim is to find *interesting* sets of entities. We propose that the interestingness of a set of entities can be measured by contrasting the number of relationship instances present between the entities with the expected number of relationship instances present between those entities, where the expectation is *subjective*, i.e., dependent on the user. We formalise this subjective interestingness in Sect. 3. For now it suffices to know that it will depend on the number of relationship instances between the entities in the set.

We use a tiered approach to achieve our general aim. First, we enumerate all dense patterns that are potentially interesting, and secondly we rank them by interestingness. Hence, the first step is to find sets of entities that have many relationships. We will refer to a set of entities and the relationship instances among them as a *pattern*. We define a pattern as potentially interesting if it is *complete*, *connected*, *maximal*, and *proper*.

Definition 1 An entity set $F \subseteq E$ is **complete** iff

$$\forall (t_1, t_2) \in R, \forall e_i, e_j \in F, t(e_i) = t_1, t(e_j) = t_2 : (e_i, e_j) \in \mathcal{R}.$$

More verbosely, a pattern F is *complete* iff all relationship instances between entities in F that are allowed by the database schema are also present.

Definition 2 An entity set $F \subseteq E$ is **connected** iff

$$\forall e, f \in F, e \neq f : (e, f) \in \mathcal{R} \vee \exists g \in F, \{e, g\} \text{ connected} \wedge \{f, g\} \text{ connected}.$$

A set of entities F s.t. $|F| \geq 2$ is *connected* iff there is a path between any two entities in F using only entities in F. Any F s.t. $|F| \leq 1$ is *connected*.

Definition 3 An entity set $F \subseteq E$ is **proper** iff

$$\forall e \in F, f \in E, f \succeq e : f \in F.$$

A pattern F is *proper* iff all super-entities of any entity in F are also in F.

Definition 4 An entity set $F \subseteq E$ is **maximal** iff

$$\nexists e \in E \setminus F : F \cup \{e\} \text{ is } \textbf{complete} \text{ and } \textbf{connected}.$$

Finally, a pattern F is *maximal* iff no entity can be added without breaking completeness or connectedness. Note that if there is an entity $e \in E \setminus F$ such that $F \cup \{e\}$ is complete and connected, there must also be an entity $f \succeq e$, $f \in E \setminus F$ such that $F \cup \{f\}$ is complete, connected, and proper. We refer to sets that are complete, connected, and proper as *complete connected proper subsets (CCPSs)*, and to sets that are also maximal as *maximal CCPSs*. In Sect. 4, we will show that we can enumerate all maximal CCPSs using the so-called fixpoint-enumeration algorithm.

In short, we add a properness constraint and the pattern syntax is otherwise equivalent to [23,24]. Our implementation and theory also support n-ary relationships, but we do not discuss this further in order to prevent unnecessary complications in the exposition. One could also consider approximate patterns by discarding the completeness constraint. This would lead to an increased computational complexity, but the increase has been shown to be manageable [22]. For simplicity, we do not consider approximate patterns in this paper.

3 Interestingness

3.1 General approach

Although we limit the output to maximal CCPSs, the number of patterns can be—and often is—exponential in the number of entities in the database. Therefore, it is vital to have a mechanism for identifying the most interesting CCPSs. To achieve this, we build upon the framework for subjective interestingness in exploratory data mining (FORSIED), introduced by De Bie [9,11]. This framework is based on modelling a user's prior belief state about the data by means of the Maximum Entropy distribution subject to any stated prior beliefs the user may hold about the data. This distribution is referred to as the *background distribution*. The interestingness of a pattern is then formalised by contrasting the pattern with this background distribution, as the ratio of two quantities:

- the *self-information* of the pattern, defined as minus the logarithm of the probability that the pattern is present under the background distribution, and
- the *description length* of the pattern, which should formalise the amount of effort the user needs to expend to assimilate the pattern.

Both are explained in more detail below. In full, the interestingness of a pattern F is its *information ratio*:

$$\text{InformationRatio}(F) = \frac{\text{Self-Information}(F)}{\text{DescriptionLength}(F)}.$$

Given the dependence of this measure on the background distribution, which may in principle differ for different users, this interestingness measure is a *subjective* quantity.

In [24], this framework is used successfully to formalise the interestingness of Complete Connected Subsets (CCSs), without the properness requirement that lies at the core of the contributions in this paper. The properness requirement creates an opportunity as well as a non-trivial challenge. It allows to describe single patterns that capture information that could previously be presented only with a set of patterns. Such patterns reduce the description length. On the other hand, it is more difficult to compute the self-information of a pattern. We briefly discuss the core principles in the next paragraphs, before discussing the computation of the self-information in greater detail in Sect. 3.5.

3.2 Description length

The description length of a CCS pattern is formalised as an affine function of the number of entities $|F|$ in F. Let $|E|$ be the total number of entities in the database and $p \in (0, 1)$ a parameter that trades off the cost between describing the presence of an entity in the pattern F, cost $log(p)$, and describing its absence, cost $log(1 - p)$. Then, the description length is defined as [24]:

$$\text{DescriptionLength}(F) = |F| \log\left(\frac{1-p}{p}\right) + |E| \log\left(\frac{1}{1-p}\right).$$

Note that, to convey a CCPS pattern to the user, only the *minima* of F need to be described. Indeed, the presence of the entities that are larger is implied; explicitly describing these would be redundant. Thus, the above expression needs to be modified by replacing $|F|$ with the number of *minima* in F. This leads to a smaller description length than would be required if the partial order $\succeq$ would be unknown or unaccounted for.

3.3 Information content

The central idea of FORSIED is to quantify the amount of information that a pattern conveys to a user, which in general terms is known as the *information content* of a pattern. The most interesting pattern is then the one that conveys the most information, i.e., that maximally reduces the uncertainty the user has about the data [9]. The self-information of a pattern quantifies the unexpectedness of that pattern, given a background distribution. We present the technical details of the self-information and the background distribution below.

In the following section, we argue that the background distribution can be fitted in the exact same way as in [24]. However, how to compute the probability that a given pattern is present—and thus its self-information—is not trivial. The difficulty stems from the fact that the presence of relationship instances is now dependent, owing to the partial order relation over the entities. Nonetheless, Sect. 3.5 describes how the probabilities can still be computed effectively by using the inclusion–exclusion principle.

3.4 The Background Distribution

In [24], interestingness is formalised under the assumption that users have prior beliefs on the number of entities of a specific type to which a given entity is related. It is argued that this is often a good assumption, and the experiments in the current paper also support that.[2] This assumption leads to a tractable distribution, under which the relationship instances are independent with probabilities that can be found by solving an efficiently solvable convex optimisation problem.

This background distribution factorises over the different relationship types, such that the self-information can be decomposed into a sum of different contributions, each one of which corresponds to the relationship instances for one particular relationship type. That is also the case in the present paper, such that in the rest of this exposition it suffices to imagine just a single relationship type.

What is new is that we implicitly make a further assumption on the user's knowledge state, namely that the user knows the partial order $\succeq$, and hence the fact that if $e \succeq f \wedge (g, f) \in \mathcal{R}$, then $(g, e) \in \mathcal{R}$. This creates hard-to-handle dependencies between the presence of relationship instances. In practice, data will often only contain relationship instances between *minimal entities*, i.e., entities that are minimal in the partial order $\succeq$. In this case, the background distribution can be fitted on the set of minimal entities without worrying about the dependencies, exactly as done in [24].

In particular, we assume prior beliefs on the number of relationship instances each (minimal) entity is involved in, for every relationship type. The maximum entropy distribution subject to these prior belief constraints is then used as the background distribution. This background distribution is a product of Bernoulli distributions with one factor for each possible relationship instance [24]. In other words: for each possible relationship instance (e, f), the distribution gives us a probability $p_{(e,f)}$ that (e, f) is present in the data.

This background distribution defines the probabilities $p_{(e,f)}$ of relationship instances between minimal entities e and f. Given this, it is possible to compute the probability $p_{(e,f)}$ of any relationship instance (e, f), whether minimal or not, as the probability of presence of any of the relationship instances (e', f') with $e \succeq e'$ and $f \succeq f'$ and e' and f' minimal. Indeed, the presence of any such (e', f') would imply the presence of (e, f). How this probability and the overall probability of a CCPS pattern can be computed given the background distribution is the subject of Sect. 3.5.

In general, for data that includes relationship instances between non-minimal entities, let us define a partial order $\succeq_{\mathcal{R}}$ over the relationship instances as follows: $(e_1, f_1) \succeq_{\mathcal{R}} (e_2, f_2)$ iff $e_1 \succeq f_1$ and $e_2 \succeq f_2$. Then, we suggest fitting the background distribution as before on the minimal relationship instances only. This includes the approach from the previous paragraph as a special case. This model is imperfect, as the user should be aware of *negative* dependencies between the presence of a relationship instance as a minimal one: if (e_2, f_2) is a minimal relationship instance, then (e_1, f_1) with $(e_1, f_1) \succeq_{\mathcal{R}} (e_2, f_2)$ and (e_3, f_3) with $(e_3, f_3) \preceq_{\mathcal{R}} (e_2, f_2)$ cannot be minimal relationship instances. Yet, we argue that in this case, assuming independence is nonetheless still a good approximation.[3]

3.5 Self-information

Given a pair of entities (e, f) such that $(t(e), t(f)) \in R$—i.e., (e, f) may be related according to the database schema—let us denote the event that $(e, f) \notin \mathcal{R}$ as $A_{(e,f)}$ (A for Absent). The probability of this event under the background distribution can be computed as[4]

$$P(A_{(e,f)}) = \prod_{(e',f'):(e,f)\succeq_{\mathcal{R}}(e',f')} (1 - p_{(e',f')}).$$

The presence of a CCPS pattern F corresponds to the event defined by the complement of the union of all events $A_{(e,f)}$ with $e, f \in F$ and $(t(e), t(f)) \in R$. Hence, the union of all these events corresponds to the event where at least one of the relationship instances is missing. The complement of the union of absence events implies the presence of the pattern. Defining $\mathcal{T}_F$ as $\mathcal{T}_F = \{(e, f) | e, f \in F, (t(e), t(f)) \in R\}$ (the set of pairs of entities in F), this means that the probability of a pattern is given as $1 - P\left(\cup_{(e,f)\in\mathcal{T}_F} A_{(e,f)}\right)$. Note that it suffices to consider only the minimal relationship instances from $\mathcal{T}_F$, because $\neg A_{(e,f)}$ implies $\neg A_{(e',f')}$ for any $e' \succeq e$, $f' \succeq f$.

Directly computing this probability is nontrivial, given the dependencies between the events $A_{(e,f)}$. Fortunately, we can use the inclusion–exclusion principle to compute it as follows:

$$P\left(\bigcup_{(e,f)\in\mathcal{T}_F} A_{(e,f)}\right) = \sum_{I\subseteq\mathcal{T}_F} (-1)^{|I|-1} P\left(\bigcap_{(e,f)\in I} A_{(e,f)}\right).$$

[2] Of course, exploring other types of prior beliefs is an important line of further work.

[3] The intuition is as follows. In practice the probabilities for relationship instances under the background distribution are small. Additionally, for two events with small probabilities p and q, the probability of their union is between $p + q$ (in the case of perfect negative dependence) and $1 - (1 - p)(1 - q) = p + q - pq$ (in the case of independence), which differs by only pq, such that assuming independence results in at most a second-order error in the probabilities.

[4] As pointed out in Sect. 3.4, this expression is exact for databases where relationship instances involve only minimal pairs, and a good approximation in practice in other cases. Note also that only minimal relationship instances have positive probability, and hence non-minimal instances can be ignored.

Now, the probability of the intersection of events $A_{(e,f)}$ can be computed straightforwardly as:[5]

$$P\left(\bigcap_{(e,f)\in I} A_{(e,f)}\right) = \prod_{(e',f'):(e,f)\succeq_{\mathcal{R}}(e',f')} 1 - p_{(e',f')}.$$

Hence, we can compute the probability of the presence of a pattern. The self-information is then given as the negative logarithm of this probability:

$$\text{Self-Information}(F) = -\log\left(1 - P\left(\bigcup_{(e,f)\in\mathcal{I}_F} A_{(e,f)}\right)\right).$$

4 Enumeration algorithm

Last but not least, we study how to efficiently enumerate all maximal CCPSs. Like previous work on mining interesting patterns in relational data [22–24], our algorithm is based on the *fixpoint-enumeration algorithm* by Boley et al. [1]. Although that algorithm already exists, it should be noted that it is a meta-algorithm, which does not directly work on the data. The fixpoint-enumeration algorithm takes as input a *set system* and a *closure operator* that together define the problem setting and the output. The definitions are given below.

We first introduce the fixpoint-enumeration algorithm, after which we introduce notation and formalise our practical problem of enumerating maximal CCPSs as a problem of enumerating all fixpoints in a set system. We prove that the introduced set system is *strongly accessible*, which is required for the fixpoint enumeration to be applicable, and present a suitable closure operator. Finally, we analyse the computational complexity, and we prove that—unfortunately—the delay time between two maximal CCPSs cannot be polynomial under this scheme.

4.1 The enumeration algorithm

The fixpoint-enumeration algorithm can efficiently enumerate all fixpoints in a strongly accessible set system $(E, \mathcal{F})$, where E is a set of objects called the *ground set* and $\mathcal{F} \subseteq \mathcal{P}(E)$ a family of sets. The *fixpoints* are defined by a closure operator σ. The output of the algorithm is valid if and only if the set system satisfies certain criteria [1]. The algorithm is very simple:

(1) Start with an empty set: $F := \{\emptyset\}$.
(2) Compute the closure of the current set: $F := \sigma(F)$. This closure is one of the fixpoints to return.
(3) If the current set can be extended, that is, $\exists G \supseteq F : G \in \mathcal{F}$, then pick any element $f \in G \setminus F : F \cup \{f\} \in \mathcal{F}$ and recurse from (2) to one branch where every set contains f and one branch where no set contains f. If the current set cannot be extended, then this branch ends.

If and only if the set system $(E, \mathcal{F})$ is strongly accessible, then all sets in $\mathcal{F}$ can be found by adding elements one by one while traversing only over sets in $\mathcal{F}$ [1]. The closure operator defines the fixpoints, which should be interpreted as the subset of sets from $\mathcal{F}$ that we would like to enumerate[6].

4.2 Enumerating CCPSs

The set of all CCPSs forms a set system $(E, \mathcal{F})$ where the ground set E is the set of entities and $\mathcal{F}$ is the set of CCPSs, defined as

$$\mathcal{F} = \{F \in \mathcal{P}(E) : \\ F \text{ connected} \wedge F \text{ complete} \wedge F \text{ proper}\}.$$

The fixpoint-enumeration algorithm can be used to enumerate all closed patterns from this set system, and it is efficient if we can define an appropriate closure operator. Ultimately, we are interested in enumerating the maximal CCPSs, while $\mathcal{F}$ contains all CCPSs.

4.3 Strong accessibility

For the fixpoint-enumeration algorithm to be applicable, the set system must be *strongly accessible*. This is the case iff

$$\forall F \in \mathcal{F} \setminus \{\emptyset\} : \exists e \in F : F \setminus \{e\} \in \mathcal{F}, \text{ and} \tag{1}$$

$$\forall F, F' \in \mathcal{F}, F \subset F' : \exists e \in F' \setminus F : F \cup \{e\} \in \mathcal{F}. \tag{2}$$

Theorem 1 *$(E, \mathcal{F})$ is strongly accessible.*

Proof We prove each of the two properties separately, but first we introduce some notation for convenience. Let $(F, \succeq)$ denote the set F partially ordered by $\succeq$. We write that an entity $e \in F$ is minimal in $(F, \succeq)$ iff $\nexists f \in F, e \neq f, e \succeq f$. Likewise an entity $e \in F$ is maximal in $(F, \succeq)$ iff $\nexists f \in F, e \neq f, f \succeq e$.

The first property states that for every CCPS F, there should be an entity $e \in F$ that can be removed such that

[5] Note again that only minimal relationship instances (e', f') need to be considered, since non-minimal relationship instances have zero probability.

[6] This is actually not the case here, because we are interested only in maximal sets, rather than all fixpoints (closed sets). This is explained further in Sect. 4.5.

we obtain another CCPS $F' = F \setminus \{e\}$. We prove this by narrowing down candidates by looking in turn at completeness, properness, and finally connectedness:

(1) $\forall F \in \mathcal{F} \setminus \{\emptyset\} : \exists e \in F : F \setminus \{e\} \in \mathcal{F}$, because

- Removing an entity never violates completeness.
- Any entity $e \in F$, e minimal in $(F, \succeq)$ can be removed without breaking properness and $\exists e \in F$, e minimal in $(F, \succeq)$.
- If $\exists e, f \in F, e \succeq f, f$ minimal in $(F, \succeq)$, then $F \setminus \{f\} \in \mathcal{F}$, because $F \setminus \{f\}$ is complete and proper (see two previous statements) and since F is connected, for any $(f, g) \in \mathcal{R}$ also $(e, g) \in \mathcal{R}$ (since $e \succeq f$), thus $F \setminus \{f\}$ is also connected.
- If $\nexists e, f \in F, e \succeq f, f$ minimal in $(F, \succeq)$, then $\forall e \in F : e$ minimal in $(F, \succeq)$. Hence, removal of any entity would not break completeness or properness. Then, we could model the entities of F as nodes in a graph and the relationship instances between entities in F as its edges. Since F is connected, that graph is also connected. Any connected graph has a spanning tree and it is possible to remove any leaf node from that spanning tree without breaking connectedness of the graph.

The second property states that for any pair of CCPS $F, F' \in \mathcal{F}, F \subset F'$, there is an entity $e \in F' \setminus F$ that can be added to F to lead to another CCPS $F \cup \{e\} \in \mathcal{F}$. We prove this property by considering all entity types of entities that are in F' and not in F, and then we condition on whether F is the empty set or whether it already contains some entities.

(2) $\forall F, F' \in \mathcal{F}, F \subset F' : \exists e \in F' \setminus F : F \cup \{e\} \in \mathcal{F}$, because

- Let $t(F) = \{t_j | t_j = t(e), e \in F\}$. For every type $t_j \in t(F' \setminus F)$, $\exists e \in F' \setminus F : t(e) = t_j$, e maximal in $(F' \setminus F, \succeq)$, since $F' \setminus F$ is finite.
- If $F = \emptyset$, then for $\forall e \in F' \setminus F, t(e) = t_j, e$ maximal in $(F' \setminus F, \succeq) : F \cup \{e\} \in \mathcal{F}$.
- If $F \supset \emptyset$, then because every entity type has one or more maximal elements and F' is connected, there is a type adjacent to or present in F which includes an entity e maximal in $(F' \setminus F, \succeq)$ and then $F \cup \{e\}$ is complete, connected and proper. □

4.4 The closure operator

Strong accessibility implies that we can efficiently enumerate all fixpoints in $\mathcal{F}$ in a single traversal over the set system without considering any set twice [1]. A trivial choice for the fixpoints would be all sets in F; in which case $\sigma(F) = F, \forall F \in \mathcal{F}$. However, in the worst case the number of CCPSs $|\mathcal{F}|$ is an exponential in $|E|$, while there is only one maximal CCPS. Hence, we would like to choose the set of fixpoints such that it includes all maximal CCPSs and as few other CCPSs as possible. It is not possible to choose the closure operator such that we enumerate only maximal CCPSs, because a CCPS may have multiple maximal extensions.

We derive a suitable closure operator from its requirements; an operator $\sigma : \mathcal{F} \rightarrow \mathcal{F}$ is a closure operator iff $\forall F, G \in \mathcal{F}, \sigma$ is

$$\begin{aligned}
&\textit{extensive} : F \subseteq \sigma(F),\\
&\textit{idempotent} : \sigma(\sigma(F)) = \sigma(F), \text{ and}\\
&\textit{monotonic} : F \subseteq G \Rightarrow \sigma(F) \subseteq \sigma(G).
\end{aligned}$$

Firstly, extensivity is straightforward to guarantee, we choose $\sigma(F)$ such that it never removes entities from F. Secondly, due to idempotency, we require that the closure of a maximal CCPS is the maximal CCPS itself; otherwise, it is not a fixpoint and will not be in the output. Thirdly, suppose that the set F has two supersets F', F'' that are maximal CCPSs: $F', F'' \supseteq F, F' \neq F''$. Since they are maximal, they both contain an entity that is not in F, nor in the other maximal CCPS. Extensivity combined with monotonicity forces us to choose $\sigma(F)$ such that it does not add any entities that are missing from any superset $G \supseteq F, G \in \mathcal{F}$.

Hence, we define the closure as follows. Let the set of *compatible entities* be $\text{Comp}(F) = \{e \in E \mid F \cup \{e\} \text{ is complete}\}$, i.e., all entities that can still be added to F, and let the set of *augmentation entities* be $\text{Aug}(F) = \{a \in A \mid F \cup \{a\} \in \mathcal{F}\}$, i.e., all entities that can be added while leading to a valid CCPS. Then we define the closure operator as in [24]:

$$\sigma(F) = \{e \in \text{Aug}(F) \mid \text{Comp}(F \cup e) = \text{Comp}(F)\}.$$

This operator is extensive and monotonic, but not idempotent. Without idempotency, the algorithm would still enumerate all maximal CCPSs, but also unnecessary non-maximal CCPSs. We achieve idempotency by repeating the closure operator until $\sigma(F) = F$. This repetition can be done efficiently by considering only entities that have just become part of $\text{Aug}(F)$.

In [24], it is assumed that the dataset does not contain any entity e that is related to all entities of a neighbouring type, because if such an entity exists, all other entities could be in its set of compatible entities ($Comp(\{e\}) = E$), hence $\sigma(\emptyset) \supseteq \{e\}$, while e need not be part of every CCS. Thus, this assumption is required for the closure operator to be monotonic.

In the current setting, entities that are fully connected to a neighbouring type would not be uncommon and this assumption is not reasonable. For example, there could be a catch-all entity in a hierarchical attribute. Hence, we additionally define $\sigma(\emptyset) = \emptyset$. Alternatively, one could redefine

Comp as $\text{Comp}(F) = \{e \in E | \exists G \supseteq F \cup \{e\}, G \in \mathcal{F}\}$, but we leave that to future work. For brevity, we omit the proof that this σ is a closure operator.

4.5 Final remarks

The fixpoint-enumeration algorithm enumerates all fixpoints, i.e., any set that results from computing the closure operator. We are only interested in maximal CCPSs, so we output only those. Maximal CCPSs are easily identified at runtime, as they are fixpoints where no entity could be added (Sect. 2, Definition 4).

Finally, we allow a user to put any number of constraints on the set of patterns in the form *"any pattern should include at least X entities of type Y"*. We implement this by continuously computing upper bounds during the mining process, such that we can prune any branch where the constraints cannot be satisfied any more. A similar approach is followed in [24].

4.6 Computational complexity

As stated previously, the number of maximal CCPSs can be exponential in $|E|$. Since P-N-RMiner exhaustively enumerates all maximal CCPSs, the worst-case complexity of P-N-RMiner is also exponential in $|E|$. Unfortunately, we are not aware of an upper bound on the number of maximal CCPSs, nor do we know the exact worst-case complexity of our algorithm.

It has been shown that the delay time between finding two *closed* CCSs using the fixpoint-enumeration algorithm is $O(|E|^3)$ [24]. The algorithm used here is almost the same, except that computing the set of augmentation entities also involves checking the properness constraint. The complexity of that is $O(|E|)$, hence the delay time for closed CCPSs is also $O(|E|^3)$.

It was previously not known whether the delay time between the enumeration of two maximal CC(P)Ss is always polynomial. Although we cannot make a general statement about the delay time, we prove here that the fixpoint-enumeration algorithm can indeed require a number of steps exponential in the number of outputs. We prove this by means of an example data set where the number of closed CCSs is exponential in the number of maximal CCSs, while indeed the number of closed CCSs is already exponential in the size of the input.

Theorem 2 *No algorithm that is an instantiation of fixpoint enumeration can guarantee a polynomial number of steps in the number of outputs (maximal CCSs).*

Proof Consider a database with entity types A and B and a single binary relation between the two types; $R = \{A, B\}$. Let both entity types have n entities, numbered $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$. Let the set of relationship instances contain all pairs (a_i, b_j), $i, j \in [1, n]$, $i \neq j$. That is, all possible relationship instances exist, except for entities a_i and b_i with the same index.

The number of *maximal* CCSs follows fairly straightforwardly: all CCSs can be extended until they have n entities and for each index i we can include either a_i or b_i. Including both would violate completeness, while the CCS is not maximal as long as for some index i neither is included. This would lead to 2^n maximal CCSs, except that neither the choice to include all entities in A, nor all entities in B are valid choices; this violates connectedness. Hence, there are $2^n - 2$ maximal CCSs.

The number of *closed* CCSs is only slightly more involved: notice that $2^n - 2 = \sum_{i=1}^{n-1} \binom{n}{i}$, which highlights that the number of maximal CCSs is indeed the number of choices to pick $1, \ldots, n-1$ entities from A, which then form a unique maximal CCS if augmented with the remaining items from B. The second observation that we can use to derive the number of closed CCSs is that for this data every CCS is closed, because any entity that we add (a_i or b_i) will reduce the set of compatible entities by one. Hence, the closure of every CCS is that CCS itself.

Let $|\mathcal{F}|$ denote the number of CCSs. The set of CCSs is found as the selection of $i \in \{1, \ldots, n\}$ entities from A, completed with $j \in \{1, \ldots, n-i\}$ entities from B, plus all $2n$ singletons. Hence, we find

$$
\begin{aligned}
|\mathcal{F}| &= 2n + \sum_{i=1}^{n-1} \binom{n}{i} \cdot \left(\sum_{j=1}^{n-i} \binom{n-i}{j} \right) \\
&= 2n + \sum_{i=1}^{n-1} \binom{n}{i} \cdot \left(2^{n-i} - 1\right) \\
&= 2n + \sum_{i=1}^{n-1} \binom{n}{i} \cdot 2^{n-i} - \sum_{i=1}^{n-1} \binom{n}{i} \\
&= 2n + \left(3^n - 2^n - 1\right) - \left(2^n - 2\right) \\
&= 3^n - 2^{n+1} + 2n + 1.
\end{aligned}
$$

Since the fixpoint-enumeration algorithm enumerates all closed CCSs, the number of closed CCSs is $O(3^n)$, while the number of maximal CCSs is $O(2^n)$. It follows that the number of steps required by the algorithm per maximal CCS is $O(3^n/2^n)$. The following is speculation: it may be possible that all maximal CCSs are enumerated in only $O(2^n)$ steps, which is why a conclusion regarding the delay time between two maximal CCSs is more difficult to obtain. However, the algorithm cannot know it has found all maximal CCSs until it has processed all $O(3^n)$ closed CCSs; hence, the general complexity per maximal CCS is $O(3^n/2^n)$.

Finally, notice that we concluded previously (Sect. 4.4) that regardless of the definition of the closure operator, the closure operator cannot add any entities to a set F unless they are part of every maximal CCPS that is a superset of F. This implies that our closure operator defined here is indeed optimal for any database involving only one relationship type. Hence, this proof holds for any instantiation of fixpoint enumeration. □

Notice that this proof is for CCSs, and since properness need not be present in the data, the proof is also valid for CCPSs, as well as all other RMiner variants.

5 Implementation

We implemented the full program in C++ and the implementation turned out to be surprisingly difficult. The main difficulty is the efficiency of the enumeration algorithm. To facilitate understanding and reproduction of the tool, we provide full pseudocode here (Algorithms 1–4). The full source code is available at https://bitbucket.org/BristolDataScience/p-n-rminer. Our implementation is based on N-RMiner [23] and the pseudocode is partly based on the description in [21].

The main function is `P-N-RMiner` (Algorithm 1), which takes as arguments four sets of entities and a list of entity types. Entity set F contains the entities whose supersets need to be enumerated in the current branch, this set is constructed via branching and the closure. Entity set B contains the entities all whose supersets already have been enumerated; this set is used for pruning. Entity sets C and A and the entity types list $types$ are passed on for efficiency; these are the compatible entities, augmentation entities, and types adjacent to F. The initial call is `P-N-RMiner`$(\emptyset, \emptyset, E, E, \emptyset)$.

The main structure of `P-N-RMiner` is the *for* loop over all augmentation entities (line 1), which is an implementation of the branch step of fixpoint enumeration. An entity is chosen and the branch including that entity is fully explored first. Once returned from the recursion (line 18), the entity is added to the B set (line 21). Iteration of the closure operator is ensured by the *while* loop (lines 7–13)[7]. F^* and A^* are used to track entities that enter the augmentation set A'. Entities in the augmentation set have to be checked for inclusion with the closure operator only once for a specific combination of F and B, because the set of compatible entities C' does not change with the closure[8]. An MCCPS is outputted whenever $A = F$ (lines 15, 16).

The computation of the set of augmentation entities `Compute_Aug` (Algorithm 2) is straightforward: the set of compatible entities C is given, so we can take the entities of the adjacent $types$ from C (line 1) and any remaining entity $e \in C'$ leads to a *complete* and *connected* set $F \cup \{e\}$. Then, we only need to verify *properness* of $F \cup \{e\}$ by checking the *parents* of e (lines 2–6).

Algorithm 1 Enumerate all maximal CCPSs

```
Global static variables:
  – Comp List of compatible entities per entity
  – Rel_types_ent List of related entity types per entity type
  P-N-RMiner($F, B, C, A, types$)
1: for all $e \in A \setminus (F \cup B)$ do
2:     $types' \leftarrow types \cup Rel\_types\_ent[t(e)]$
3:     $C' \leftarrow F \cup \{e\} \cup$ Compute_Comp$(C \cap Comp[e] \setminus (F \cup \{e\}), F \cup \{e\})$
4:     $F^* \leftarrow F$
5:     $A' \leftarrow$ Compute_Aug$(C', F, types')$
6:     $F' \leftarrow F \cup \{e\} \cup$ Compute_Comp$(A' \setminus (F \cup \{e\}), C')$
7:     while $F' \setminus F^* \neq \emptyset$ do
8:         $types' \leftarrow types' \cup \bigcup_{e \in F' \setminus F^*} Rel\_types\_ent[t(e)]$
9:         $A^* \leftarrow A'$
10:        $F^* \leftarrow F'$
11:        $A' \leftarrow$ Compute_Aug$(C', F', types')$
12:        $F' \leftarrow F' \cup$ Compute_Comp$(A' \setminus (A^* \cup F'), C')$
13:    end while
14:    if $F' \cap B' = \emptyset$ then
15:        if $F' = A'$ then
16:            Output $F'$
17:        else
18:            P-N-RMiner$(F', B, C', A', types')$
19:        end if
20:    end if
21:    $B \leftarrow B \cup \{e\}$
22: end for
```

Algorithm 2 Enumerate the augmentation set of F (this assumes entities in C are compatible with F)

```
Global static variables:
  – Parents List of parents per entity
  Compute_Aug($C, F, types$)
1: $C' \leftarrow \bigcup_{t \in types} C_t$
2: for all $e \in C'$ do
3:     if $\neg(Parents[e] \subseteq F)$ then
4:         $C' \leftarrow C' \setminus \{e\}$
5:     end if
6: end for
7: return $C'$
```

For the compatible entities computation, we present pseudocode for the general n-ary case (Algorithm 3). `Compute_Comp` takes as arguments two sets of entities: G is the entities to verify for compatibility, and F is the set of entities to check compatibility against. The routine works by considering each entity $e \in G$ separately (line 3). Then, compatibility with F is checked for each relationship type that e can participate in (line 5). If the check fails for any relationship type, e is not compatible with F, the routine breaks

[7] The current implementation computes the closure only once, which probably negatively impacts the performance.

[8] NB. C, A, and $types$ are fixed also for given sets F and B.

Algorithm 3 Enumerate the entities in G that are compatible with F

```
Global static variables:
– Rel_inst List of relationship instance ids per relationship type per entity
– Entity_types List of entity types per relationship type
– Rel_types_types List of relationship types a given entity type participates in
Compute_Comp($G$, $F$)
1: $S \leftarrow \emptyset$
2: $F' \leftarrow \{f \in F \mid \nexists g \in F, f \in Parents[g]\}$
3: for all $e \in G$ do
4:     $insert$ ← true
5:     for all $r \in Rel\_types\_types[t(e)]$ do
6:         $T \leftarrow Entity\_types[r] \cap t(F) \setminus t(e)$
7:         if ¬Is_Comp($T$, $F'$, $Rel\_inst[e][r]$, $r$) then
8:             $insert$ ← false
9:             break
10:        end if
11:    end for
12:    if $insert$ then
13:        $S \leftarrow S \cup \{e\}$
14:    end if
15: end for
16: return $S$
```

(lines 7–9) and continues from line 3. Line 2 contains an optimisation that is explained below after introducing `Is_Comp`.

The compatibility check for the n-ary case is based on verification of the coverage of the critical sets of $F \cup \{e\}$ [21,23]. A *critical set* of a set of entities G and a relationship type r is any subset of G containing only entities present in r and at most one entity per type. A *maximal* critical set is one that contains as many entities as possible, i.e., one entity from each entity type of r that is present in G. A critical set $G' \subseteq G$ is *covered* if there exists a relationship instance $i \in R$, $G' \subseteq i$. If all maximal critical sets of G are covered, all critical sets of G are covered, and then and only then G is *complete*.

The function `Is_Comp` (Algorithm 4) checks whether all combinations of entities of types T in an entity set F—which could have more types than T—are covered by the set of relationship instances I of type r. The function works via recursion; if I becomes empty, the set is not covered (lines 1–3). If I is not empty, select an entity type $t \in T$ and check for every entity $e \in F$ of type t whether it is covered by recursion, while decreasing $|T|$ by one every time and selecting only the relationship instances of type r and the entity e (lines 4–9). Line 2 of `Compute_Comp` is an optimisation specific to P-N-RMiner because maximal critical sets that involve an entity e that is a parent of another entity f are covered by definition if the child f is covered[9].

[9] This optimisation is currently not in the implementation, and that probably negatively impacts the performance.

Algorithm 4 Check compatibility of all entities in F for specific entity types T and relationship type r

```
Global static variables:
– Rel_inst List of relationship instance ids per relationship type per entity
Is_Comp($T$, $F$, $I$, $r$)
1: if $I = \emptyset$ then
2:     return false
3: end if
4: Select any $t \in T$
5: for all $e \in F_t$ do
6:     if ¬Is_Comp($T \setminus t$, $F$, $I \cap$ Rel_inst[$e$][$r$],$r$) then
7:         return false
8:     end if
9: end for
10: return true
```

6 Case studies

The framework and theory presented in the previous sections give rise to several empirical questions, which we aim to address in this section through three case studies. Our primary contribution is the formalisation of a more general pattern syntax; hence, the primary question that we need to verify experimentally is:

1. Can we find patterns that are more interesting?

Our secondary contribution is the derivation of an interestingness score that accounts for the dependence between relationship instances of structured attributes. Hence, the second question is:

2. Is the new interestingness score relevant?

Thirdly, we present a novel enumeration algorithm. Given the appropriate input, the enumeration algorithm from N-RMiner [23] would output the same maximal CCPSs. However, we claim P-N-RMiner is faster, because it can capitalise on the partial order structure. Hence, the third question is:

3. Is the new enumeration algorithm faster?

We aim to answer the first two questions in the following case studies and also showcase the type of patterns that one can find using the method introduced in this paper. The third question we discuss in Sect. 7.

6.1 Foursquare check-ins

First we return to the Foursquare Check-ins data discussed in the introduction. These data were gathered by Cheng et al. [8] from several online social media but mostly (>50 %) from Foursquare, and consists of user-ids, check-in times, and venues. The data consists of 225K users and 22M recorded

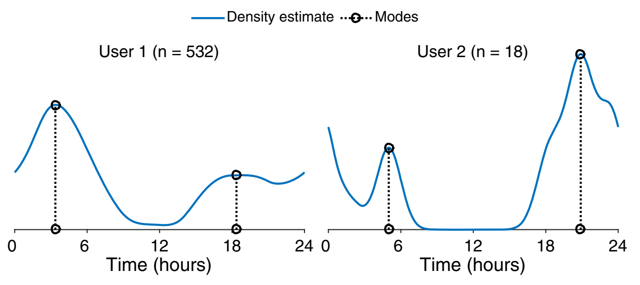

Fig. 3 Two examples of density estimation and mode location finding for the check-ins data

check-ins. Data such as this could be useful to identify patterns of people's mobility, busy times of certain services, etc. Ordinarily, we would represent this data using three entity types and triary relationship instances; user x checks in at time t into venue y. To make this example as simple as possible, we omit the information about venues.

In this case, we are interested in finding patterns in the check-in times across users such as *"many users check in somewhere both between 8.30 and 9.30 in the morning and between 11.30 and 12.30 around noon"*. Such patterns cannot be identified by running P-N-RMiner on the data directly, because relationship instances carry no weights or any information about their probability. Hence, users that check in frequently and are tracked over a long period of time will have checked in somewhere at many times of the day.

To remedy this, we preprocess the data by computing kernel density estimates for each user, using a Gaussian kernel with a width of one hour and then locating the modes of their check-in times with 10-minute precision. Two examples are visualised in Figure 3. As a result, instead of 22M check-ins, the relationship instances correspond to 684K modes, 3 per user on average. This way, more data ensure that our patterns will become more accurate.

We are interested in discovering patterns that possibly include time intervals and not just specific times. As possibilities, we considered asking P-N-RMiner to try intervals up to one, one and a half, and two hours. The reason we consider several options is because the more intervals there are, the more difficult the computational problem is. We identified for each interval size the largest subsampled data that we could run in less than 8 hours[10], using a reasonable constraint on minimum number of users in any CCPS, each time cutting the data size in half. We found these sample sizes to be 2^{-8} (879 users), 2^{-9} (440 users), and 2^{-10} (220 users), with minimum constraints of 0, 10, and 10 users in all patterns.

None of the settings yields substantially more interesting patterns than another. The 'up to 2-hour intervals' adds least information to the other two; more than half of the top-100 patterns for that setting contain only intervals that are shorter than 1.5 h and are thus also present in those results, and the interestingness scores are <0.815, while the top-65 for '$\leq$1-hour' and the top-26 for '$\leq$1.5-hours' have higher scores; up to 0.861 and 0.855, respectively. Notice that such scores are not straightforward to interpret, because whether such a score is low or high depends on the data at hand. For example, the pattern ranked 4th for '$\leq$2-hours' is interesting. It contains three intervals and reads: 4.5 % of the users checked in frequently between [1.10 am–2.30 am], [4.30 pm–6.30 pm], as well as [8.30 pm–9.30 pm].

The overall most informative pattern that we identify is: 1.6 % of the users checked in frequently between [6 am–7 am], as well as [10.10 am–10.50 am]. This means that, compared to the number of users that check in frequently between those intervals, there is a surprisingly large set of users that checks in frequently during both intervals. This pattern was found in the subsample of 879 users using intervals up to one hour in duration. Interestingly, in that case computing the results without constraints took 2 hours 20 minutes, but all except one pattern in the top-700 (ranked 269) have at least ten users, a result that can be computed in roughly half the time (1 h 13 m).

To confirm that handling intervals is relevant, we identified the top pattern that does not include any intervals; it is ranked 892nd, 2962nd, and 10138th, for the three settings, respectively. This clearly shows that patterns with intervals are more interesting in terms of information content. We also test the relevance of the new interestingness score, by comparing the ranking of P-N-RMiner against N-RMiner on data augmented such that they produce the same patterns. We find that Kendall's tau is 0.337 and 0.352, respectively (N-RMiner did not finish in the specified maximum of 8 hours on the third dataset), which highlights that accounting for the partial order when computing interestingness is highly relevant.

6.2 Amazon book ratings

As a second case study, we downloaded a snapshot of Amazon product reviews from SNAP[11]. This dataset contains around 500K products, 8M reviews with ratings from 1 to 5, and 2.5M product category memberships. From this we selected all reviews about books and uniformly sub-sampled 1 % of the customers.

All books have one or more category memberships, which are given as paths in the Amazon product category hierarchy. From this we extracted the relationship between books and categories and the hierarchy itself, keeping two levels below the category $\mathit{Book} \rightarrow \mathit{Subject}$. The dataset that we obtain has the structure shown in Figure 4 and consists of 22,003

[10] Unfortunately our current implementation does not use any parallelisation, so it runs only in a single thread.

[11] https://snap.stanford.edu/data/amazon-meta.html

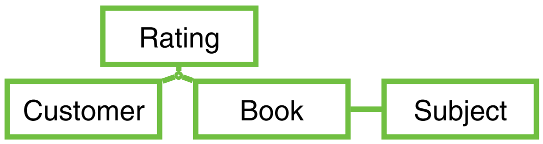

Fig. 4 Relational schema of the Amazon Book Ratings data

books, 9,855 customers, 417 hierarchically structured book subjects, as well as 36,415 ratings and 53,403 subject memberships.

We ran P-N-RMiner on this dataset with constraints of at least 6 books and 20 customers. As an example, we present the most highly-ranked pattern. This contains 23 customers and 8 books, all of which are different versions of the book "Left Behind: A Novel of the Earth's Last Days", a rating [1–5] and the subjects *Fiction* and *Christianity*. To our surprise, we found that most of the patterns in the result are like this; different versions of the same book (hard cover, audiobook, etc.).

Inspection of the raw data led us to the hypothesis that this happens because reviews are copied across different versions of the same book. Unfortunately, the text of reviews was not crawled, so it is not straightforward to identify reviews for different items that are equivalent. We attempted to tackle this problem by keeping only one such version of a book by looking for reviews that have the same date, rating, and user. However, after removing duplicates using this procedure, it appears that little structure remains in the data.

We also ran N-RMiner on the same dataset, augmenting it with all the implied relationship instances. We see that the same pattern is now ranked at the 21st position. This is because N-RMiner does not take into account the dependencies between the intervals and, as a result, intervals are by definition more highly connected and relationship instances containing intervals are more probable. This confirms that our new derivation of the interestingness score is indeed relevant.

6.3 Fisher's Iris data

The Iris data[12] have been pervasively used in machine learning and pattern recognition text books. The data consist of 150 measurements of plants. Each has four numerical attributes and a class label (one of three species). In Sect. 2, we have shown that P-N-RMiner can be used to mine tiles and frequent patterns. However, it can also be used to mine subgroups and subspace clusters, which we highlight in this case study.

Subgroup discovery is a form of pattern mining where a user chooses a target attribute and the aim is to find rules that predict high values of this attribute, or rules that predict *true* if the attribute is binary. For the Iris data, this means that we would like to find rules based on the four numerical attributes that predict a specific class label. We model the data as five entity types. We discretise each numerical attribute to ten different values using equal spacing and include intervals up to six adjacent values. This substantially reduces the computation time, while hardly affecting the patterns.

We then ran P-N-RMiner with a constraint that all patterns have to include a class label. The top pattern for each class is visualised in Figure 5. All top patterns include values for all four numerical attributes, indicating that they are all informative for the class label and the set of points that they describe. The first pattern that omits an attribute is ranked 120th and is equivalent to the second most informative pattern in the data (and second most informative for class 1), except that it omits *sepal width*. Figure 5 visually confirms that *sepal width* is the least informative feature for that pattern.

Subspace clustering is a form of pattern mining that is unsupervised. The goal is to discover clusters in the data, but unlike traditional clustering, the goal is not to provide a full partitioning of the data, and there is no requirement to use all variables. Our framework has roughly the same aim and could as such be considered a relational (exhaustive) approach to subspace clustering. Like in the case of the check-ins data, our framework enables identification of patterns that are otherwise unattainable using existing methods.

To find subspace clusters in the data, we ran P-N-RMiner without constraints on the Iris data, leaving out the class labels. As output we find 25,365 patterns. The top pattern is: $pl = [1.295–1.885]$, $pw = [0.22–0.7]$, $sl = [4.84–5.2]$, $sw = [3.08–4.04]$, with an interestingness score of 1.4744. So, it is similar to the top pattern predicting class 1 (see Figure 5), except that the intervals for *sepal length* and *sepal width* are slightly more narrow. The first *sub*space cluster occurs at rank 347 and is quite specific already: $pl = [1.295–1.885]$, $pw = [0.22–0.7]$, $sl = [4.48–5.2]$, with an interestingness score of 1.1040, again omitting *sepal width*.

As a final remark, we are not suggesting that P-N-RMiner can replace all existing subgroup discovery and subspace clustering methods, because P-N-RMiner has high computational cost, owing to the exhaustive search strategy. On the other hand, the advantage of exhaustive search is that the identified patterns are truly the most informative patterns in the data.

7 Scalability

To test the scalability of the algorithm and study what we gain by using the attribute structure in the form of the properness constraint, we again look at the check-ins data. We created

[12] https://archive.ics.uci.edu/ml/datasets/Iris.

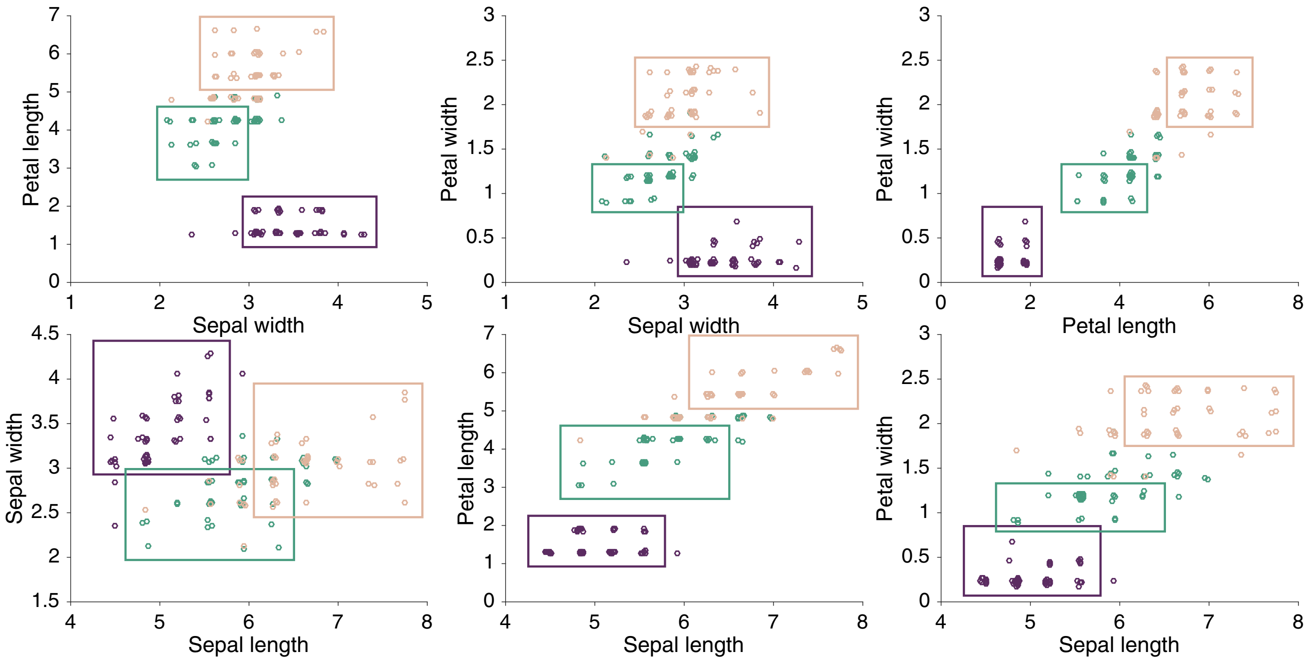

Fig. 5 Visualisation of the full Iris data, projected for each pair of features. Colours depict class labels and the boxes represent the top subgroup pattern for each class, as discovered by P-N-RMiner. Incidentally, each most informative pattern includes all four attributes

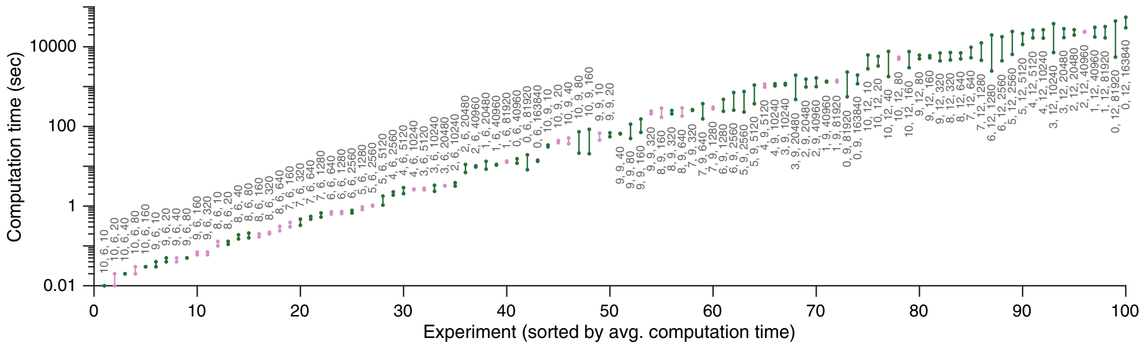

Fig. 6 Comparison of computation times between P-N-RMiner (introduced here) and N-RMiner [23]. On the x-axis are experiments, sorted by average computation time over the two methods, and on the y-axes is computation time. The runtime of each experiment is visualised by two dots, corresponding to the runtime of the two methods. For dark/green bars and dots, P-N-RMiner is the lower dot of the two (and thus faster), while for light/pink bars and dots, N-RMiner is lower dot of the two (and thus faster). P-N-RMiner is typically faster, but especially for problems that are computationally more demanding

11 versions of the data, each time throwing away half of the remaining users and their relationship instances (the check-in modes). We then ran both N-RMiner on augmented data with the additional entities and relationship instances, and P-N-RMiner, which then output the same set of maximal CCPSs.

We are interested also in how the depth of the partial order of an attribute affects the scalability and potential speed-up by P-N-RMiner. Hence, we tested runtimes for 6, 9, and 12 levels, i.e., time intervals up to one, one and a half, and two hours. We exhaustively tested constraints on the number of users from 10, 20, 40, etc. up to the sample size. We stopped any experiment that had not finished after 24 hours.

A comparison of runtimes for all cases where N-RMiner finished succesfully is given in Figure 6. The general trend is that P-N-RMiner in faster in 71 out of this subset of 100 experiments and that the speed-up grows as the computation time grows. The largest observed speed-up is a factor of 8; 12.3 vs. 1.5 hours, for the full data, 12 levels, and mining patterns with at least 81920 users. N-RMiner is mainly faster (up to a factor of 2) for runtimes shorter than 1 second. Not shown in the figure is that P-N-RMiner uses substantially less memory. For example, for the full data with 12 levels, P-N-RMiner uses 5.5 MB of memory at its peak, while N-RMiner uses more than 10 GB.

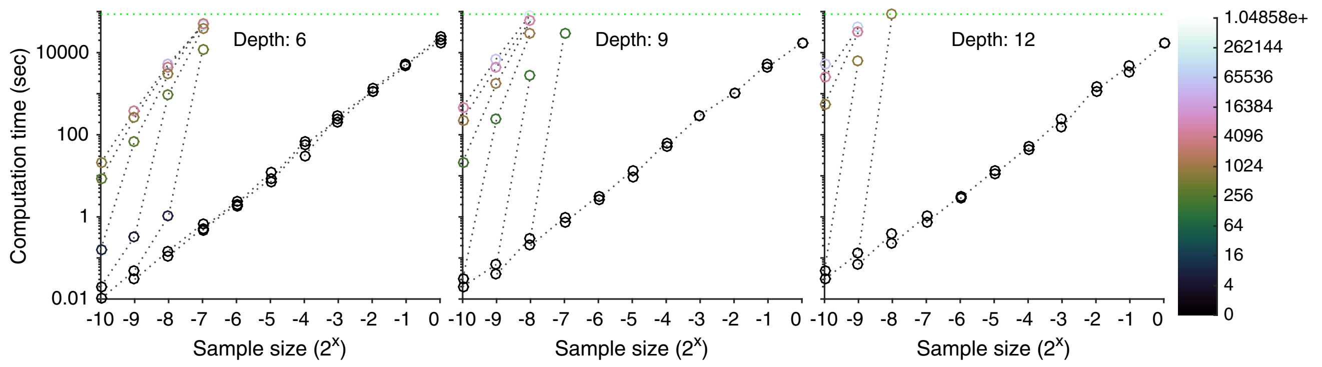

Fig. 7 Scalability of P-N-RMiner for increasing data size. Each dotted line with circles corresponds to a constraint on the minimum number of users. For any such constraint there are typically two or three measurements (sample sizes) that finished in 24 hours, due to the constraint being too low and demanding, or too high (larger than the sample size). On the x-axis is the sample size, on the y-axis the runtime. The colour of a circle corresponds to the number of patterns (CCPSs) enumerated, ranging from 0 (black) to ca. 250k (light blue), see also the colour bar. There appears to be linear relation between runtime and sample size when the number of patterns is the same, and the depth of the partial order structure has a limited effect as well

Runtimes of P-N-RMiner for increasing sample sizes are illustrated in Fig. 7. It appears that the number of levels, i.e., the depth of the partial order, actually only has a small effect on the runtime—for example, compare the trend of brown points across the depths. Yet, the number of patterns in the data explodes much more easily. For depth 12, we can only enumerate all patterns in the samples that have at most 879 users (0.4 % of the data). There appears to be a linear relationship between the size of the data and the runtime if the number of patterns is equivalent; for any of the subfigures, one could fit a straight line through measurements that have roughly the same number of patterns (i.e., points of the same colour). This relationship holds even when there are no patterns.

8 Related work

8.1 Exploratory versus predictive patterns

The broad purpose of the framework presented in this paper is to facilitate *exploration* of data in an entirely unsupervised manner. This distinguishes the framework from other types of local pattern for multi-relational data mining such as Safarii [15], and more generally from approaches based on inductive logic programming. These alternative frameworks operate by a user selecting one or a set of attributes as a target, after which an algorithm builds rules to predict that target using the full relational data. Wu et al. [27] introduced a method for finding interesting chains of biclusters in relational data, which has a similar goal as our framework. Their approach differs in that they only consider binary relationships, they employ a heuristic greedy algorithm to find interesting patterns, and their method does not account for structure of attributes in any way.

8.2 Pattern syntax

The pattern syntax proposed in this paper is unique in being both relational and able to deal with structured attribute types such as ordinal and real-valued attributes, taxonomy terms, and more. The proposed pattern syntax, in being *local*, owes to the frequent pattern mining literature. Indeed, the CCS pattern syntax [24], which it generalises, has already been shown to be a generalisation itself of a local pattern type in binary databases known as *tiles* [12], which are essentially equivalent to frequent itemsets.

8.3 Structured attribute types

Real-valued and ordinal attributes have also been dealt with before in local pattern mining, in subgroup discovery and exceptional model mining. For example, in subgroup discovery, approaches have been developed to infer subgroup descriptions in terms of intervals for real-valued attribute types and subsets of categorical attributes. A notable paper in this regard is [19], where an efficient algorithm is introduced for finding optimal subgroups using any convex quality measure. Exceptional model mining, on the other hand, aims to extend subgroup discovery beyond a single target attribute [17]. None of these approaches, however, are as generic as our proposed approach: they are either ad hoc or remain limited to a very specific types of structured attributes. The approach of modelling the structure of the attributes as a partial order is also entirely novel.

8.4 Interestingness formalisations

The formalisation of interestingness of local patterns is a highly active research area, with most research targeted on itemsets in binary databases. This makes sense, as the problem is most acute for exploratory data mining approaches, in the absence of a particular set of target attributes to be predicted. Many approaches to formalising interestingness are based on modelling the *unexpectedness* of a pattern: the extent to which the pattern presents novel, surprising, or unexpected information to the user. A recent survey is [16].

There are three major lines of research aimed at mining (sets of) 'interesting' local patterns. Constrained randomisation techniques are based on the assumption that a pattern is more interesting if it is not present in randomised data [13,18,20]. Methods based on the Minimum Description Length principle assume that a pattern is more interesting if provides better compression [26]. Approaches based on the Maximum Entropy (MaxEnt) principle assume a pattern is more interesting the more surprising it is given a MaxEnt-based background model [9,10]. Both randomisation and MaxEnt approaches have been shown to allow for accounting prior knowledge, thus enabling subjective interestingness and iterative data mining.

The MaxEnt approach and the subjective interestingness framework FORSIED have been shown to be highly flexible in terms of pattern types [11]. Additionally, they have been used successfully to quantify interestingness patterns for RMiner [24], which we directly build upon. For these reasons we used this paradigm to formalise the interestingness of the patterns in the current paper. Clearly, a direct application of interestingness as defined in [24] would not have yielded desirable results, as the dependencies between relationship instances would be ignored (see also Sect. 6).

In the work on Domain Driven Data Mining [3,5], it is stressed that there is a difference between technical and business interestingness. For patterns to be *actionable*, technical interestingness often does not suffice and patterns are only truly interesting if they reveal relations that are directly related to the business model, i.e., they take into account domain knowledge of the business [4]. Furthermore, a distinction is made between *objective* and *subjective* interestingness. Notably, in that line of work there are also results on mining patterns across data tables, called *combined mining* [6].

It is important to note that the FORSIED framework [9,11] attempts to integrate objective and subjective interestingness by means of an objective score function that explicitly accounts for prior beliefs specified by the user. We have so far assumed that the user wants to learn everything about the data and largely ignored what to do if the user is interested only in (relationships to) part of the data. We envision that in our framework it should be possible to integrate both technical and business interestingness. It seems possible to manipulate the constraints on the minimum number of entities of certain types as well as the prior beliefs to ensure only patterns are found that are indeed interesting to the end user, whatever the context. However, further research in this direction is necessary.

8.5 Enumeration algorithms

The algorithm that we derived for enumeration of maximal CCPSs is based on the generic fixpoint-enumeration algorithm for enumerating all closed sets in a strongly accessible set system, introduced by Boley et al. [1]. This algorithmic scheme has been used before in the data mining literature for enumerating maximal CCSs [24], including extensions to n-ary relations [23] and approximate CCSs [22]. Here, we adhere to the same algorithmic scheme. In order to be able to use the scheme, we model the structure of attributes as a partial order, augment the pattern syntax, and add a properness constraint to the definition of the set of augmentation elements. As may be apparent, these changes are not trivial, and neither is the proof that the algorithmic scheme still works.

9 Conclusions

An important obstacle for the adoption of exploratory data mining techniques in general, and local pattern mining approaches in particular, is their limited flexibility in terms of data type to which they can be applied (e.g., only tabular data), and type of pattern they can generate, e.g. subgroups, itemsets, n-sets. In reality, however, data are often complexly structured (as in, e.g., a relational database), and additionally there is often structure among the different values data attributes may attain, i.e., attribute values can be ordinal, interval, taxonomy terms, and more.

Attempts to resolve this inflexibility for specific data and pattern types are numerous. Yet, we are unaware of any generic approach that comes close to subsuming the range of pattern syntaxes considered by the local pattern mining research community, allowing for data types of a broad range of structures. The contributions made here may be an important step in this direction.

Our contributions raise a number of new research challenges. Ideally, the pattern syntax is tolerant to missing relations to ensure noise resilience, similar to [22]. The interestingness can be made more versatile by considering a more varied range of prior belief types. Another interesting question is whether the enumeration algorithm could still be improved. Our algorithm is similar to the Bron-Kerbosch algorithm for enumerating maximal cliques in a graph, for which it is known that the worst case complexity of $O(3^{n/3})$ is optimal, since it is equivalent to the number of maximal cliques in a graph [25]. Yet another interesting direction for future work is developing heuristic algorithms for finding interesting CCPSs directly, in order to avoid the costly exhaustive search step.

Acknowledgments This work was supported by the European Union (ERC Grant FORSIED 615517) and the EPSRC (Grant EP/M000060/1).

References

1. Boley, M., Horváth, T., Poigné, A., Wrobel, S.: Listing closed sets of strongly accessible set systems with applications to data mining. TCS **411**(3), 691–700 (2010)
2. Borgelt, C.: Frequent item set mining. WIREs: DMKD **2**(6), 437–456 (2012)
3. Cao, L.: Domain driven data mining (D3M). In: ICDM Workshops, pp. 74–76 (2008)
4. Cao, L.: Domain-driven data mining: Challenges and prospects. IEEE TKDE **22**(6), 755–769 (2010)
5. Cao, L., Yu, P.S., Zhang, C., Zhao, Y.: Domain Driven Data Mining. Springer, New York (2010)
6. Cao, L., Zhang, H., Zhao, Y., Luo, D., Zhang, C.: Combined mining: Discovering informative knowledge in complex data. IEEE TSMC-B **41**(3), 699–712 (2011)
7. Cerf, L., Besson, J., Robardet, C., Boulicaut, J.F.: Data peeler: Constraint-based closed pattern mining in n-ary relations. In: Proceedings of SDM, pp. 37–48 (2008)
8. Cheng, Z., Caverlee, J., Lee, K., Sui, D.Z.: Exploring millions of footprints in location sharing services. In: Proc. of ICWSM, pp. 81–88 (2011)
9. De Bie, T.: An information-theoretic framework for data mining. In: Proceedings of KDD, pp. 564–572 (2011)
10. De Bie, T.: Maximum entropy models and subjective interestingness: an application to tiles in binary databases. DMKD **23**(3), 407–446 (2011)
11. De Bie, T.: Subjective interestingness in exploratory data mining. In: Proceedings of IDA, pp. 19–31 (2013)
12. Geerts, F., Goethals, B., Mielikäinen, T.: Tiling databases. In: Proceedings of DS, pp. 278–289 (2004)
13. Gionis, A., Mannila, H., Mielikäinen, T., Tsaparas, P.: Assessing data mining results via swap randomization. TKDD **1**(3), 14 (2007)
14. Herrera, F., Carmona, C.J., González, P., del Jesus, M.J.: An overview on subgroup discovery: foundations and applications. KAIS **29**(3), 495–525 (2011)
15. Knobbe, A.J.: Multi-relational data mining. IOS Press, Amsterdam (2006)
16. Kontonasios, K.N., Spyropoulou, E., De Bie, T.: Knowledge discovery interestingness measures based on unexpectedness. WIREs DMKD **2**(5), 386–399 (2012)
17. Leman, D., Feelders, A., Knobbe, A.: Exceptional model mining. In: Proceedings of ECML-PKDD, pp. 1–16 (2008)
18. Lijffijt, J., Papapetrou, P., Puolamäki, K.: A statistical significance testing approach to mining the most informative set of patterns. DMKD **28**(1), 238–263 (2014)
19. Mampaey, M., Nijssen, S., Feelders, A., Knobbe, A.: Efficient algorithms for finding richer subgroup descriptions in numeric and nominal data. In: Proceedings of ICDM, pp. 499–508 (2012)
20. Ojala, M., Vuokko, N., Kallio, A., Haiminen, N., Mannila, H.: Randomization of real-valued matrices for assessing the significance of data mining results. Proc SDM **8**, 494–505 (2008)
21. Spyropoulou, E.: Local pattern mining in multi-relational data. Ph.D. thesis, University of Bristol (2013)
22. Spyropoulou, E., De Bie, T.: Approximate multi-relational patterns. In: Proceedings of DSAA, pp. 477–483 (2014)
23. Spyropoulou, E., De Bie, T., Boley, M.: Mining interesting patterns in multi-relational data with N-ary relationships. In: Proceedings of DS, pp. 217–232 (2013)
24. Spyropoulou, E., De Bie, T., Boley, M.: Interesting pattern mining in multi-relational data. DMKD **28**(3), 808–849 (2014)
25. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. TCS **363**(1), 28–42 (2006)
26. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: mining itemsets that compress. DMKD **23**(1), 169–214 (2011)
27. Wu, H., Vreeken, J., Tatti, N., Ramakrishnan, N.: Uncovering the plot: detecting surprising coalitions of entities in multi-relational schemas. DMKD **28**(5–6), 1398–1428 (2014)