Mining Closed Patterns in Relational, Graph and Network Data

Gemma C Garriga, Roni Khardon, Luc De Raedt

Mining Closed Patterns in Relational, Graph and Network Data

Gemma C. Garriga
INRIA Lille Nord Europe
Lille, France
gemma.garriga@inria.fr

Roni Khardon*
Department of Computer Science
Tufts University, USA
roni@cs.tufts.edu

Luc De Raedt†
Department of Computer Science
Katholieke Universiteit Leuven, Belgium
luc.deraedt@cs.kuleuven.be

Abstract

Recent theoretical insights have led to the introduction of efficient algorithms for mining closed item-sets. This paper investigates potential generalizations of this paradigm to mine closed patterns in relational, graph and network databases. Several semantics and associated definitions for closed patterns in relational data have been introduced in previous work, but the differences among these and the implications of the choice of semantics was not clear. The paper investigates these implications in the context of generalizing the LCM algorithm, an algorithm for enumerating closed item-sets. LCM is attractive since its run time is linear in the number of closed patterns and since it does not need to store the patterns output in order to avoid duplicates, further reducing memory signature and run time. Our investigation shows that the choice of semantics has a dramatic effect on the properties of closed patterns and as a result, in some settings a generalization of the LCM algorithm is not possible. On the other hand, we provide a full generalization of LCM for the semantic setting that has been previously used by the Claudien system.

1 Introduction

In recent years there has been a growing interest in learning and mining of relational data, that is, information that is organized by specifying objects and relations among them. The data is often organized in one of two forms as illustrated by the following examples.

- In the National Center for Toxicological Research Estrogen Receptor Binding dataset (Fang et al., 2001; Blair et al., 2000; Branham et al., 2002) (NCTRER), each example is a molecule described by a graph of its atoms and bonds. Here the objects are the atoms and the relations are the bonds between atoms. In this case, each example is a separate graph and the dataset is a collection of graphs. This setting is known in the data mining community as *graph mining*. It is a special case of the *learning from interpretations* (LI) setting in inductive logic

*Partly supported by NSF Grant IIS-0099446, and by a Research Semester Fellowship Award from Tufts University.
†Partly supported by the EU IST FET project IQ.

programming (De Raedt, 2008), where the relations may refer to more than two objects and where multiple relations are used to annotate the data.

- The Cora database (McCallum et al., 1999) is a compilation of research articles in computer science, their topics, authors, and citations among them. In this case, the objects are the articles, authors and topics, and the relations capture authorship of papers, citations among papers, and paper topic association. Thus, we have several relations over a set of objects, that is, a large network of relations. The analysis of such data is known in the data mining community as *network mining*. In the more general case studied in inductive logic programing, one can have an arbitrary number of relations and each relation may have a large number of arguments. To distinguish this setup from the previous case, we call this the *single interpretation* (SI) setting below.

Throughout this paper, we shall largely employ the terminology of inductive logic programming (Muggleton & De Raedt, 1994; Nienhuys-Cheng & De Wolf, 1997; De Raedt, 2008), because it is more general. Nevertheless, it is important to keep in mind that the work we present directly applies to graphs and networks as they are studied in contemporary data mining.

Many data mining tasks have been studied for this type of data. The one we focus on in this paper is the problem of mining frequent patterns. In this problem the goal is to find patterns, corresponding to sub-graphs in our examples, that *occur frequently* in the database, where *occur* and *frequently* are defined appropriately. For the NCTRER dataset, this means identifying sub-molecules that occur in many of the molecules. For instance, a certain nitrogen-based fragment could be common in the data. This type of data mining has been used as a preprocessing step to extract patterns from the dataset, and these patterns have often been used as features in classifiers (Deshpande et al., 2003; Kramer & De Raedt, 2001). For the Cora dataset, frequent patterns correspond to trends in the data such as "authors of neural networks papers are also authors of image processing papers". This type of network mining has been studied before, with different applications, for example, in web-graph mining (Bringmann & Nijssen, 2008; Kuramochi & Karypis, 2004; Fiedler & Borgelt, 2007).

Our problem may be roughly stated as follows: the data describes multiple relations among objects and can be organized in LI or SI form (a set of graphs or a single network); the goal is to identify frequently occurring patterns. This is a generalization of the problem of mining frequent item-sets that has been a topic of intensive research (see e.g. Agrawal et al., 1996; Goethals & Zaki, 2004; Han et al., 2000). When mining frequent item-sets, there is a single relation whose arguments give an exhaustive list of items and whose tuples describe subsets of items. For example, in the celebrated market-basket database (Agrawal et al., 1993) arguments are items for sale in a supermarket and a tuple is a 0-1 assignment of values to arguments giving the set of items purchased by a single customer. Frequent item-sets are groups of items that are bought together for a significant proportion of customers and such item-sets can be used in marketing decisions. Since the number of frequent item-sets is huge, it is common to first search for the maximal frequent sets (that is, for the largest sets that are frequent); the maximal sets capture all frequent sets since subsets of frequent sets are frequent (Mannila & Toivonen, 1997; Gunopulos et al., 2003). In addition, several authors have shown that it is often more efficient to search for the *closed sets* (Bastide et al., 2000; Pei et al., 2000; Zaki, 2004; Zaki & Hsiao, 2002), which include all maximal frequent sets. The collection of frequent closed sets contains the same information as the overall collection of frequent item-sets, but is much smaller (Boros et al., 2003).

Of particular interest in this paper is the LCM algorithm (Linear time Closed item set Mining) (Uno et al., 2004) which has interesting theoretical properties and has been shown to be very effective in practice. In the item-set case, a set is not closed if its existence in a tuple implies the existence of other items w.r.t. the data. Therefore, a set is closed if it cannot be extended in this way with new items that appear in the same set of tuples. Consider for example the dataset T given in Figure 1. In the example, the dataset T has six tuples where items in a tuple are simply given by a numerical identifiers from 1 to 9. The set $\{1\}$ is not closed since whenever 1 appears in the data both 7 and 9 appear as well. On the other hand, the set $\{1, 7, 9\}$ is closed as it is maximal for the set of tuples where it appears. The LCM algorithm relies on three major points to find the closed sets efficiently:

(1) A subroutine closure(X) that calculates a unique closed set from the set X.

(2) A refinement step takes a closed set X and adds an item to it to get another (possibly) non closed set Y. The set Y can then be closed by calculating $Z = \text{closure}(Y)$.

(3) Every closed set has a unique parent in this process. This is achieved through a construction called Prefix Preserving Core extensions (*PPC extensions*) described in detail in Section 4.

Given these properties, the algorithm can calculate all frequent closed sets by recursively refining closed sets and calculating their closure. In this process a closed set may be reached in several ways but its "true parent" is identified by property (3) so that the set is refined exactly once. Thus, property (3) guarantees that we get unique outputs without explicitly storing closed sets in memory, an important property reducing the memory requirement of the algorithm and leading to efficient implementations. The operation of the algorithm is illustrated in Figure 1, where the nodes correspond to closed sets, edges record refinements and dark edges (blue in color diagram) are the ones corresponding to PPC extensions. As the discussion suggests, a node has more than one incoming edge, thus it is discovered more than once, but each node has exactly one incoming edge that is blue. This example is explained in more detail in the technical section once appropriate definitions are given.

The notions of frequent sets and closed sets have already been imported to the richer relational and graph-based setup. However, there are several possible ways to capture these notions and different authors have used different definitions. Thus, there is no general agreement in the literature on the choice of definition or the implications of using one of these possibilities. In particular, one distinction is between the LI and SI settings illustrated above. A second difference exists in the notion of coverage. Most approaches to data mining employ the object identity subsumption, corresponding to subgraph isomorphism in the graph and network case, where two nodes in a pattern cannot match the same node in the data (Malerba & Lisi, 2001; Nijssen & Kok, 2003; Yan & Han, 2003; Kuznetsov & Samokhin, 2005). Others, especially in inductive logic programming, employ the usual θ-subsumption from logic, that corresponds to graph homomorphism in the graph setting (Dehaspe & Toivonen, 2000). Finally, some authors (e.g. De Raedt & Ramon, 2004; De Raedt & Dehaspe, 1997) restrict the implication relation to so-called range-restricted clauses, where one cannot conclude properties of unobserved objects. From the literature one might be tempted to think that the choice of semantics is not important and that algorithmic issues can be studied independently of the semantics, but as we show in this paper, this is not the case.

This paper investigates the possibility of generalizing the LCM algorithm for mining relational data. Our investigation shows that the choice of semantics has a dramatic effect on the properties of

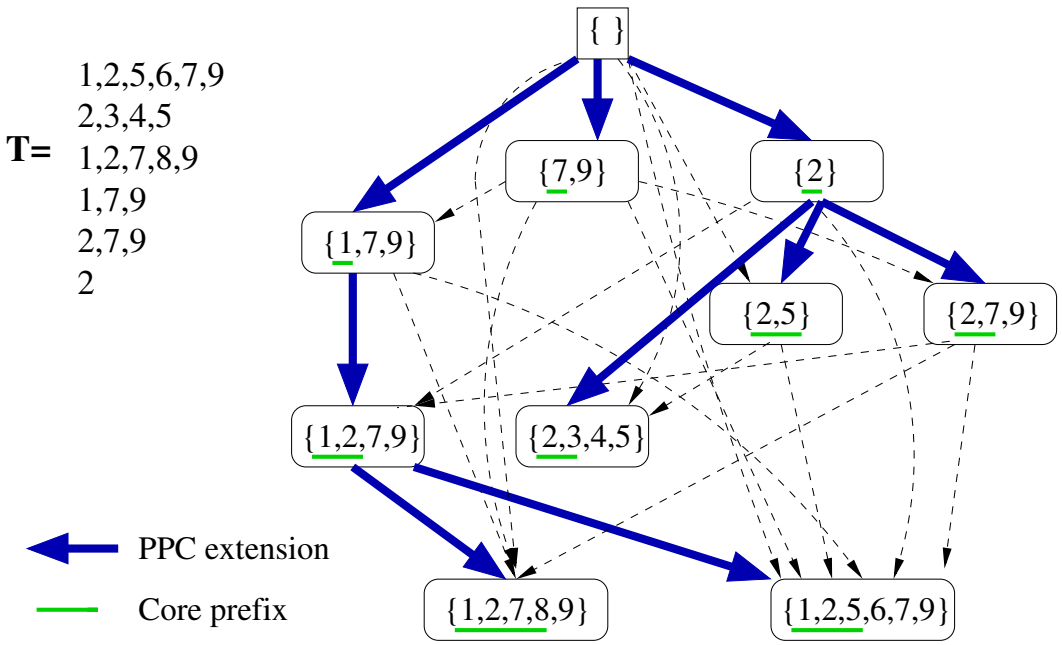

Figure 1: Example of the LCM algorithm for mining closed item-sets. Nodes are closed sets and edges define refinement steps. Dark edges correspond to refinements that are PPC extensions.

closed sets and as a result also to the potential generalization of the LCM algorithm. In particular, in the setting defined for CloseGraph (Yan & Han, 2003) and Farmer (Nijssen & Kok, 2003), using LI and object identity, closures of patterns are not unique. As a result, LCM cannot be applied and this setting is limited to methods that search by applying refinements (using breadth-first search (BFS) or depth-first search (DFS)) until a closed set is found. For the setting of Warmr (De Raedt & Ramon, 2004; Dehaspe & Toivonen, 2000) and Jimi (Maloberti & Suzuki, 2003), using LI and θ-subsumption, one can define unique closures, but PPC-extension properties do not hold. In this case one can enumerate closed sets as in LCM; however, one must store previously discovered sets to avoid duplicates in the output. Finally, for the setting of Claudien (De Raedt & Dehaspe, 1997), using SI and range restriction (with either object identity or θ-subsumption), we provide a full generalization of the LCM algorithm that does not need to store previous sets in the enumeration. The Claudien setting is also closely related to recent efforts to improve efficiency in network mining (Bringmann & Nijssen, 2008). As we show below, one can use the new frequency notion of Bringmann and Nijssen (2008) with LCM, but we cannot use the implicitly assumed underlying semantics, so that closures must still be calculated relative to the SI semantics.

The rest of the paper is organized as follows. The next section provides technical definitions of the different settings, makes some initial observations and develops a normal form for conjunctions that is needed by the algorithms. Sections 3 and 4 develop the solutions for the LI and SI settings respectively. The final section concludes with a discussion of the results and perspective for future work. The paper provides a theoretical investigation of the ideas described above. Throughout the text we develop the main results and provide additional comments and examples that we believe are important for an in depth understanding of the questions investigated. Some of these comments,

marked as *remarks* below, can be skipped in first reading without losing continuity for the main results.

2 Problem Definition and Basic Insights

The problems we consider assume that we have a given database $\mathcal{D}$, a language of patterns $\mathcal{L}$ and a notion of "frequency" measuring how often a pattern occurs in the database. Our databases and patterns are expressed in logical notation. We assume some familiarity with basic notions of logic (Lloyd, 1987) but the following introduces some notation and terminology. We use logical terms to define the relevant notions for this paper, because it is more general than that based on graph theoretic concepts. However, where relevant, we point out the correspondences in terminology.

An *atom* is an expression of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol, also called *relation* of arity n and each t_i is a *term*, i.e. a constant or a variable. Thus, in terms of logic, we do not allow functions symbols except for constants. To simplify notation, we sometimes use the symbol p to denote an atom $p(t_1, \ldots, t_n)$.

In this paper data is given as an *interpretation* that specifies a domain of objects (identified with constants) and the truth values of all predicates on this set of objects. In our context an interpretation can be identified with the set of atoms that are true in it $\{p_1, \ldots, p_n\}$, and in the same way also with the conjunction of true atoms $p_1 \wedge \cdots \wedge p_n$. Therefore, when no confusion can arise we refer to an interpretation, its set of true atoms, and the conjunction of those atoms interchangeably. For example, a labeled cycle graph of three nodes could be represented as: $e(1,2,a) \wedge e(2,3,b) \wedge e(3,1,c)$, where $1, 2, 3, a, b, c$ are constants. Here the predicate e represents an edge between the nodes provided by first two arguments of the predicate, and with the edge label given by the third argument. This conjunction corresponds to the interpretation with objects $\{1, 2, 3, a, b, c\}$ and true atoms $\{e(1,2,a), e(2,3,b), e(3,1,c)\}$.

Conjunctions of atoms also serve as patterns (where terms can be variables) which are sometimes referred to as *queries*. An example of a query, capturing a walk of length two with edge labels a, b, is $e(x,y,a) \wedge e(y,z,b)$, where x, y, z are variables. Notice that when all relations are of arity 0, conjunctions correspond to item-sets. Notice also that if we only have one binary predicate, we can consider it as capturing the edge relation of a graph, so graphs and graph patterns form another special case of relational patterns.

We shall represent conjunctions of atoms in list notation, i.e. as $[p_1, \ldots, p_n]$. For a conjunction C and atom p, by $[C, p]$ we refer to the conjunction that results from adding p after the last element of C. For a conjunction $C = [p_1, \ldots, p_n]$ and index $i \leq n$, $C[i]$ is the *i-prefix* of C, i.e. $[p_1, \ldots, p_i]$. A *substitution* $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ maps variables to terms. The result of applying a substitution θ to an expression C is the expression $C\theta$, where all variables x_i have been simultaneously replaced by their corresponding term t_i in θ.

2.1 Semantics

The syntax given above does not allow for negation of atoms in queries. Therefore, to relate the data to a query, we can employ the notion of subsumption, that relates query atoms to data atoms. Within inductive logic programming, two standard notions exist: θ-subsumption and OI-subsumption. Let C_1 and C_2 be two sets of atoms.

θ-subsumption (Plotkin, 1970): C_1 θ-subsumes C_2 if and only if there exists a substitution θ such that $C_1\theta \subseteq C_2$.

OI-subsumption (Malerba & Lisi, 2001): C_1 OI-subsumes C_2 if and only if there exists a substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ such that $C_1\theta \subseteq C_2$ and the t_i are different terms not occurring in C_1.

We will refer to both cases as subsumption, denoted as $C_1 \preceq C_2$ and when we want to identify the substitution witnessing the subsumption we say that $C_1 \preceq C_2$ via substitution θ. In our context we apply subsumption when C_1, C_2 are conjunctions and often when C_2 is an interpretation D. It is easy to see that $C_1 \preceq D$, means that D is a model of C_1 (where the variables in C_1 are existentially quantified), which is denoted $D \models C_1$ in standard logic notation. Similarly, for a conjunction C_2, $C_1 \preceq C_2$ means that C_2 implies C_1. Applied to graphs expressed in Datalog, the former notion corresponds to *graph homomorphism*; the latter one to *subgraph isomorphism.*[1]

Example 1 Consider the sets of atoms $C_1 = [p(x, y), q(y)]$, $C_2 = [p(a, b), q(b)]$, and $C_3 = [p(a, a), q(a)]$. Then, C_1 θ-subsumes both C_2 (with $\theta = \{x/a, y/b\}$) and C_3 (with $\theta = \{x/a, y/a\}$). C_1 also OI-subsumes C_2, but does not OI-subsumes C_3.

Two forms of databases will be considered. In *learning from (multiple) interpretations* (LI), the database $\mathcal{D}$ contains a set of interpretations (or, a set of graphs). In this case the natural notion of coverage is:

$$\text{covers}_{LI}(C, \mathcal{D}) = \{D \in \mathcal{D} | C \preceq D\}$$

where each interpretation contributes one element to the cover set. In this case $\mathcal{D}$ represents a set of graphs as in the NCTRER dataset mentioned in the introduction; the set $\text{covers}_{LI}(C, \mathcal{D})$ consists of the graphs from $\mathcal{D}$ that contain (cover) the graph pattern expressed in C. This is similar to the notion of support in frequent subgraph mining.

In the *single interpretation* (SI) (or, the network) setting, the database $\mathcal{D}$ is a single conjunction and the natural notion of coverage is:

$$\text{covers}_{SI}(C, \mathcal{D}) = \{\theta | C \preceq \mathcal{D} \text{ via substitution } \theta\}.$$

In this case $\mathcal{D}$ represents one single large graph or network as in the Cora dataset described in the introduction; the set $\text{covers}_{SI}(C, \mathcal{D})$ consists of all the mappings showing that the subgraph C is embedded in the large graph $\mathcal{D}$. This corresponds to the notion of occurrence of a pattern for network mining. Notice that here the cover set includes substitutions and not interpretations. In both definitions $\preceq$ can be either of the notions of subsumption given above.

We now have four different semantics. The LI-θ setting learns from interpretations using θ-subsumption. This closely corresponds to the setting employed by Warmr (Dehaspe & Toivonen, 2000) and Jimi (Maloberti & Suzuki, 2003). The LI-OI employs OI-subsumption instead and is employed by CloseGraph (Yan & Han, 2003) and Farmer (Nijssen & Kok, 2003). Finally, the SI-θ and SI-OI settings learn from a single interpretation only. As we shall see, SI-θ corresponds to Claudien's setting (De Raedt & Dehaspe, 1997).

[1] The standard encoding of an unlabeled graph as a database uses one binary predicate to capture the edge relation. In this case, OI-subsumption requires us to use a one to one mapping of nodes in graph G_1 into another graph G_2 and therefore checks for an isomorphic embedding of G_1 in G_2 (although this does not directly check for omission of edges). θ-subsumption allows for a mapping which is not one to one and therefore corresponds to homomorphism.

2.2 Frequent Sets

The problem of finding frequent queries is that of finding all queries $C \in \mathcal{L}$ whose frequency $\text{freq}(C, \mathcal{D}) \geq t$, for some fixed threshold t. The natural notion for frequency counts the total size of the covered set as follows,

$$\text{totfreq}(C, \mathcal{D}) = |\text{covers}(C, \mathcal{D})|.$$

It is easy to see that totfreq is anti-monotonic for the LI setting: adding an atom to a conjunction can only decrease the frequency since an interpretation covered before might not be covered with the additional constraint. This is an important property since one can use it to prune the search for frequent patterns and it is in fact used by all frequent set miners.

Example 2 The measure $\text{totfreq}(C, \mathcal{D})$ is not anti-monotonic for the SI setting. To see this, consider the query $[q(x)]$ which subsumes $[q(x), p(x, y)]$. For the interpretation $[q(a), p(a, b), p(a, c)]$, the query $[q(x)]$ has one substitution (that is, $\theta = \{x/a\}$) but $[q(x), p(x, y)]$ has two substitutions ($\theta = \{x/a, y/b\}$ or $\theta = \{x/a, y/c\}$).

This problem has received some attention in the network mining literature (Bringmann & Nijssen, 2008; Kuramochi & Karypis, 2004; Fiedler & Borgelt, 2007) and there are several alternative formulations for anti-monotonic notions of frequency. It is natural to use the notion of relative frequency defined as

$$\text{relfreq}(C, \mathcal{D}) = |\text{covers}(C, \mathcal{D})|/|D|^v = \text{totfreq}(C, \mathcal{D})/|D|^v,$$

where D is the domain (the set of constants) or a bound on its size, and v the number of variables appearing in C. This scales the absolute coverage by its maximum possible value so that patterns with different number of variables are in the same scale. Relative frequency is intuitively appealing and it is anti-monotonic with respect to adding atoms for the SI setting. In particular, if we add an atom with new variables then we can expand existing substitutions by a factor of at most D per variable and the denominator grows by the same factor. For databases with objects of different types one can consider the variables to be typed and consider an appropriate variation of D^v, where only legal substitutions are counted.

Another approach (Bringmann & Nijssen, 2008) uses a notion of frequency based on single variables:

$$\text{projfreq}(C, \mathcal{D}) = min_{v \in Vars(C)} |\text{projcovers}(C, v, \mathcal{D})|$$

where

$$\text{projcovers}(C, v, \mathcal{D}) = \{\{v/t\} | \{v/t\} \text{ occurs in } \theta \in \text{covers}(C, \mathcal{D})\}\}$$

Thus, $\text{projcovers}(C, v, \mathcal{D})$ projects the substitutions in $\text{covers}(C, \mathcal{D})$ on individual variables, and the frequency $\text{projfreq}(C, \mathcal{D})$ is the size of the smallest projected set. Bringmann and Nijssen (2008) show that projected frequency can be computed more efficiently than some of the alternatives that had been proposed in the literature (Kuramochi & Karypis, 2004; Fiedler & Borgelt, 2007). Furthermore, it is anti-monotonic as adding further conditions to a query C can only decrease number of projected substitutions $\{v/t\}$ for which the query succeeds. When using $\text{projfreq}(C, \mathcal{D})$, either $\text{covers}(C, \mathcal{D})$ or $\text{projcovers}(C, v, \mathcal{D})$ can be considered to be the underlying semantics which lead to the notion of frequency. As we show later, this also affects the notion of implication and closure, and for LCM we must use $\text{covers}(C, \mathcal{D})$ as the underlying semantics.

Remark 3 Note that if we have data from the LI setting we can modify it slightly and interpret it in the SI setting. This is a standard transformation in inductive logic programming. For example, consider a dataset $\mathcal{D} = \{I_1, I_2\}$ with two interpretations $I_1 = [p(a), q(a,b), q(a,c)]$ and $I_2 = [p(d), q(e,f)]$. We add an identifier argument to each predicate and collapse the data into one interpretation: $\mathcal{D}' = [p(id_1, a),\ q(id_1, a, b),\ q(id_1, a, c),\ p(id_2, d),\ q(id_2, e, f)]$. Now we have a choice and can use either totfreq$(C, \mathcal{D})$, projfreq$(C, \mathcal{D}')$ or relfreq$(C, \mathcal{D}')$ for this data. Notice that this representation allows queries to mix atoms from different interpretations potentially giving unintuitive results. For example, a query such as $[p(x, a), q(y, e, f)]$ matches $\mathcal{D}'$, but it does not provide interesting information. This can be avoided by requiring that each query use exactly one interpretation identifier as discussed in the next remark.

Remark 4 One can further generalize the setup for data and queries so that both LI and SI settings are captured simultaneously. In particular, the encoding of the previous remark suggests that we treat the data always in the SI setting, but identify a special *key* predicate separating the data into the corresponding interpretations. In this way a query $[p(x), q(x, y)]$ for $\mathcal{D}$ can be translated to a query $[key(k), p(k, x), q(k, x, y)]$ for $\mathcal{D}'$ yielding exactly the same results. In general, given data in SI format we can designate a key predicate (even allowing more than one argument) identifying objects of interest, and restrict the queries to have exactly one instance of the key predicate. The use of a key predicate corresponds to always projecting substitution on specific variables in the queries. We can use a notion of "key frequency" to capture behavior similar to the LI setting and "full frequency" if keys are not used in the queries. While some of the results in the paper transfer to this more general representation, our proofs for the algorithm and constructions for the LI setting in Section 3 do not generalize, and we therefore keep the separate notations for LI and SI settings. A generalization along these lines can be both useful and insightful but requires further research.

2.3 Closures

Finding frequent item-sets is a special case of frequent pattern mining in the LI case. When mining frequent item-sets many patterns are redundant, and various research efforts have tried to identify compact representations capturing all frequent sets. The frequent sets that are maximal in terms of item inclusion provide such a representation but their enumeration can be computationally demanding. Therefore, previous work has attempted to enumerate all closed sets, that include all maximal sets and can still provide time and space savings relative to all frequent sets. Consider again the dataset T given in Figure 1. Then, freq$([7], \mathcal{D})$ = freq$([7, 9], \mathcal{D}) = 4$. The item 7 is said to imply item 9 w.r.t. the database. We similarly define this for relational atoms.

Definition 5 *A conjunction C implies an atom p in the database $\mathcal{D}$, denoted $\mathcal{D} \models C \rightarrow p$, if and only if* covers$(C, \mathcal{D})$ = covers$([C, p], \mathcal{D})$.

In other words, a conjunction C implies an atom p if whenever C holds in any of the interpretations in $D \in \mathcal{D}$ then $[C, p]$ holds in D as well. Thus, the rules of the form $C \rightarrow [C, p]$ denote relational association rules that hold with 100 percent confidence. They are to be understood as implications that are satisfied by all conjunctions in the database. These expressions correspond to the relational association rules, also called query extensions, introduced in Warmr (Dehaspe & Toivonen, 2000). In case the database consists of a single interpretation only, it does not make

sense to allow for atoms p that contain variables that do not appear in C. The reason is that the resulting substitutions are not comparable, cf. also Example 2. Therefore, in this case one imposes the *range-restriction* requirement (De Raedt & Ramon, 2004), which states that p only contains variables and constants appearing in C. The resulting expression can then be simplified to a (range-restricted) clause $C \rightarrow p$ and corresponds to the pattern type induced by Claudien (De Raedt & Dehaspe, 1997).

In the item-set case the closure of a set of items X is defined as the set of items whose existence in a tuple is implied by the existence of X. Continuing our illustration for the dataset T in Figure 1, one can verify that the closure of $\{7\}$ w.r.t. the specified tuples is $\{7, 9\}$. An alternative characterization of closed item-sets states that the closure of an item-set I corresponds to the intersection of all tuples in the database subsumed by the item-set X (Bastide et al., 2000). This can be easily verified for the set $\{7\}$ in our example. Note that intersecting two item-sets corresponds to computing the minimally general generalization of two patterns.

Thus, we have two potential approaches to define closures: based on implication or on least generalization (the intersection operation for conjunctions of atoms). In Section 3 we use the least generalization approach for the LI setting. However, it is simpler to start our discussion with the implication approach captured by the iterative closure procedure that we now introduce. Before doings so, we must define the notion of a *refinement operator* ρ which computes atoms p to be added to the conjunction. More formally, for the set of ordered conjunctions and the relation $\preceq$, the operator ρ is called a downward refinement operator if for any conjunction C we have $\rho(C) \subseteq \{C' | C \preceq C'\}$. Although in general we can allow $\rho(C)$ to output arbitrary conjunctions C', here we focus on the case where $C' = C \cup \{p\}$ for some atom p and thus $C \preceq C'$ holds automatically. We therefore also use $\rho(C)$ to refer to the set of atoms p that are added to C in this way.

ITERATIVE CLOSURE PROCEDURE (ICP)
closure(Input: pattern $C = q_1, \ldots, q_n$, and a refinement operator ρ)

1 $C' \leftarrow C$
2 **repeat**
3 Find an atom $p \in \rho(C')$ s.t. $\mathcal{D} \models C' \rightarrow p$
4 $C' \leftarrow [C', p]$
5 **until** no such atom is found
6 Output C'

The procedure can be defined for all the four semantics given above by interpreting $\mathcal{D} \models C' \rightarrow p$ accordingly. We first consider a general refinement operator ρ that imposes no restrictions on p other than that $p \notin C'$. Some of our theorems below using the normal form require the following syntactic version for the input and output of the refinement operator. We will assume that C' uses variables $x_1, \ldots, x_m$ for some m and that new variables introduced by p form a contiguous sequence starting with x_{m+1}. This does not change the semantics and does not restrict the form of refinements so we still refer to this case as the general ρ. Indeed, until Section 4 we assume that ICP uses a general ρ.

The following notion is natural.

Definition 6 (Closed conjunctions of atoms) A conjunction C is closed if closure$(C) = C$.

The ICP algorithm defines closure in a non-deterministic way that may depend on the order of atom additions. Depending on the semantics, the properties of ICP may vary. In particular, the

result may or may not be unique, or the algorithm may not always terminate, in which case the result would not be well defined.

Example 7 Consider using θ-subsumption and the dataset $[p(1,2), p(2,1)]$. Then, the pattern $p(x_1, x_2)$ implies $[p(x_1, x_2),\ p(x_2, x_3)]$ as well as $[p(x_1, x_2),\ p(x_2, x_3),\ p(x_3, x_4)]$ and so on. In this way we can add a chain of any size, implying that the closure is not finite in size, and the procedure may not terminate.

Therefore, under θ-subsumption, it is necessary to restrict the atoms that can be added to the initial conjunction. Our solutions get around this problem in different ways for the LI and SI settings. For LI, Section 3 gives a solution that avoids iterative closure and builds on the notion of least generalization to capture implied atoms. For SI, Section 4 uses range-restricted implication to avoid infinite chains. Before presenting these, we briefly consider in the following example, the LI-OI setting used in CloseGraph (Yan & Han, 2003) and Farmer (Nijssen & Kok, 2003).

Example 8 Consider data $\mathcal{D} = \{I_1, I_2\}$ with two interpretations captured by the following two conjunctions and used in the LI-OI setting:

$$I_1 = [e(1,2,a), e(2,3,b), e(4,5,a), e(5,6,c)],$$
$$I_2 = [e(11,12,a), e(12,13,b), e(12,14,c)]$$

This database represents two edge-labeled graphs, where the first two arguments of e are nodes and the third argument is an edge label. We consider calculating the closure of $C = [e(x_1, x_2, a)]$ with OI-subsumption. Then, the ICP algorithm may add $e(x_2, x_3, b)$ to get the closed set $[e(x_1, x_2, a),\ e(x_2, x_3, b)]$. On the other hand it can add $e(x_2, x_3, c)$ to get the closed set $[e(x_1, x_2, a),\ e(x_2, x_3, c)]$.

We therefore have the following result.

Proposition 9 *The closure of a query in the LI-OI setting is not unique.*

In other words, in the graph mining setting of CloseGraph (Yan & Han, 2003) and Farmer (Nijssen & Kok, 2003) closed sets are well defined, but the notion of closure is not unique. This fact was already pointed out in previous work (Kuznetsov & Samokhin, 2005; Kuznetsov, 2004). In that case, algorithms like LCM, which rely on calculating the unique closure of a pattern, cannot be directly used.

2.4 Normal Form

Most graph and relational frequent pattern mining algorithms employ a normal-form based on an ordering of patterns to avoid investigating the same pattern more than once (Yan & Han, 2002; Yan & Han, 2003; Nijssen & Kok, 2003). We use the following order which is similar to the one introduced by Nijssen and Kok (2003).

We assume that the predicates and constants are ordered (e.g. alphabetically). In addition, we employ a special set of variables $z_1, z_2, z_3, \ldots$, ordered according to their index, and impose that all constants are smaller than these variables. An order over atoms is then induced by the order over lists composed by the predicate name as first element and its list of arguments following that. An order over conjunctions is induced by the order over lists, where the atoms in the conjunction are listed in order.

Definition 10 (Normal form) *The normal form* $\mathrm{nf}(C)$ *of a conjunction* C *over variables* $x_1, \ldots, x_u$ *is* $C\theta$ *where* θ *is (1) a one to one substitution from* $x_1, \ldots, x_u$ *to new variables* $z_1, \ldots, z_u$ *and (2)* $C\theta$ *is minimal in the order of conjunctions for substitutions of type (1).*

Example 11 Consider the conjunction $[p(x_2), q(x_1, x_2), q(x_2, x_3)]$ and assume that $p < q$. Then, there are 6 substitutions to consider and since $p < q$, the least one must map x_2/z_1. The substitution $\theta = \{x_1/z_3,\ x_2/z_1,\ x_3/z_2\}$ gives $C\theta = [p(z_1),\ q(z_3, z_1),\ q(z_1, z_2)]$ and when sorted, we get $C\theta = [p(z_1),\ q(z_1, z_2),\ q(z_3, z_1)]$. One can check that this is the normal form.

Remark 12 Note that in the special case when the database is a graph the normal form gives a way to check graph isomorphism so we cannot expect an efficient algorithm for the worst case. Our algorithm will need to calculate normal forms for conjunctions. As the example above illustrated, the order on predicates can be used in this process. The following example shows that when passing substitutions iteratively across the sorted predicates, one must be careful to pass multiple substitutions at the different steps. Consider the conjunction $[p_1(x_1, x_2), p_1(x_2, x_1), p_2(x_3, x_1), p_3(x_3, x_2), p_4(x_1)]$ which is in normal form. In calculating the minimal substitution for the first predicate p_1 we cannot distinguish the form above from $[p_1(x_1, x_2), p_1(x_2, x_1), p_2(x_3, x_2), p_3(x_3, x_1), p_4(x_2)]$. If we carry the wrong substitution to the next step we will not get the correct normal form. We must therefore carry all substitutions realizing the minimum for the predicate and consider them for the next predicate.

Normal forms satisfy the following important properties.

Lemma 13 *Let* $C = [q_1, q_2, \ldots, q_n]$ *be a conjunction in normal form. (i) For any* $i \leq n$, $[q_1, \ldots, q_i]$ *is in normal form. (ii) For any* $i < n$ *and any subset of indices* $i < j_1 < j_2 < \ldots < j_k \leq n$, $[q_1, \ldots, q_i]$ *is a prefix of the normal form of* $[q_1, \ldots, q_i, q_{j_1}, \ldots, q_{j_k}]$.

Proof. Since (ii) implies (i), it suffices to show that (ii) holds. Let $L = q_1, \ldots, q_i$ and $T = q_{j_1}, \ldots, q_{j_k}$. Assume that property (ii) does not hold: since L is not a prefix of the normal form, this implies that there is a substitution θ such that $L' = \mathrm{sort}([L, T]\theta) < [L, T]$. In particular, there is a least index l such that $L'[l] < L[l]$.

Notice that it is possible that $[L, T]$ and hence, also θ do not refer to all the variables in C. Assume that C uses variables $x_1, \ldots, x_v$ and that $[L, T]$ uses only $u < v$ variables. Then, the domain of θ is clearly $x_1, \ldots, x_u$. Extend θ by mapping the remaining variables arbitrarily onto $z_{u+1}, \ldots, z_v$ to get a substitution θ'.

We claim that $C' = \mathrm{sort}(C\theta') < C$ contradicting the fact that C is in normal form. Notice that L' is a subset of C' and consider how the atoms of L' are arranged in C'. If $L'[l]$ is a prefix of C' then $C' < C$ since $C'[l] < C[l]$. Otherwise there is an atom $p \in C$ and $p \notin [L, T]$ s.t. $p' = p\theta'$ appears in position $w < l$ in C'. But this again implies that $C' < C$ since $C'[w] < C[w]$. ■

3 LI-θ: the lgg Closure

One way to define closures in the item-set case is by intersection of the covered tuples. The equivalent approach in the LI-θ setting is to apply the well-known least general generalization (lgg) operator introduced by Plotkin (1970). The lgg of two conjunctions (sets of atoms) C_1, C_2 is a conjunction of atoms C that θ-subsumes both C_1 and C_2 i.e. $C \preceq C_1$, and $C \preceq C_2$; in addition, for

any other conjunction C' that θ-subsumes both C_1, C_2 it holds that $C' \preceq C$. In other words, C is the least general generalization of C_1 and C_2 with respect to $\preceq$.

It is well-known that syntactically different clauses can be equivalent under theta-subsumption. However, Plotkin (1970) proved that theta-subsumption induces a complete lattice at the level of equivalence classes and provided an algorithm to calculate the *lgg* of two clauses (which corresponds to the least upper bound of the classes of the two clauses in that lattice). The algorithm works for expressions with complex term structure, but here we only use it for terms which are constants or variables. The algorithm is as follows: the *lgg* of two identical constants is the same constant; in any other case, the lgg of two terms is a new variable x, where x stands for the *lgg* of that pair of terms throughout the expression. This information is kept in what we call the *lgg* table. The *lgg* of two *compatible* atoms $p_i(t_1, ..., t_n)$ and $p_i(t'_1, ..., t'_n)$ is the atom $p_i(lgg(t_1, t'_1), ..., lgg(t_n, t'_n))$. The *lgg* is only defined for compatible atoms, that is, atoms with the same predicate symbol and arity. The *lgg* of two conjunctions (sets of atoms) C_1 and C_2 returns another conjunction of atoms whose set representation is:

$$\left\{ lgg(p, p') \;\middle|\; (p, p') \text{ are two compatible atoms of } C_1 \text{ and } C_2 \text{ respectively} \right\}.$$

It is important to note that all atom *lgg*s in this computation share the same table.

Example 14 Let $C_1 = [p_1(1,1), p_2(1,2,2)]$, $C_2 = [p_1(3,3), p_2(3,4,4)]$, and $C_3 = [p_1(5,5),\ p_1(5,6),\ p_2(5,6,7)]$. Then, $C = lgg(C_1, C_2) = [p_1(x_1, x_1), p_2(x_1, x_2, x_2)]$. The *lgg* table produced during the computation of $lgg(C_1, C_2)$ is:

`[1 - 3 ⇒ ` x_1`]`	(from $p_1(\underline{1}, 1)$ with $p_1(\underline{3}, 3)$)
`[2 - 4 ⇒ ` x_2`]`	(from $p_2(1, \underline{2}, 2)$ with $p_2(3, \underline{4}, 4)$)

Next, calculating $C' = lgg(C, C_3)$ gives $C' = [p_1(z_1, z_1), p_1(z_1, z_2), p_2(z_1, z_3, z_4)]$ with the *lgg* table:

`[`x_1 `- 5 ⇒ ` z_1`]`	(from $p_1(\underline{x_1}, x_1)$ with $p_1(\underline{5}, 5)$)
`[`x_1 `- 6 ⇒ ` z_2`]`	(from $p_1(x_1, \underline{x_1})$ with $p_1(5, \underline{6})$)
`[`x_2 `- 6 ⇒ ` z_3`]`	(from $p_2(x_1, \underline{x_2}, x_2)$ with $p_2(5, \underline{6}, 7)$)
`[`x_2 `- 7 ⇒ ` z_4`]`	(from $p_2(x_1, x_2, \underline{x_2})$ with $p_2(5, 6, \underline{7})$)

Note that in the algorithm above we do not perform a logical reduction of the *lgg*.[2] It will be useful later to refer to this syntactic and non-reduced form of the *lgg*. The example illustrates that for sets larger than two we simply calculate the *lgg* by iteratively adding one clause at a time. The result is unique up to variable renaming regardless of the order of adding the conjunctions. However, since the size of the output may grow as the product of the sizes of the input clauses, when we calculate the *lgg* of many clauses the size may grow exponentially. We can now define a notion of closure based on the *lgg* properties: the closure of a pattern is the *lgg* of the interpretations it covers.

Definition 15 (*lgg* closure) Let C be a conjunction of atoms and $\mathcal{D}$ a database in the LI setting, then $\text{closure}_{LGG}(C) = lgg(\text{covers}_{LI}(C, \mathcal{D}))$.

[2] Under θ-subsumption it is possible that two syntactically different sets of atoms are equivalent, e.g., $[p(x, y)]$ and $[p(x, y), p(x, z)]$. In inductive logic programming, it is common to work with the smallest sets as a representative for the equivalence class; this is the so-called reduced form. In the present work, we do not apply such reductions.

We next show that this notion of closure can be used to yield a generalization of LCM . A generalization of LCM using the idea of lgg is reported by Arimura and Uno (2005). Their setting however is specialized to ordered tree patterns with a matching relation that corresponds to OI subsumption. Interestingly in this setting the size of the lgg is always small. Also, in the context of LI-OI, that is, the graph mining setting, one might want to consider the maximum common subgraph instead of the lgg under θ-subsumption (Schietgat et al., 2011; Kuznetsov & Samokhin, 2005). Unfortunately, the result of this operation is not necessarily unique.

Formal Concept Analysis (Ganter & Wille, 1998) has already been used to analyze rich settings for data mining, such as closed sets of sequences (Balcázar & Garriga, 2007). We next show that when using lgg closures, formal concepts arise yielding a Galois connection. This connection is useful for presentation and to formally state that lgg-based closures will satisfy the most general properties of any closure system.

We consider the formal context $(2^{\mathcal{D}}, \mathcal{L})$, where $\mathcal{L}$ is the set of possible conjunctions (the language of patterns) and $2^{\mathcal{D}}$ the set of possible sets of interpretations. We use the partial orders given by $\subseteq$ on $2^{\mathcal{D}}$ and by $\preceq$ on $\mathcal{L}$. From a formal context one typically defines two mappings that form a Galois connection. The mapping ψ maps descriptions to interpretations and the mapping ϕ maps interpretations to descriptions. In particular, for $C \in \mathcal{L}$, we define $\psi(C) = \text{covers}_{LI}(C, \mathcal{D})$ and for a set of interpretations $S \subseteq \mathcal{D}$ we define $\phi(S) = lgg(S)$. Then, we have the following result.

Theorem 16 *The mappings ϕ and ψ form a Galois connection. More specifically, for $S, S' \subseteq \mathcal{D}$ and $C, C' \in \mathcal{L}$ we have:*
(1) $S \subseteq S' \Rightarrow \phi(S') \preceq \phi(S)$;
(2) $S \subseteq \psi(\phi(S))$;
(3) $C \preceq C' \Rightarrow \psi(C') \subseteq \psi(C)$;
(4) $C \preceq \phi(\psi(C))$.

Proof. (1) By definition we have that $\phi(S') \preceq s'$ for all $s' \in S'$; in particular, since $S \subseteq S'$, then $\phi(S') \preceq s$ for all $s \in S$. Then, since $\phi(S) = lgg(S)$ the definition of lgg implies $\phi(S') \preceq \phi(S)$.
(2) Let $C = \phi(S) = lgg(S)$. Then $C \preceq s$ for all $s \in S$, so that $s \in \psi(\phi(S))$.
(3) By definition, for all $s' \in \psi(C')$ we have that $C' \preceq s'$; and the hypothesis is that $C \preceq C'$, so by transitivity $C \preceq s'$ and $s' \in \psi(C)$. Thus, $\psi(C') \subseteq \psi(C)$.
(4) By definition we have that for all $s \in \psi(C)$, $C \preceq s$. This implies that $lgg(\psi(C))$ returns a conjunction C' such that $C \preceq C'$. Thus, we have that $C \preceq \phi(\psi(C))$. ■

The Galois connection gives us two closure systems that are dually isomorphic to each other.

Theorem 17 *The compositions $\widehat{\Delta} = \psi \cdot \phi$ and $\Delta = \phi \cdot \psi$ are closure operators. That is, they satisfy monotonicity, extensivity, and idempotency.*

Proof. We give the proof for Δ; the proof for $\widehat{\Delta}$ is similar. We need to show the following properties: monotonicity: $C \preceq C' \Rightarrow \Delta(C) \preceq \Delta(C')$; extensivity: $C \preceq \Delta(C)$; idempotency: $\Delta(\Delta(C)) = \Delta(C)$. These are easily established using Theorem 16.
Monotonicity: By (3) $C \preceq C'$ implies $\psi(C') \subseteq \psi(C)$, and by (1) we get $\phi(\psi(C)) \preceq \phi(\psi(C'))$.
Extensivity: Follows directly from (4).
Idempotency: By substituting $\phi(\psi(C))$ for C in (4) we obtain $\phi(\psi(C)) \preceq \phi(\psi(\phi(\psi(C))))$, thus $\Delta(C) \preceq \Delta(\Delta(C))$. On the other hand, with $S = \psi(C)$ we obtain by condition (2) that $\psi(C) \subseteq \psi(\phi(\psi(C)))$, and condition (1) yields $\phi(\psi(\phi(\psi(C)))) \preceq \phi(\psi(C))$, thus $\Delta(\Delta(C)) \preceq \Delta(C)$. ■

Using this result, *closed conjunctions* can be formalized as those coinciding with their closure, that is, $\Delta(C) = C$. The result also establishes that for any set of interpretations S of the database, there is a unique conjunction (up to subsumption equivalence) that is the most specific conjunction satisfied in S. Moreover, the number of closed queries is finite because the number of interpretations in our database $\mathcal{D}$ is finite.

We now develop an algorithm RelLCM1 that upgrades the first algorithm of Uno et al. (2004) to the LI-θ case. We need a notion of closure extension that allows us to go directly from one closed conjunction to another.

Definition 18 (Closure extension) A conjunction C' is a closure extension of a conjunction C if $C' = \text{closure}_{LGG}([C, p])$ for $p \in \rho(C)$.

RelLCM1 stores all previously discovered closed sets in a table and in this way avoids calling the procedure twice on the same set. It uses depth first search. Each closed conjunction is refined in all possible ways and closed. The algorithm checks whether the resulting closed set was already discovered, and if not it is output and further refined to identify more closed conjunctions. The algorithm is started by calling RelLCM1($\text{closure}_{LGG}(\emptyset)$), where $\text{closure}_{LGG}(\emptyset)$ represents the closed pattern satisfied by all the interpretations in $\mathcal{D}$.

RelLCM1(input: closed pattern C)

```
1  if C is not frequent then return
2  if C is already in closed pattern table then return
3  store C in closed pattern table
4  output C
5  for all refinements p ∈ ρ(C)
6          if covers_LI([C,p]) = ∅
7            or covers_LI([C,p]) = covers_LI(C)
8             then skip refinement
9             else  Calculate Ĉ = nf(closure_LGG([C,p]))
10                  RelLCM1(Ĉ)
11 return
```

Clearly, every conjunction output by the algorithm is closed. The following theorem guarantees completeness, that is, that every closed conjunction is output by the algorithm.

Theorem 19 *If $C \neq \text{closure}_{LGG}(\emptyset)$ is a closed conjunction in normal form, then there is a closed conjunction C' and a literal $p \in \rho(C')$ such that $C = \text{nf}(\text{closure}_{LGG}([C', p]))$.*

Proof. Since $C \neq \text{closure}_{LGG}(\emptyset)$, it violates at least one of the interpretations in $\mathcal{D}$. Let $S \subset \mathcal{D}$ be the set of interpretations covered by C, i.e. $S = \psi(C)$. Remove atoms from C until its covered set changes (at least one more interpretation is covered) and then add back atoms that were removed as long adding them does not change the covered set. As a result we get a maximal set $B \subset C$ so that $\psi(B) = S' \supset S$. Let p be an atom in $C \setminus B$ so that $\psi([B, p]) = S$. So, B almost satisfies the requirements of the theorem. The only problem is that B may not be closed, as the following example illustrates.

Example 20 It is instructive to look at the details for the interpretations given in Example 14. Recall that $C_1 = [p_1(1,1), p_2(1,2,2)]$, $C_2 = [p_1(3,3),\ p_2(3,4,4)]$, and $C_3 = [p_1(5,5),\ p_1(5,6),\ p_2(5,6,7)]$. Then, $C = lgg(C_1, C_2) = [p_1(x_1, x_1),\ p_2(x_1, x_2, x_2)]$ and $C' = lgg(C_1, C_2, C_3) = [p_1(z_1, z_1),\ p_1(z_1, z_2),\ p_2(z_1, z_3, z_4)]$. Now if we remove the $p_2()$ atom from C we get a maximal subset $B = [p_1(x_1, x_1)]$, where the literal p from the previous paragraph corresponds to $p_2(x_1, x_2, x_2)$. The clause B covers C_1, C_2, C_3 and it is not closed (the closure is C').

However, we can see that a variable renaming of B appears as a subset of C'. As the next argument in the proof shows this is not a coincidence. For the argument it is worth recalling the lgg table constructed when calculating $C' = lgg(C, C_3)$ repeated here:

$[x_1$ - 5 $\Rightarrow$ $z_1]$	(from $p_1(\underline{x_1}, x_1)$ with $p_1(\underline{5}, 5)$)
$[x_1$ - 6 $\Rightarrow$ $z_2]$	(from $p_1(x_1, \underline{x_1})$ with $p_1(5, \underline{6})$)
$[x_2$ - 6 $\Rightarrow$ $z_3]$	(from $p_2(x_1, \underline{x_2}, x_2)$ with $p_2(5, \underline{6}, 7)$)
$[x_2$ - 7 $\Rightarrow$ $z_4]$	(from $p_2(x_1, x_2, \underline{x_2})$ with $p_2(5, 6, \underline{7})$)

and the associated mappings θ_1=$\{x_1/z_1\}$ showing that B is embedded in C' and θ_2=$\{z_1/x_1,\ z_2/x_1,\ z_3/x_2,\ z_4/x_2\}$ mapping C' back to C.

So far, we have the following conditions: (1) $B \subset C$, (2) for every interpretation $s \in \hat{S}$ with $\hat{S} = S' \setminus S$, we have $B \preceq s$, (3) $C' = lgg(C, lgg(\hat{S}))$. But since B subsumes s for $s \in \hat{S}$, every atom in B can be matched to an atom in s so that a variable in B is always matched with the same object in s. Therefore, these mappings can be reconstructed from the lgg table, and each atom of B has a variable renaming copy in C'. Let θ_1 be the portion of the lgg table capturing this one to one mapping and reversed so as to map C to C'.

We claim that $[C', p\theta_1]$ is the refinement required in the theorem. In the example, $[C', p\theta_1] = [p_1(z_1, z_1),\ p_1(z_1, z_2),\ p_2(z_1, z_3, z_4),\ p_2(z_1, x_2, x_2)]$. Notice how it mixes variables from C' with variables from C. We need to show that $\psi([C', p\theta_1]) = S$.

Consider any interpretation s in $\mathcal{D}$ such that $[C', p\theta_1] \preceq s$, that is, there is a θ such that $[C', p\theta_1]\theta \subseteq s$. Then, since $B\theta_1 \subset C'$ we have $[[B, p]\theta_1]\theta \subseteq s$. But then since $[B, p]$ covers exactly S we have that $s \in S$.

For the other direction, let θ_2 be the substitution mapping C' back to C that can be obtained from the lgg table. Then, we have $[C', p\theta_1]\theta_2 \subseteq C$. The inclusion for the part of C' holds by the definition of the lgg. We also have that $[p\theta_1]\theta_2 \in C$. To see this, consider the variables in p and partition them according to whether they appear in B or not. For variables that appear in B the mapping $\theta_1\theta_2$ is the identity mapping. For variables not in B, since the calculation of lgg renames all variables, $\theta_1\theta_2$ does not replace the variable and again we get the identity mapping. So $p\theta_1\theta_2 = p \in C$ as required. Now for all $s \in S$ we have $[C', p\theta_1] \preceq C \preceq s$ as required. ■

Clearly, in any run of the algorithm the number of calls to RelLCM1 is exactly the number of closed sets. While the number of calls is linear, we do discover some patterns more than once. Thus, we need to store patterns in some table or cache in order to avoid listing them more than once in the output. In addition, this step has to account for the efficiency of looking up a new query in the table every time a closed pattern is discovered. The maximum number of patterns discovered is bounded by the branching factor (treated as a constant) of the refinement operator times the number of closed sets.

The efficiency of the algorithm in this section relies directly on the ability to calculate the lgg of sets of interpretations. In some cases where interpretations have a simple structure, one can

calculate a compact representation of the *lgg* (Horváth & Turán, 2001) and this has been applied in the context of learning for natural languages (Horváth et al., 1999). However, in the general case, the size of the *lgg* grows exponentially with the number of interpretations. In this case, one can try to limit the *lgg* size. One idea, related to the computation in Jimi (Maloberti & Suzuki, 2003), is to calculate some variant of the least generalization iteratively by pruning atoms that lead to duplicate instances in the data. We discuss this further below in the context of LCM for the SI setting.

Remark 21 In the graph mining context of LI-OI one can consider the maximum common subgraph instead of *lgg* under θ-subsumption. Unfortunately, the result of such operation is not unique (see also Proposition 9). Still, as previously suggested (Kuznetsov & Samokhin, 2005; Kuznetsov, 2004), it is possible to define a closure operator by setting $\phi(S)$ to be the set of all common subgraphs in the set of graphs S. Notice that this new definition of ϕ distinguishes between a closed graph G and the closed set $\{G\}$. Closed sets of graphs form a lattice of partially ordered sets by introducing a maximal top element. However, closed graphs do not have this property. Even if a closure operator can be defined for closed sets of graphs in the LI-OI setting, the drawback is that algorithms such as LCM, that rely on the computation of a unique closure, cannot be directly used. A first issue already discussed by Kuznetsov (1999), is that computing the closure as the set of all common subgraphs is NP-hard; this can be approximated by using projections on graphs, but then there is no guarantee that all the closed patterns would be enumerated by the LCM strategy. A second issue for this approach is defining proper refinements for sets of closed patterns, as required by Definition 18.

4 SI: Range Restricted Closures

We now consider the SI setting (both under θ- and OI subsumption). This scenario corresponds to the network mining setting described in the introduction: like in the Cora database, we have several relations over a set of objects, that is a large network of relations and the frequent patterns can be used to identify trends in the data. This setting has different applications, for example in social-network mining or web-graph mining (Bringmann & Nijssen, 2008; Kuramochi & Karypis, 2004; Fiedler & Borgelt, 2007).

As argued in Section 2.3 we need to impose range-restriction when defining the closures. Thus, the ICP algorithm employs the operator $\rho_{RR}(C)$ that can only generate atoms containing terms already occurring in C (De Raedt & Ramon, 2004).

Definition 22 (Range restricted closure) *Let C be a conjunction of atoms, then we define* $\text{closure}_{RR}(C) = \text{closure}(C, \rho_{RR})$*, where* $\text{closure}(C, \rho_{RR})$ *applies the* ICP *algorithm with a range-restricted refinement operator.*

The next lemma shows that $\text{closure}_{RR}(C)$ is well behaved.

Lemma 23 *For any conjunction C and atoms p, q, where p includes only terms that appear in C, if $\mathcal{D} \models C \rightarrow p$ then $\mathcal{D} \models [C, q] \rightarrow p$.*

Proof. Notice that for the SI setting, the statement $\mathcal{D} \models C \rightarrow p$ means that the atom p is true in $\mathcal{D}$ in all substitutions that make C true. The addition of q can remove some of the substitutions

to variables in C (or make copies of them) but cannot modify the substitutions to these variables in any other way. As a result p is still satisfied in all the remaining substitutions and therefore $\mathcal{D} \models [C, q] \rightarrow p$. ■

Remark 24 It is instructive to see that the property in the previous lemma is violated in the LI setting, where the range restriction requirement is not used. Consider a database $\mathcal{D}$ with one interpretation: $I_1 = [p_1(1,2), p_1(5,6), p_2(2,3), p_3(7,6)]$. Now consider the conjunction $C = [p_1(x,y)]$ and atoms $p = p_2(y,z)$ and $q = p_3(w,y)$. Then, we have that $\mathcal{D} \models C \rightarrow p$, and $\mathcal{D} \models C \rightarrow q$ but $\mathcal{D} \not\models [C, q] \rightarrow p$. Essentially, each of p and q chooses one of the substitutions of the predicate $p_1()$ in I_1, but they cannot be combined.

Remark 25 The same property fails for the projcovers() semantics discussed in Section 2. To see this, consider the database with one interpretation $\mathcal{D} = [r(1,2), r(1,3), r(2,3), p(2,1), p(3,2), q(3)]$ and let $C = [r(x,y)]$, $p = p(y,x)$ and $q = q(y)$. Now, if we use projcovers() as the semantics, then for implication we only need to verify that the projected substitutions are the same. Therefore, we have $\mathcal{D} \models C \rightarrow p$, where the projected substitutions are $x/\{1,2\}$ and $y/\{2,3\}$. However, $\mathcal{D} \not\models [C, q] \rightarrow p$ because for $[C, q]$ the projected substitutions are $x/\{1,2\}$ and $y/\{3\}$ whereas for $[C, q, p]$ they are $x/\{2\}$ and $y/\{3\}$. Therefore, the algorithm given below can use projfreq() as frequency with covers() to define closure, but it cannot use projcovers() to define closures. This affects the potential speedup obtained by the projection.

To get a complete generalization of LCM we need to ensure that each conjunction has a unique parent. Then, the algorithm does not need to store the enumerated conjunctions in order to avoid duplicates as in RelLCM1. Therefore, we need a more refined notion of closure extension. The following generalizes the corresponding definitions of Uno et al. (2004). The basic idea is that only those closed sets that are prefix preserving can be used as extensions of other closed sets.

Definition 26 (Core prefix) Let C be a conjunction in normal form. The core prefix core(C) of C is the least prefix pr of C such that covers($pr, \mathcal{D}$) = covers($C, \mathcal{D}$).

Notice that since we are working in the SI setting, if the covers are the same then the prefix pr and C must have the same variables.

Definition 27 (Prefix preserving closure extension (PPC-extension)) *Let $C = [q_1, \ldots, q_n]$ be a closed conjunction in normal form. A conjunction C' is a* PPC-extension *of C if it satisfies:*

ppc1: $C' = \text{nf}(\text{closure}_{RR}([C, p]))$ with $p \in \rho(C)$, i.e. C' is obtained by adding the atom p to C, taking the closure of $[C, p]$ and normalizing.

ppc2: For all $q \in \text{core}(C)$, the atom p satisfies $p > q$, that is p goes after q in the order established by the normal form. Note that because C is in normal form this only requires a check that p is larger than the last atom q in core(C).

ppc3: Let $C[j] = [q_1, \ldots, q_j]$ be the prefix of C up to q_j, where q_j is the largest atom in C s.t. $q_j < p$. Then, $[C[j], p]$ is a prefix of C'.

Notice that the property ppc3 implies that the core prefix is preserved and that no new atom can appear between p and q_j. In fact, the entire portion $[C[j], p]$ cannot be renamed or modified by the normal form. It is convenient to think about condition ppc3 in terms of atoms added by iterative closure, but *lgg* closures are also possible. As we show below, however, various semantic problems prevent the use of *lgg* with prefix preserving closures.

We can now revisit the example of Figure 1 explaining the details of PPC extensions. The core prefix for the closed sets in the figure are underlined in the nodes.[3] As shown by Lemma 31 below, the largest index in this set is the item used in refining the set from its parent. The following cases illustrate the possible scenarios.

- Starting with the empty set, and refining by adding item 1, we get the closed set $\{1, 7, 9\}$. This is a PPC-extension with the core prefix $\{1\}$.
- Refining the closed set $\{7, 9\}$ with 1 we get the closed set $\{1, 7, 9\}$. This is not a PPC-extension since condition ppc2 is violated. In fact, the LCM algorithm avoids this refinement altogether since ppc2 can be tested in advance.
- Refining the empty set with 8 leads to the closed set $\{1, 2, 7, 8, 9\}$, but this is not a PPC-extension since the items $1, 2, 7$ are smaller than 8 and condition ppc3 is violated. In this case, ppc3 cannot be tested in advance. Therefore, LCM will try the refinement and discover $\{1, 2, 7, 8, 9\}$, but will abandon this path since it is not a PPC-extension.

Given these definitions the LCM algorithm recursively refines and closes the resulting sets producing an output whenever a PPC-extension is found. The next example illustrates various cases of closure extensions and prefix preserving extensions for multi-relational data.

Example 28 Consider a database of a single interpretation with the relational atoms:

$$\begin{aligned}\mathcal{D} = \quad & [\ p_0(1), p_0(2), p_0(3), p_0(4), p_0(5), \\ & p_1(a,a), p_1(a,b), p_1(b,c), p_1(a,h), p_1(c,a), p_1(c,b), \\ & p_2(1,a), p_2(1,b), p_2(2,c), p_2(3,h), p_2(4,c), p_2(5,a), \\ & p_3(1,a), p_3(2,b), p_3(3,a), p_3(4,b), p_3(5,c), \\ & p_4(a), p_4(b), p_4(c)\ ]\end{aligned}$$

Intuitively, we have two types of objects, where p_0 captures numeric ones and p_1 and p_4 describe the alphabetical objects. Predicates p_2 and p_3 relate the two types. The following observations are useful in analyzing this data. Both of p_1's first argument and p_3's second argument only hold for a, b, c so they imply p_4. When both p_2 and p_3 hold for the same first argument then p_1 holds between their second arguments. In this example, core prefixes of the relevant conjunctions will be marked with an underline.

- The conjunction $C_0 = [\underline{p_0(x_0)}]$ is closed.

[3] The original definition for the item-set case uses the *core index* which is the last item in the prefix. However, for our case, due to the potential for variable renaming we must use the entire prefix.

- Expanding C_0 with $p_2(x_0, x_1)$, no other atom is implied so we get the closed normal form:

$$C_1 = [\underline{p_0(x_0), p_2(x_0, x_1)}].$$

Clearly, this is a PPC-extension. The substitution table for C_1 is the following,

x_0	x_1
1	a
1	b
2	c
3	h
4	c
5	a

- Expanding C_0 with $p_3(x_0, x_1)$, we find that only $p_4(x_1)$ is implied. Therefore, we get the closed normal form:

$$C_2 = [\underline{p_0(x_0), p_3(x_0, x_1)}, p_4(x_1)].$$

This is a PPC-extension and the core prefix of C_2 is $[p_0(x_0), p_3(x_0, x_1)]$. The substitution table for C_2 is

x_0	x_1
1	a
2	b
3	a
4	b
5	c

- Expanding C_0 with $p_1(x_1, x_2)$ we find again that only $p_4(x_1)$ is implied. We get the closed normal form:

$$C_3 = [\underline{p_0(x_0), p_1(x_1, x_2)}, p_4(x_1)].$$

This is a PPC-extension and the core prefix of C_3 is $[p_0(x_0), p_1(x_1, x_2)]$. The substitution table for C_3 is

x_0	x_1	x_2
$1-5$	a	a
$1-5$	a	b
$1-5$	b	c
$1-5$	a	h
$1-5$	c	a
$1-5$	c	b

- Expanding C_2 with $p_2(x_0, x_2)$ we find that $p_1(x_1, x_2)$ is implied and get the closed normal form:

$$C_4 = [\underline{p_0(x_0), p_1(x_1, x_2), p_2(x_0, x_2), p_3(x_0, x_1)}, p_4(x_1)].$$

Notice that this extension violates conditions ppc2 and ppc3 and therefore this is not a PPC-extension. In this case, we can avoid the extension since condition ppc2 can be tested before

expanding. The substitution table for C_4 is

x_0	x_1	x_2
1	a	a
1	a	b
2	b	c
3	a	h
4	b	c
5	c	a

- Expanding C_1 with $p_3(x_0, x_2)$ we find two implied atoms with $p_4(x_2)$ and $p_1(x_2, x_1)$ respectively. The result is $[p_0(x_0), p_2(x_0, x_1), p_3(x_0, x_2), p_1(x_2, x_1), p_4(x_2)]$. In this case, the normal form renames the variables and we get the closed normal form C_4 (identical to previous case). Notice that this extension violates condition ppc3 and therefore this is not a PPC-extension, although we could not tell this in advance.

- Expanding C_3 with $p_2(x_0, x_2)$ we get the closed normal form:

$$C_5 = [\underline{p_0(x_0), p_1(x_1, x_2), p_2(x_0, x_2)}, p_4(x_1)].$$

  This is a PPC-extension and the core prefix of C_5 is $[p_0(x_0), p_1(x_1, x_2), p_2(x_0, x_2)]$. The substitution table for C_5 is

x_0	x_1	x_2
1	a	a
1	a	b
1	c	a
1	c	b
2	b	c
3	a	h
4	b	c
5	a	a
5	c	a

- Expanding C_5 with $p_3(x_0, x_1)$ we get the closed normal form C_4 (identical to previous cases). This is a PPC-extension and the core prefix of C_4 is $[p_0(x_0), p_1(x_1, x_2), p_2(x_0, x_2), p_3(x_0, x_1)]$.

- Expanding C_3 with $p_1(x_2, x_1)$ we get the closed normal form:

$$C_6 = [\underline{p_0(x_0), p_1(x_1, x_2), p_1(x_2, x_1)}, p_4(x_1), p_4(x_2)].$$

  This is a PPC-extension and the core prefix of C_6 is $[p_0(x_0), p_1(x_1, x_2), p_1(x_2, x_1)]$. The substitution table for C_6 is

x_0	x_1	x_2
$1-5$	a	a
$1-5$	b	c
$1-5$	c	b

In order to analyze the PPC construction, we need the following properties that guarantee that closures and cores are well behaved.

Lemma 29 *The following conditions hold for any conjunction C in normal form, any subset $C' \subseteq C \setminus \text{core}(C)$, and any atom $p \notin \text{closure}_{RR}(C)$:*
(1) $\text{closure}_{RR}(C) \subseteq \text{closure}_{RR}([C,p])$*, and,*
(2) $\text{closure}_{RR}([C,p]) = \text{closure}_{RR}([\text{core}(C),p]) = \text{closure}_{RR}([\text{core}(C),C',p])$.

Proof. For property (1) notice that any atom A in $\text{closure}_{RR}(C)$, but not in C satisfies $\mathcal{D} \models C \to A$. Now Lemma 23 implies that we also have $\mathcal{D} \models [C,p] \to A$. Iterating this argument we can add all atoms in $\text{closure}_{RR}(C)$ one by one to $\text{closure}_{RR}([C,p])$.

For property (2) consider starting closure_{RR} with one of $[\text{core}(C),p]$ or $[\text{core}(C),C',p]$. Since $C \subseteq \text{closure}_{RR}(\text{core}(C))$, by part (1) we can add all atoms from C first and then continue with the closure calculation. Now since closure is unique we get the same closure in both cases. ■

Remark 30 It is interesting to note that part (2) of the lemma above is violated by *lgg* closures in the LI setting. To see this, consider the following database $\mathcal{D} = \{I_1, I_2, I_3\}$ including three interpretations:

$$I_1 = [p_0(a), p_1(a,1), p_1(a,2), p_2(a,1), p_3(a,2)],$$
$$I_2 = [p_0(b), p_1(b,3), p_1(b,4), p_2(b,3), p_3(b,3), p_4(b,3)],$$
$$I_3 = [p_0(c)].$$

The conjunction $C = [p_0(x_0), p_1(x_0,x_1), p_1(x_0,x_2), p_1(x_0,x_3), p_1(x_0,x_4), p_2(x_0,x_1), p_3(x_0,x_2)]$ is the *lgg* of interpretations I_1 and I_2, and C is closed. Note that the core just needs any atom after p_0 to exclude I_3, so we have $\text{core}(C) = [p_0(x_0), p_1(x_0,x_1)]$. Now let $p = p_2(x_0,x_2)$. Then, $[C,p]$ is only satisfied by I_2, but $[\text{core}(C),p]$ is satisfied by I_1 and I_2 so their closure is clearly different.

We are now in a position to analyze PPC extensions. The next lemma relates the core of a query to the cores of its extensions.

Lemma 31 *Let C be a closed conjunction and C' a PPC-extension of C obtained as in condition* ppc1 *in Definition 27. Then,* $\text{core}(C') = [C[j],p]$*, where $C[j]$ is given by condition* ppc3.

Proof. We have that $C' = \text{nf}(\text{closure}_{RR}([C,p])) = \text{nf}(\text{closure}_{RR}([C[j],p]))$ by property (2) of Lemma 29 and condition ppc2. Then, by condition ppc3, $T = [C[j],p]$ is a prefix of C' and since C' is in normal form the core prefix of C' is a prefix of T. Now $\text{core}(C') = T$, i.e. we cannot omit p since $\text{nf}(\text{closure}_{RR}(C[j])) = \text{nf}(\text{closure}_{RR}(\text{core}(C))) = C \neq C'$. ■

The following lemma shows completeness, that is, that every closed query has a parent through PPC extensions.

Lemma 32 *Let C' be a closed conjunction in normal form. Let $T = \text{core}(C') = [L,p]$ (that is, L is a conjunction and p is the last atom in T). Let $C_1 = \text{closure}_{RR}(L)$ and let $C = \text{nf}(C_1)$. Then, C' is a PPC-extension of C.*

Proof. By Lemma 13, L is in normal form since it is a prefix of a normal form. Consider atoms added to L by the closure operation. By (1) of Lemma 29, the closure of L is a subset of C'. Notice also that p is not in the closure since otherwise T would not be the core prefix of C'. Any added

atom q satisfies $q \notin L$ and $q \in C'$ so it follows that $q > p$. Let $q_1, \ldots, q_n$ be the atoms added to L in this process, so that $C_1 = [L, q_1, \ldots, q_n]$ and C_1 is a subset of C' as in Lemma 13.

Therefore, by Lemma 13 L is a prefix of $C = \mathrm{nf}(C_1)$, and we also have that the core prefix of C is a prefix of L (since C is obtained from the closure of L and L appears in its normal form). As a result C' satisfies the conditions of a PPC-extension. Condition ppc1 holds since $C' = \mathrm{closure}_{RR}(T) = \mathrm{closure}_{RR}([L, p]) = \mathrm{closure}_{RR}([C, p])$, where the last equality is true by part (2) of Lemma 29. Condition ppc2 is satisfied since p is greater than all atoms in L and L includes the core of C. Condition ppc3 holds by construction since T is a prefix of C'. ■

The previous two lemmas together imply that a closed set has precisely one parent under the definition of PPC-extensions. This is formalized in the following theorem.

Theorem 33 *Let C' be a non-empty closed conjunction of atoms. Then, there is exactly one conjunction C such that C' is a PPC-extension of C.*

Proof. Let C be as in Lemma 32. Then, we know that C' is a PPC-extension of C. Consider any other conjunction C'' in normal form such that C' is a PPC-extension of C''. Let $j_C, j_{C''}$ be the indices defined as in condition ppc3 of Definition 27 with respect to C and C'' respectively. By Lemma 31, we have that $\mathrm{core}(C') = [C[j_C], p_C] = [C''[j_{C'}], p_{C'}]$ implying that $C[j_C] = C''[j_{C''}]$ so that $\mathrm{core}(C) = \mathrm{core}(C'')$ and that $C = C''$. ■

From this theorem we can finally get the promised generalization of LCM. However, there is one final caveat to consider: the set of closed patterns is infinite since adding a variable changes the semantics of the conjunction. Therefore, one should employ a "depth bound" on the searched patterns. This can be instantiated with a bound on the number of atoms, a bound on the number of refinements done, or a bound on the number of variables in the pattern. The algorithm below is started by calling RelLCM2($\mathrm{closure}_{RR}(\emptyset)$).

```
RelLCM2(input: closed pattern C = q_1, ..., q_n)
 1  if C violates depth bound then return
 2  if C is not frequent then return
 3  output C
 4  for all refinements [C, p] with p ∈ ρ(C)
 5            and s.t. p is greater than all atoms in core(C)
 6            if covers_SI([C, p]) = ∅
 7               then skip refinement
 8               else  Calculate Ĉ = nf(closure_RR([C, p]))
 9                     Let C[j] = [q_1, ..., q_j], where
10                        [q_j is largest atom in C with q_j < p]
11                     if [C[j], p] is a prefix of Ĉ
12                        then RelLCM2(Ĉ)
13  return
```

Theorem 34 *In any run of the algorithm the number of calls to* RelLCM2 *is exactly the number of frequent closed sets under the specified depth bound.*

Proof. Theorem 33 shows that every closed pattern has exactly one legal parent. By induction on the number of predecessors on the path to the empty pattern we get that the algorithm will discover every closed pattern through its legal parent. When this happens the prefix test in the algorithm holds and the pattern is expanded. By condition ppc3, the same test also detects any time a closed set is discovered via a non-legal parent. ■

While the number of calls is linear, we do discover some patterns more than once. The maximum number of patterns discovered is bounded by the branching factor of the refinement operator times the number of closed sets. Notice that in addition to avoiding extra storage and tests, the use of PPC-extensions reduces the branching factor.

Remark 35 Notice that for *lgg* closures, we can apply the idea of RelLCM2 from two directions. Remark 30 pointed out that working with refinements over conjunctions may be tricky. However, the results of Section 3 show that we can also traverse the lattice of closed sets of interpretations to get the desired result. In this case we simply have a subset relation over sets of interpretations and it is easy to derive the required PPC conditions. In fact, the propositional proof of Uno et al. (2004) suffices here. However, if we run RelLCM2 over sets of interpretations, we start with the empty set of interpretations and keep on adding interpretations to it. Thus, we start with the most specific conjunction and generalize it in the process. This is the opposite direction from the usual traversal from the empty conjunction, where we add atoms to make it more specific. It has the disadvantage that the relation is no longer anti-monotonic and cannot be combined with the threshold of frequent sets. Obviously for this to work we need to allow for sets with even just one interpretation. So, this approach is only useful with a trivial threshold value. Another difficulty is that the branching factor in this form is equal to the number of interpretations which can be very large.

Remark 36 As discussed for the LI setting, a potential improvement may be possible by using the idea of instantiations (Maloberti & Suzuki, 2003), where we compare instances of a clause instead of just substitutions. Here we might say that a refinement is *useful* for C whenever it leads to different instances in $\mathcal{D}$. For those refinements leading to the same instances, we know that the obtained closed set would be useless (equivalent to a previous one that we already computed). Then, the redundant closed set does not need to be output. These ideas can be developed formally leading to a reduction in the size of the output. Unfortunately, our current construction does not allow for full pruning; one can construct examples showing that the parent of some useful closed sets is redundant, so one cannot prune the search when reaching a redundant clause. We leave this idea of pruning based on instances for future research.

5 Discussion

The paper investigates different semantic settings for mining patterns in relational data, including graph and network based datasets. We have demonstrated that variant definitions from the literature have significant implications for properties of closed sets, and algorithms for discovering them. The paper developed relational variants of the LCM algorithm that are a promising alternative for mining closed conjunctions in multi-relational datasets. Our results are summarized in Table 1. We have shown that in the setting using LI and object identity (thus also graph mining under subgraph isomorphism) closures of patterns are not unique. As a result the LCM algorithm cannot

	OI-subsumption	θ-subsumption
LI	• Setting of CloseGraph (Yan & Han, 2003) and Farmer (Nijssen & Kok, 2003). • Closures of patterns are not unique. • LCM cannot be used. • Limited to methods that search by applying refinements (in BFS or DFS mode) until a closed set is found.	• Setting of Warmr (De Raedt & Ramon, 2004; Dehaspe & Toivonen, 2000) and Jimi (Maloberti & Suzuki, 2003). • Unique closures can be defined via lgg of conjunctions. • LCM can be applied, but PPC-extension properties do not hold. • Limited to store previously discovered sets to avoid duplicates.
SI	• Setting of Claudien (De Raedt & Dehaspe, 1997); full generalization of LCM is possible with range-restriction.	

Table 1: Summary of results in different semantic settings.

be generalized and this setting is limited to methods that search by applying refinements (in BFS or DFS mode) until a closed set is found. For the setting using LI and θ-subsumption, one can define unique closures, but PPC-extension properties do not hold. In this case, one can enumerate closed sets as in LCM however one must store previously discovered sets to avoid duplicates in the output. Finally, for the setting using SI and range restriction (with either object identity or θ-subsumption), we show a full generalization of the LCM algorithm that does not need to store previous sets in the enumeration.

This paper has focused on theoretical foundations for closed relational patterns and a thorough experimental evaluation is beyond the scope of the paper. A preliminary experimental evaluation showing promising results was reported in (Garriga et al., 2007). For the LI setting the calculation of the covers() relation, which is needed to determine closure, can be implemented efficiently assuming an efficient subsumption procedure. However, for the SI setting, calculating the value of covers() represents a significant challenge for the implementation of the LCM. In this case, covers() is the set of all substitutions for the pattern and it can be exponential in the number of variables, for example, when the portions constraining each variable are independent of each other. Therefore, an efficient implementation must make use of compact representations for such sets (cf. (Di Mauro et al., 2003)) or use an alternative algorithm for implication or a different semantics for covers(). Unfortunately, as shown in Remark 25, while the projcovers() semantics solves the size problem and can be calculated more efficiently, it cannot be used with LCM. Identifying an alternative that is intuitively appealing, computationally tractable, and semantically coherent is an important challenge for future work.

6 Acknowledgments

This work was partly supported by NSF Grant IIS-0099446 and a research semester fellowship award from Tufts University (Roni Khardon) and by the EU IST FET project IQ (Luc De Raedt).

References

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *Proc. of ACM SIGMOD Conference on Management of Data* (pp. 207–216).

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. (1996). Fast discovery of association rules. *Advances in Knowledge Discovery and Data Mining*, 307–328.

Arimura, H., & Uno, T. (2005). An output-polynomial time algorithm for mining frequent closed attribute trees. *Proc. 15th Conference on Inductive Logic Programming* (pp. 1–19).

Balcázar, J., & Garriga, G. (2007). Horn axiomatizations for sequential data. *Theor. Comput. Sci.*, *371*, 247–264.

Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., & Lakhal, L. (2000). Mining minimal non-redundant association rules using frequent closed itemsets. *Lecture Notes in Computer Science*, *1861*, 972–986.

Blair, R., Fang, H., Branham, W., Hass, B., Dial, S., Moland, C., Tong, W., Shi, L., Perkins, R., & Sheehan, D. (2000). The estrogen receptor relative binding affinities of 188 natural and xenochemicals: Structural diversity of ligands. *Toxicol. Sci.*, *54*, 138–153.

Boros, E., Gurvich, V., Khachiyan, L., & Makino, K. (2003). On maximal frequent and minimal infrequent sets in binary matrices. *Ann. Math. Artif. Intell.*, *39*, 211–221.

Branham, W., Dial, S., Moland, C., Hass, B., Blair, R., Fang, H., Shi, L., Tong, W., Perkins, R., & Sheehan, D. (2002). Binding of Phytoestrogens and Mycoestrogens to the rat uterine estrogen receptor. *J. Nutr.*, *132*, 658–664.

Bringmann, B., & Nijssen, S. (2008). What is frequent in a single graph? *Proceedings 12th Pacific-Asia Conference on Knowledge Discovery in Databases* (pp. 858–863). Springer.

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

De Raedt, L., & Dehaspe, L. (1997). Clausal discovery. *Mach. Learn.*, *26*.

De Raedt, L., & Ramon, J. (2004). Condensed representations for Inductive Logic Programming. *Proc. of the 9th International Conference on Principles of Knowledge Representation and Reasoning* (pp. 438–446).

Dehaspe, L., & Toivonen, H. (2000). Discovery of relational association rules. *Relational Data Mining*, 189–208.

Deshpande, M., Kuramochi, M., & Karypis, G. (2003). Frequent sub-structure-based approaches for classifying chemical compounds. *Proc. of the Third IEEE International Conference on Data Mining* (pp. 35–42).

Di Mauro, N., Basile, T., Ferilli, S., Esposito, F., & Fanizzi, N. (2003). An exhaustive matching procedure for the improvement of learning efficiency. *Proceedings 13th International Conference on Inductive Logic Programming* (pp. 112–129). Springer.

Fang, H., Tong, W., Shi, L., Blair, R., Perkins, R., Branham, W., Hass, B., Xie, Q., Dial, S., Moland, C., & Sheehan, D. (2001). Structure-activity relationships for a large diverse set of natural, synthetic, and environmental estrogens. *Chem. Res. Tox.*, *14*, 280–294.

Fiedler, M., & Borgelt, C. (2007). Support computation for mining frequent subgraphs in a single graph. *Proceedings of the Workshop on Mining and Learning with Graphs.*

Ganter, B., & Wille, R. (1998). *Formal Concept Analysis. mathematical foundations.* Springer.

Garriga, G., Khardon, R., & De Raedt, L. (2007). On mining closed sets in multi-relational data. *Proceedings of the 20th International Joint Conference on Artificial Intelligence.*

Goethals, B., & Zaki, M. (2004). Advances in frequent itemset mining implementations: report on FIMI'03. *SIGKDD Explor. Newsl.*, *6*, 109–117.

Gunopulos, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., & Sharma, R. (2003). Discovering all most specific sentences. *ACM Transactions on Database Systems*, *28*.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *Proc. of the ACM SIGMOD International Conference on Management of Data* (pp. 1–12).

Horváth, T., Alexin, Z., Gyimóthy, T., & Wrobel, S. (1999). Application of different learning methods to hungarian part-of-speech tagging. *Proceedings 9th International Workshop on Inductive Logic Programming* (pp. 128–139).

Horváth, T., & Turán, G. (2001). Learning logic programs with structured background knowledge. *Artificial Intelligence*, *128*, 31–97.

Kramer, S., & De Raedt, L. (2001). Feature construction with version spaces for biochemical applications. *Proceedings of the 18th International Conference on Machine Learning* (pp. 258–265).

Kuramochi, M., & Karypis, G. (2004). Finding frequent patterns in a large sparse graph. *Proceedings of the Fourth SIAM International Conference on Data Mining*. SIAM.

Kuznetsov, S. (1999). Learning of simple conceptual graphs from positive and negative examples. *Proceedings of the 3rd European Conference on Principles and Practive of Knowledge Discovery in Databases* (pp. 384–391).

Kuznetsov, S. (2004). Machine learning and formal concept analysis. *Proceedings of the 2nd International Conference on Formal Concept Analysis* (pp. 287–312).

Kuznetsov, S., & Samokhin, M. (2005). Learning closed sets of labeled graphs for chemical applications. *Proceedings of the 15th International Conference on Inductive Logic Programming* (pp. 190–208).

Lloyd, J. (1987). *Foundations of logic programming*. Springer Verlag.

Malerba, D., & Lisi, F. (2001). Discovering associations between spatial objects: An ILP application. *11th International Conference on ILP* (pp. 156–163).

Maloberti, J., & Suzuki, E. (2003). Improving efficiency of frequent query discovery by eliminating non-relevant candidates. *Discovery Science* (pp. 220–232).

Mannila, H., & Toivonen, H. (1997). Levelwise search and borders of theories in knowledgediscovery. *Data Mining Knowledge Discovery*, *1*, 241–258.

McCallum, A., Nigam, K., Rennie, J., & Seymore, K. (1999). A machine learning approach to building domain-specific search engines. *Proc. of the 16th International Joint Conference on Artificial Intelligence*.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *20*, 629–679.

Nienhuys-Cheng, S., & De Wolf, R. (1997). *Foundations of inductive logic programming*. No. 1228 in Lecture Notes in Artificial Intelligence. Springer-verlag.

Nijssen, S., & Kok, J. (2003). Efficient frequent query discovery in FARMER. *Proc. of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases* (pp. 350–362).

Pei, J., Han, J., & Mao, R. (2000). CLOSET: An efficient algorithm for mining frequent closed itemsets. *Proc. of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (pp. 21–30).

Plotkin, G. D. (1970). A note on inductive generalization. In B. Meltzer and D. Michie (Eds.), *Machine intelligence 5*, 153–163. American Elsevier.

Schietgat, L., Costa, F., Ramon, J., & De Raedt, L. (2011). Effective feature construction by maximum common subgraph sampling. *Machine Learning*, *83*, 137–161.

Uno, T., Asai, T., Uchida, Y., & Arimura, H. (2004). An efficient algorithm for enumerating closed patterns in transaction databases. *Proceedings of the 7th International Conference on Discovery Science* (pp. 16–31).

Yan, X., & Han, J. (2002). gSpan: Graph-based substructure pattern mining. *Proceedings of the 2nd IEEE International Conference on Data Mining* (pp. 721–724).

Yan, X., & Han, J. (2003). CloseGraph: mining closed frequent graph patterns. *Proc. of the 9th ACM SIGKDD International conference on Knowledge discovery and data mining* (pp. 286–295).

Zaki, M. (2004). Mining non-redundant association rules. *Data Mining and Knowledge Discovery: An International Journal*, *4*, 223–248.

Zaki, M., & Hsiao, C. (2002). CHARM: An efficient algorithm for closed itemset mining. *Proc. of the 2nd. SIAM International Conference on Data Mining.*