MINE GRAPH RULE: A New GQL Operator for Mining Association Rules in Property Graph Databases

Francesco Cambria · Francesco Invernici · Anna Bernasconi · Stefano Ceri

Received: 27 January 2025 / Accepted: 17 June 2025

Abstract Mining information from graph databases is becoming overly important. To approach this problem, current methods focus on identifying subgraphs with specific topologies; as of today, no work has been dedicated to jointly expressing the syntax and semantics of mining operations over rich property graphs.

We define `MINE GRAPH RULE`, a new operator for mining association rules from property graph databases, by following a research trend that has already been pursued for relational and XML databases. We describe the syntax and semantics of the operator, which allows measuring the support and confidence of each rule, and then we show many examples of increasing complexity, thereby providing a gentle introduction to the rich expressive power of the language, which is designed to be easy-to-use by GQL experts.

Although the emphasis of this paper is on providing the syntax and semantics of the `MINE GRAPH RULE` operator, with several examples of use, we also developed an implementation of the operator on top of Neo4j, the most successful/adopted graph database system to date; the implementation is available as a portable Neo4j plugin, which we use to showcase real-world applications.

At the end of our paper, we show the execution performance in a variety of synthetically generated settings, by varying the text of operators, the size of the graph, the ratio between node types, the method for creating relationships, and the maximum support and confidence; we also show our operator at work on two real-life graphs respectively describing music playlists and archived literature, and provide interesting examples of extracted association rules.

F. Cambria
Via Ponzio 34/5, Politecnico di Milano, DEIB
E-mail: francescoluciano.cambria@polimi.it

F. Invernici
Via Ponzio 34/5, Politecnico di Milano, DEIB
E-mail: francesco.invernici@polimi.it

A. Bernasconi
Via Ponzio 34/5, Politecnico di Milano, DEIB
E-mail: anna.bernasconi@polimi.it

S. Ceri
Via Ponzio 34/5, Politecnico di Milano, DEIB
E-mail: stefano.ceri@polimi.it

1 Introduction

Association rules for relational datasets take the form $X \Rightarrow Y$, where the left-hand side X is denoted as body and the right-hand side Y is denoted as head Both are sets instantiated from the same underlying domain (e.g., items), and their pairing within rules occurs statistically more often than "normal" in the context of certain groups of tuples within the same relational dataset (e.g., all tuples referring to the same transaction) [6,7,55]. Statistical significance is measured by two indexes, called support and confidence; the former is the probability of having both items X and Y within all transactions; the latter is the probability of having the Y item in the transactions with the X item.

Association rules satisfy a classical antimonotonicity property stating that, for any two rules $r_1 : X_1 \Rightarrow Y$ and $r_2 : X_2 \Rightarrow Y$, with $X_1 \supset X_2$ (e.g., X_1 properly contains all the items of X_2), the support of r_1 is smaller or at most equal than the support of r_2. The antimonotonicity property and the constraint on minimum support allow, when the support for a given association rule $r : X \Rightarrow Y$ is below the threshold, to exclude also all rules whose body strictly includes X [59]. This property is at the basis of efficient mining rule methods, such

as Apriori [30,48] or Equivalence Class Transformation Algorithm (ECLAT) [62,35], which organize the search by arranging extracted bodies in a tree and cease exploring descendants of nodes that are not supported.

The classical interpretation of association rules in recommender systems [16,36,44] is to propose items that are often purchased in the same transaction, as an indication that they are jointly chosen by customers who – in turn – may share the same taste, although it is possible to have different interpretations of association rules for many other contexts of application, e.g., in the medical domain symptoms of patients affected by the same disease [57,42,58], or in social networks interests that are shared by users [54]. Pattern languages have been designed for extracting given rules based upon application needs, e.g., focusing on given items in purchases (e.g., bread and butter), on given habits/morbidities in patients (e.g., male smokers), and on given interests among users (e.g., movies and music).

One such pattern language is the `MINE RULE` operator, defined in [40], which extracts semantically meaningful association rule patterns on relational datasets. The operator uses the expressive power of SQL and consists of several clauses, each mimicking an SQL query or predicate. Typically, `MINE RULE` extracts tables with four columns [`BODY`, `HEAD`, `SUPPORT`, `CONFIDENCE`], each corresponding to an association rule, extracted from an underlying target table, constructed as an arbitrary SQL query expression whose tuples can be arbitrarily grouped and further partitioned; the table extracts all rules having support and confidence beyond a given threshold.

In the digital age, vast amounts of data are being generated and collected at an unprecedented pace [5, 45]. Relational databases are very effective in managing structured data, but face limitations when handling complex and interconnected data [52,41]; graph databases are emerging as the leading data management technology for storing large knowledge graphs [60, 47]. Therefore, there is a strong need to express association rules in this context. The main challenge when extending the broad work on relational association rule mining to graph-based association rule mining is to move from a simple, tabular data model, where associations are discovered within simple groups, to a complex graph-based data model, where associations can be semantically richer, and at the same time adapt the notions of support and confidence so as to preserve the antimonotonicity property, guaranteeing good convergence of the discovery methods.

In this work, we present a new operator for graph databases, called `MINE GRAPH RULE`, which is inspired by the relational `MINE RULE` operator [40]; the operator provides a declarative definition that takes advantage of the schema description available for property graphs (essentially labeled nodes and relationships with properties). We redefine the notion of 'items' (introducing the novel concept of 'anchor nodes') and progressively define the left and right sides of the association rules extracted by the `MINE GRAPH RULE` operator; left and right sides are built by arbitrarily constructed orthogonal structures that support semantically rich path expressions, whose syntax and semantics are inspired by GQL, the emerging standard for graph query languages [20,50]. Our operator adapts the relational notion of *enclosing transactions* to property graphs, thereby allowing the computation of support and confidence for the pair of left and right sides extracted by the operator, along the classic semantics of association rules; as in [40], we extract association rules whose support and confidence is above given thresholds.

This work is different from most previous work in graph data mining, which typically focuses on searching for regular structures that form interesting subgraphs that reflect given constraints upon their nodes or edges, without taking full advantage of the availability of schema description. Another main difference between our method and related work is that our declarative patterns can be translated to GQL-compliant Cypher and therefore be executed on graph database engines, taking advantage of their physical organization and query optimization methods, whereas most previous work is concerned with algorithmic approaches over ad-hoc data structures built from imported files, using programming languages such as Java [24] or C++ [22].

Contributions

This article presents five significant contributions:

- Syntax and semantics of the `MINE GRAPH RULE` operator. The operator supports the extraction of many items in the body and in the head, defined by a conjunction of relationship chains of GQL-compliant expressions of arbitrary length, which can be orthogonally composed.
- A rich set of progressive examples, which showcase the expressive power of the operator by extracting association rules of increasing complexity, both for what concerns the operator expressions and the complex tabular structure which returns the association rules whose support and confidence are above threshold.
- Implementation and deployment of the operator to a real GQL-compliant [20,50] graph database. Among existing graph databases, ranked in [2], we choose

the top-ranked Neo4j system [34,27]. We then describe how the MINE GRAPH RULE operator can be installed as a Neo4j plugin. The mining algorithm takes advantage of built-in optimizations of the Neo4j engine as well as optimizations that take advantage of the *Apriori* approach.
– Performance evaluation of several data mining operators on artificially generated graphs, by varying several settings (including the artificial graph generation, graph size, relationship density, and minimum support/confidence).
– Application of the operator to two large real-life datasets, respectively describing Spotify playlists and archived literature, and extraction of interesting association rules that showcase the potential of the approach.

Outline

This article is organized as follows: Section 2 presents the related work; Section 3 describes the syntax and semantics of the MINE GRAPH RULE Operator and Section 4 illustrates several examples of increasing complexity, also showing the operator's output, produced as a table with several rows, one for each association rule, listing body, head, support, and confidence. Section 5 describes the implementation and the generation of the example graph database, and Section 6 describes the evaluation of our approach. Section 7 compares our work with PARM, a recently presented approach which also applies to property graphs. Finally, Section 8 presents the discussion and conclusion.

2 Related Work

According to the panel discussion reported in [11], one of the key needs to improve graph database analytics is the development of more expressive languages, capable of supporting complex and diverse analyses. MINE GRAPH RULE aims to address this challenge, by providing a declarative pattern language that defines complex association rules for property graph databases.

Declarative approaches for association rule mining. In the existing data mining literature, several declarative pattern languages have been defined; among them, the highly cited MINE RULE operator (by Meo, Ceri, and Psaila) [40,39] presents an SQL-based pattern language that extracts association rules calculated by using arbitrary data grouping and partitions; the operator produces readable outputs in tabular form, taking advantage of the NF2 model [4], where each rule is associated with given support and confidence, enabling their filtering and ordering based on such statistical properties. The same authors provided a pipeline and framework to translate the MINE RULE operator to NF2 tables, using SQL [39]. The MINE RULE approach and framework were then exploited by Boulicaut, Klemettinen, and Mannila, who define inductive databases as databases augmented by generic patterns and by an evaluation function telling how the pattern occurs in the data. In [12], they explain that the MINE RULE approach perfectly fits the inductive database vision, being a *serious step towards an implementation framework for inductive databases.* The adaptation of MINE RULE to XML was proposed by Braga, Campi, Klemettinen, and Lanzi [13], who defined XMINE as an operator to discover association rules based on the XQuery language.

Graph pattern mining. Most works in graph data mining look for regular structures in a graph, without taking advantage of a schema description; they search for interesting networks of nodes and edges, extracted from the graph, which reflect given constraints. For doing so, they typically read the graph into a memory structure and propose optimal algorithms, executed using programming languages such as Java or C++. Of course, the various definitions of *pattern* (GPAR, GFD, GAR, REE, PARM: see below) reflect the antimonotonicity property, which is the core of association rule mining.

A framework for Graph Pattern Association Rule (GPAR) Mining is defined by Fan et al. in [24], where a pattern is a subgraph is an arbitrary selection of nodes and edges from the graph (e.g., extracted by a query), and a rule is defined as follows: *when a pattern in which two designated nodes x and y are present is frequent in the graph, then also an edge between x and y is present*; for computing rule support, they use the minimum image-based support, as introduced in [14], with a suitable revision to satisfy the antimonotonicity constraint. Their D-MINE algorithm uses ad-hoc auxiliary structures and takes advantage of parallelization to extract rules that are very interesting (emerging from a top-k selection for diversified patterns).

This work was extended by Wang and Xu [61], who redefine GPAR, as rules are between two patterns such that each pattern is connected (includes connected nodes and edges) and two patterns cannot share any edge. The method is based on bisimulation as pattern matching semantics. They show that the problem can be decomposed into two steps, called frequent pattern mining and rule generation; their algorithm (FPMiner) exploits parallelism as well as look-ahead and backtracking to

discover frequent patterns, defined as those above a minimal given threshold.

Another interesting approach, also proposed by Fan et al., is concerned with mining graph functional dependencies (GFDs) [25], i.e. attribute-value dependencies and topological structures of entities, using an algorithm that explores trees progressively built out of graphs; the problem is coNP-complete, hence they develop an efficient parallel version that can cope with such complexity in specific contexts. Parallelism was later applied, by Fan et al. [22], with application-driven reduction and sampling, to efficiently extract Graph Association Rules (GARs) from big graphs.

Recent work by Fan et al. [23] has discussed how to exploit parallelism and sampling in order to mine an extended set of Entity Enhancing Rules (REEs), which subsume functional dependencies and many other cases of dependencies – named conditional dependencies, denial dependencies, and match dependencies (see [23]) suggested by Rock, an industrial system for data cleaning. Their method, called PRMiner, includes sampling and parallelism. Along this line of thought, recent work by Liu et al. [37] has proposed a comprehensive framework designed to define and extract graph patterns by exploiting *oracles*, i.e., abstract machines developed for making decisions, typically either by importing external knowledge or by using internal computations such as aggregate operators or machine learning methods.

Other works address a broader problem, i.e. solving the so-called Frequent Subgraph Mining (FSM) problem [29], defined as finding all the subgraphs within a graph that appear frequently (more than a give threshold); in general, the solution consists of two steps: generating candidate subgraphs and calculating their support, along with the Apriori method. [21] presents a method to prune options in the step generation, [31] presents a solution in a single pass.

More recently, Sasaki and Karras defined Path Association Rule Mining (PARM) [49] as the problem of finding all the Path Association Rules (PAR) with specific thresholds on the paths' lengths and the rules' support. Given that PARM applies to property graphs and employs a similar approach to ours for the definition of graph association rules, it will be explored in greater detail in a dedicated section.

Association rules mining using graphs managed by Neo4j. Some works, with an application-oriented approach, address the extraction of association rules in real-world scenarios supported by Neo4j. Interesting case studies are reported in [17] (evaluating groups of artists appreciated by the same users) and in [56] (studying combinations of teaching methods that are more effective than others). In both works, data is only extracted from Neo4j and structured in a way that the association rules are calculated thanks to established Python packages [1,46]. [15] implements a case study in Neo4j to find popular Twitter hashtags that are posted together; the work includes an evaluation of the effect of pre-processing techniques that enhance the performance of the rule mining algorithm. In [51], a recommender system is fully developed in Neo4j. Note that all the mentioned works apply algorithms and the concepts of association rules to graph data, without formalizing rules and without fully exploiting the richness given by the natural structure of graphs.

3 Operator Syntax and Semantics

In graphs, the definition of association rules $r : X \Rightarrow Y$ can be expanded considering X and Y not as unique data entries but as patterns of nodes and relationships. Our operator aims to define precisely and effectively these graph patterns.

The syntax of `MINE GRAPH RULE` is shown in Fig. 1; it is inspired by the MINE RULE operator [40], but it is significantly more complex to exploit the many ways of building associations by using property graph databases. It embeds some constructs of GQL [20, 50], so that it can easily be adapted for all the GQL-compatible database systems, and it is also more readable for graph database programmers.

```
MINE GRAPH RULE <name>
GROUPING ON ( <anchor> : <anchorLabel>) [ WHERE <wherePredicate> ]
DEFINING BODY AS <itemSet>
              HEAD AS <itemSet>
[ WHERE <wherePredicate> ]
[ IGNORE <variable> [, <variable>]* ]
EXTRACTING RULES WITH SUPPORT > <minsupp> AND CONFIDENCE > <minconf>

<itemSet> ::= <pattern> | <pattern> , <itemSet>
<pattern> ::= [ 1..<n> ] ( <anchor> ) <relPattern> <patternTail> [AS <patternName>]
<patternTail> ::= ( <variable> : <nodeLabel> ) |
                  ( <variable> : <nodeLabel> ) <relPattern> <patternTail>
<relPattern> ::= <singleRel> | <countRel> | <anyRel>
<singleRel> ::= -[ <relType> ]->
<countRel> ::= -[ <relType> >= <mincount> ]->
<anyRel> ::= -[]->{1,<pathLength>}
```

Fig. 1 `MINE GRAPH RULE` syntax. Uppercase bold type is used for terminal symbols. Lowercase is for non-terminal symbols; bold symbols are not explained in the grammar, as their meaning is well-defined in the context of property graphs. Note that square brackets are used to indicate optionality ([]) in the grammar; moreover, since GQL uses the symbols `-[` and `]-` for denoting relationships, square brackets are also used in bold-type to enclose the three expressions of `<relPattern>`. Note as well that the < and > symbols, used to delimitate non-terminal symbols, are also used -in larger font size- to make comparisons in the pattern language: > is used to denote a 'greater-than' comparison in the support/confidence threshold definition and in the `<countRel>` grammar production.

To each operator we associate a <name>, allowing users to uniquely associate each operator with a set of extracted association rules. Rules are built according to three expressions, respectively defining the GROUPING, the rule's BODY, and the rule's HEAD. The grouping expression defines, through the <anchorLabel>, the set of *anchor nodes* of the graph that need to satisfy, when present, the WHERE condition. They provide the *context of evaluation of association rules*; by analogy, the *anchor* acts as a pivot for the set of rules in the same way the purchase transaction groups the basket of items that are bought together. The BODY and HEAD expressions build respectively the left and right parts of an association rule; they are both defined by the <itemSet> production.

An itemSet is defined as a conjunction of expressions, each called <pattern>, that extract the sets of items forming the head and body; the cardinality of each <pattern> varies between 1 (the default case) and a value <n> (the cardinality must be defined only if <n> is greater than 1). A <pattern> generally consists of a linear path of relationships that starts from the anchor nodes (i.e., the nodes defined in the grouping expression) and can include an arbitrary number of relationship patterns (<relPattern>); thanks to an optional alias clause, introducing a user-defined <patternName>, users can assign a custom name to each <pattern>. Note that in patterns, every node needs a corresponding variable associated with a node label; using the same variable name is allowed only upon nodes with the same label, and implies the identity of the nodes extracted by the operator.

Recursion is introduced by the <patternTail> production, which can either close the current recursion step with a final node or recursively include another <patternTail>. Relationship patterns include three options (respectively denoted as <singleRel>, <countRel>, and <anyRel>) whose explanation is postponed to three specific subsections. Note that -for simplicity- we assume that each relationship pattern is to be traversed from left to right, as explicitly expressed by the arrow direction.

The three expressions, GROUPING, BODY, and HEAD, extract nodes that are referenced by variables; these can be used both to build any GQL-legal predicate (<wherePredicate>) for evaluating restricted parts of the graph database or for adding other grouping conditions defined by the optional IGNORE clause with a list of at least one <variable>. The operator syntax is closed by setting two independent conditions for the *minimum support* <minsupp> and *minimum confidence* <minconf> that must be satisfied by the extracted rules.

Concerning the semantics of pattern expressions, an important aspect is that the MINE GRAPH RULE uses the "trail" path mode semantics [3]; essentially, nodes can be revisited, but each relationship is used only once per path, to resolve the issue of cyclic instances. This is the most widely used semantics in graph database engines[1].

Concerning the extraction of association rules, we need to calculate both support and confidence. We define A as the set of anchor nodes; we denote A's cardinality as $C(A)$. Evaluating the body and head expression matched with a set of anchor nodes $a \in A$ generates pairs $R_i = \langle B_i, H_i \rangle$, which are candidate association rules. Next we discuss how B_i and H_i are built.

Consider that, both in the BODY and HEAD of a rule, each <itemSet> is associated with an ordered list of m conjunctive expressions of <pattern>s P_m, where each pattern corresponds to a list L_m of named variables that are neither anchors nor included in the IGNORE list of the pattern. Then, an operator evaluation, under trail semantics and different edges matching, maps each anchor node $a \in A$ (conceptually equivalent to a transaction) to a list $\mathbb{P}$ of lists P_j, with $1 \leq j \leq m$, such that each P_j is a list of nodes of the graph obtained by matching the corresponding named variables L_m; a list $\mathbb{P}$ is separately computed for the body (producing B_i) and head (producing H_i). We then consider the mapping from the set A of anchor nodes to the lists B_i and H_i and we denote as $C(B_i)$ the cardinality of the set of anchor nodes matched to B_i and as $C(R_i)$ the cardinality of the set of anchor nodes matched to both B_i and H_i.

Then, for a given rule R_i, its support is $C(R_i)/C(A)$ (by analogy: the fraction of transactions having both the body and head items in the basket). Its confidence is $C(R_i)/C(B_i)$ (by analogy: the fraction of transactions that have both the body and the head over the transactions that have only the body). Finally, the operator extracts those candidate association rules that satisfy minimal constraints on support and confidence.

Each operator execution over a graph produces a non-normalized table (along the NF2 model [4]); the table is named as the operator and has four top-level columns. The first two columns carry names which are progressively constructed to reflect the specific pattern expressions for body and head (path expressions can be renamed using aliases); the third and fourth columns contain support and confidence.

Each row of the table corresponds to an extracted association rule; the first and second columns respec-

[1] In addition to "trail", path modes include "walk", "acyclic", and "simple". These path modes provide alternative path bindings, as discussed in Section 4.11.7 of the GQL standard [3].

tively contain the complex structures B_i and H_i, describing the rule's body and head. Each node forming the structures B_i and H_i is represented by system-generated node identifiers ([3], Section 3.1.7); for ease of readability, for each node used in operators, we provide a map (<NodeLabel>: <IdentifyingPropertyList>), where the latter list is a user-provided identifier, made of suitable node properties, which are used instead of internal node identifiers for providing readable association rule instances[2].

Note that, in the search for association rules, it is customary to exclude tautological rules, i.e., rules whose body and head are identical. Similarly, in the evaluation of the `MINE GRAPH RULE` pattern, we will exclude those rules whose body and head are identical.

4 Progressive Illustration of the Expressive Power of `MINE GRAPH RULE`

We next proceed to illustrate the expressive power of the operator, starting with a simple case that mimics the market basket analysis and then progressively addressing more complex cases.

4.1 Running Example

Fig. 2 reports the schema of a property graph database, storing information about people's purchases that may reflect their social interactions; for easing interpretation, nodes and relationships of the running example are partitioned into subsets and each subset is associated with a specific label. The `PERSON` nodes represent users of a social network, who can `FOLLOW` each other and who can `RECOMMEND` certain products represented by `ITEM` nodes; the `BUY` relationship indicates that given persons buy given products using an electronic marketplace, whereas the `OF` relationship indicates a non-exclusive `CATEGORY` which is assigned to each item by vendors on the basis of the expected context of use of that item.

We also provide a super-small instance of the graph, shown in Fig. 3, in order to progressively introduce examples of `MINE GRAPH RULE` applications. Every node is uniquely identified by its `Name` and carries specific properties: persons have `Age` and `City`, items have `Color` and `Price`.

[2] If a node does not have an identifier, it will appear in association rules as many times as it is extracted by the relevant pattern, without being able to give an identity to each node occurrence; but this could be considered as a poor combined design of the scheme and operator.

Fig. 2 Schema of the running example

4.2 Simple Association Rules

As a starting point, we show the most common application of association rules, i.e., pairs of different items usually bought together by the same person. These association rules are expressed by the natural language sentence: *"People who buy X, also buy Y"*, where X and Y are different items. The `MINE GRAPH RULE` operator extracting these rules is simply:

```
MINE GRAPH RULE SimpleAssociationRules
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[:BUY]->(X:Item)
         HEAD AS (p)-[:BUY]->(Y:Item)
EXTRACTING RULES WITH SUPPORT > 0.1
                  AND CONFIDENCE > 0.1
```

The operator looks for all nodes labeled as `(:Person)`. Both body and head expressions extract a single `(:Item)`, connected by using the `[:BUY]` relationship; hence the association rule combines the different items that are bought by the same person. Once applied to the graph database instance of Fig. 3, the operator produces the result shown in Table 1. The table's header has a hierarchical structure; the top of the hierarchy has four fixed attributes, respectively named `Body`, `Head`, `Support`, and `Confidence`. Below the `Body` and `Head`, the header includes terms describing the role of each extracted instance forming an association rule (in this case, `BuyX` and `BuyY`). The table's content includes one row for each association rule; `Body` and `Head` include the `Names` (i.e., the identifying property of nodes) describing the extracted Items.

Body	Head	Support	Confidence
BuyX	BuyY		
Dress	Jeans	0.25	0.5
Dress	Shoes	0.5	1
Jeans	Dress	0.25	0.5
Jeans	Shoes	0.25	0.5
Shoes	Dress	0.5	1
Shoes	Jeans	0.25	0.5
Shorts	T-shirt	0.25	1
T-shirt	Shorts	0.25	1

Table 1 Output of SimpleAssociationRules

Let us consider the first row of the table, corresponding to the rule: *"People who buy Dress, also buy Jeans"*. For this row, $C(A) = 4$, $C(B_1) = 2$, $C(R_1) = 1$. The first counter reports the total number of people

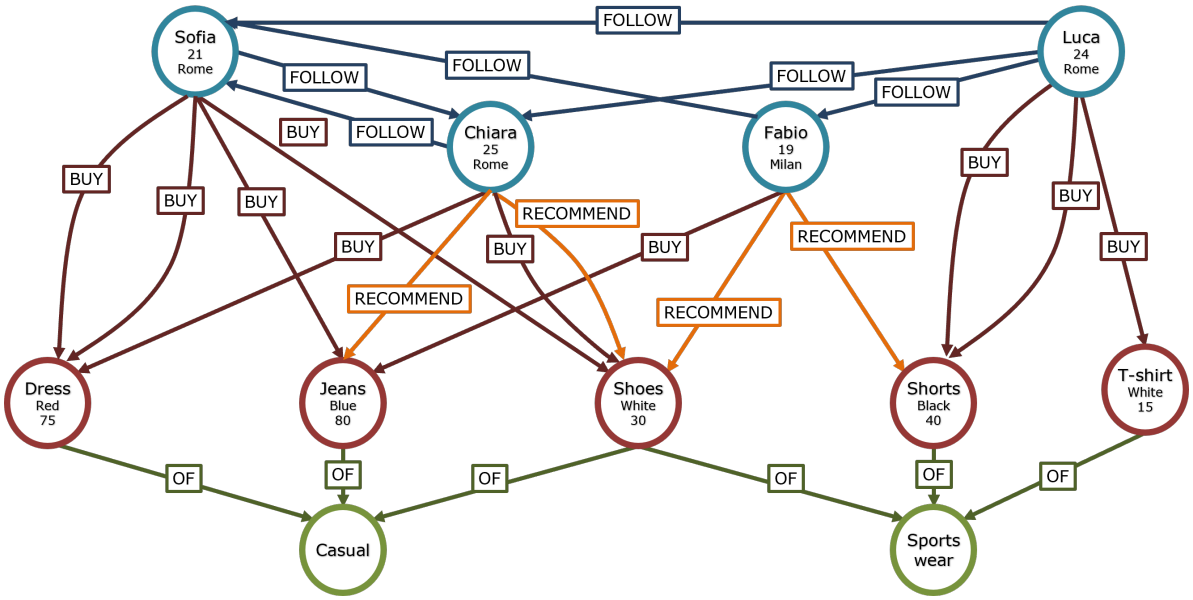

Fig. 3 Instance of the running example

(anchor node), the second the number of people who buy Dress (hence, Sofia and Chiara), and the third the total number of people who buy both Dress and Jeans (hence, just Sofia). Given these counters, the rule support is $C(R_1)/C(A) = 0.25$, the rule confidence is $C(R_1)/C(B_1) = 0.5$.

4.3 Association Rules with Many Items in the Body or Head

Next, we show association rules with many items in the body or head. Consider the following `MINE GRAPH RULE` operator:

```
MINE GRAPH RULE SimpleAssociationRulesWithManyItems
GROUPING ON (p:Person)
DEFINING BODY AS 1..2 (p)-[:BUY]->(X:Item)
         HEAD AS (p)-[:BUY]->(Y:Item)
EXTRACTING RULES WITH SUPPORT > 0.1
                  AND CONFIDENCE > 0.1
```

The operator extracts the association rules whose body consists of up to two items, i.e., taking the form: *"People who buy X_1 or X_1 and X_2, also buy Y"*; the body items are connected to the same anchor node person along the `[:BUY]` relationship. The extracted association rules are reported in Table 2; note that the header now includes two columns for the body, respectively denoted as `BuyX1` and `BuyX2`. The rows of the table also include the rules extracted by the `SimpleAssociationRules` operator (with a missing entry for the `BuyX2` column). Three additional rows include two items in the body; among them, the rule of the first row indicates that *"People who buy Dress and Jeans, also buy Shoes"*. Note that, for the first row, $C(A) = 4$, $C(B_1) = 1$, $C(R_1) = 1$, hence the rule has support 0.25 and confidence 1.

Body		Head	Support	Confidence
ItemSetB1	ItemSetB2	ItemSetH1		
$\mathbf{BuyX_1}$	$\mathbf{BuyX_2}$	BuyY		
Dress	Jeans	Shoes	0.25	1
Dress	Shoes	Jeans	0.25	0.5
Jeans	Shoes	Dress	0.25	1
Dress	-	Jeans	0.25	0.5
Dress	-	Shoes	0.25	1
Jeans	-	Dress	0.25	0.5
Jeans	-	Shoes	0.25	0.5
Shoes	-	Dress	0.5	1
Shoes	-	Jeans	0.25	0.5
Shorts	-	T-shirt	0.25	1
T-shirt	-	Shorts	0.25	1

Table 2 Output of SimpleAssociationRulesWithManyItems

4.4 Association Rules with Where Conditions

The evaluation of association rules can be restricted by adding simple predicates over the properties of variables defined in the `MINE GRAPH RULE` operator. For instance, it is possible to select the Person nodes by setting their `City` to be "Rome". Adding a condition does not change the structure of the association rules,

which are expressed as *"People (living in Rome) who buy X, also buy Y"*.

```
MINE GRAPH RULE ConditionedAssociationRules1
GROUPING ON (p:Person) WHERE p.city = "Rome"
DEFINING BODY AS (p)-[:BUY]->(X:Item)
         HEAD AS (p)-[:BUY]->(Y:Item)
EXTRACTING RULES WITH SUPPORT > 0.1
                  AND CONFIDENCE > 0.1
```

Clearly, the restriction may reduce the cardinality of the anchor nodes, and thus produce different support and confidence for the extracted association rules. In our simple instance, the predicate is satisfied by Sofia, Chiara, and Fabio, hence $C(A) = 3$. Table 3 reports the corresponding output.

Body	Head	Support	Confidence
BuyX	BuyY		
Dress	Jeans	0.33	0.5
Dress	Shoes	0.67	1
Jeans	Dress	0.33	1
Shoes	Dress	0.67	1
Jeans	Shoes	0.33	1
Shoes	Jeans	0.33	0.5
Shorts	T-shirt	0.33	1
T-shirt	Shorts	0.33	1

Table 3 Output of ConditionedAssociationRules1

Similarly, it is possible to define simple predicates on the properties of other variables of the body and head, for example by adding a restriction on the items' prices. The next example extracts the association rules: *"People who buy X (with a price higher than 50), also buy Y (with a price lower than 50)"*:

```
MINE GRAPH RULE ConditionedAssociationRules2
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[:BUY]->(X:Item)
              AS HighPriceItem
         HEAD AS (p)-[:BUY]->(Y:Item)
              AS LowPriceItem
WHERE X.Price > 50 and Y.Price < 50
EXTRACTING RULES WITH SUPPORT > 0.1
                  AND CONFIDENCE > 0.1
```

The resulting output is shown in Table 4. Note that, for better clarity, we renamed the output columns of the body and head.

Body	Head	Support	Confidence
HighPriceItem	LowPriceItem		
Dress	Shoes	0.5	1
Jeans	Shoes	0.25	0.5

Table 4 Output of ConditionedAssociationRules2

4.5 Association Rules with `CountRel`

Next, we describe the use of the counting relations pattern, which is introduced by the `<countRel>` non-terminal symbol in the grammar (Fig. 1). The pattern allows setting a threshold on the number of times a type of relationship should be present between any two nodes, in order to be selected in the body or head of an association rule. The simplest example of application is to impose, in the body of the rule, that a given item is purchased several times. The operator should be used with a positive `mincount` number to denote the threshold (`<relType>` $>=$ `<mincount>`). An example application of the counting pattern is shown below:

```
MINE GRAPH RULE CountItemsAssociationRules
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[:BUY>=2]->(X:Item)
         HEAD AS (p)-[:BUY]->(Y:Item)
EXTRACTING RULES WITH SUPPORT > 0.1
                  AND CONFIDENCE > 0.1
```

The association rules are expressed in the sentence: *"People who buy X at least 2 times, also buy Y"*. The corresponding output is shown in Table 5.

Body	Head	Support	Confidence
BuyAtLeast2X	BuyY		
Dress	Jeans	0.25	1
Dress	Shoes	0.25	1
Shorts	T-shirt	0.25	1

Table 5 Output of CountItemsAssociationRules

Considering the association rule in the third row: *"People who buy Shorts at least 2 times, also buy T-shirt"*, which associates people who buy the product Shorts two or more times with those who also buy T-shirt; Luca is the only person satisfying both conditions. The relevant counters for this rule are: $C(A) = 4$, $C(B_3) = 1$, $C(R_3) = 1$, hence the rule has support 0.25 and confidence 1.

4.6 Association Rules with `AnyRel`

Next, we describe the use of the `AnyRel` pattern, which is introduced by the `<anyRel>` non-terminal symbol in the grammar (Fig. 1). The pattern allows us to connect a given node to any node reachable through any path (i.e., an arbitrary sequence of relationships) within a maximum length. The simplest example of application is to impose, in the body of the rule, that a given item is reached through at most one link (more complex examples are shown later). The operator should be used with

a positive `length` number to denote the path length. An example of the application is shown below:

```
MINE GRAPH RULE AnyPathAssociationRules
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[]->{1,1}(X:Item)
         HEAD AS (p)-[:BUY]->(Y:Item)
EXTRACTING RULES WITH SUPPORT > 0.1
                  AND CONFIDENCE > 0.1
```

Note that `(:Item)` nodes are reached from the `(:Person)` anchor nodes using either the `[:BUY]` or the `[:RECOMMEND]` relationships; thus, in our scenario, the sentence *"People with any link to X, also buy Y"* can also be phrased as *"People who buy or recommend X, also buy Y"*. The corresponding output is shown in Table 6.

Body	Head	Support	Confidence
AnyLinkToX	BuyY		
Dress	Jeans	0.25	0.5
Dress	Shoes	0.5	1
Jeans	Dress	0.5	0.67
Jeans	Shoes	0.5	0.67
Shoes	Dress	0.5	0.67
Shoes	Jeans	0.5	0.67
Shorts	Jeans	0.25	0.5
Shorts	T-shirt	0.25	0.5
T-shit	Shorts	0.25	1

Table 6 Output of AnyPathAssociationRules

In order to better understand it, consider the fourth association rule of the table, i.e., *"People who buy or recommend Jeans, also buy Shoes"*. As usual, $C(A) = 4$. Sofia, Chiara, and Fabio buy or recommend Jeans, thus $C(B_4) = 3$; out of them, just Sofia and Chiara buy Shoes, thus $C(R_4) = 2$. Given these counters, the rule support is $C(R_4)/C(A) = 0.5$, the rule confidence is $C(R_4)/C(B_4) = 0.67$.

By increasing the `length` parameter, the operator evaluates longer patterns, introducing more alternatives. For instance, consider the following operator:

```
MINE GRAPH RULE
AnyLongerPathAssociationRules
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[]->{1,2}(X:Item)
         HEAD AS (p)-[:BUY]->(Y:Item)
EXTRACTING RULES WITH SUPPORT > 0.1
                  AND CONFIDENCE > 0.1
```

The operator extracts the association rules taking form: *"People with any link (of length up to 2) to X, also buy Y"*. In this scenario, nodes `(:Person)` can reach nodes `(:Item)` either directly, by buying or recommending them (as the previous example), or also with two-hops paths, by using the relationship `[:FOLLOW]` connecting a node `(:Person)` to another node `(:Person)`, then connected to the item with a `[:BUY]` or `[:RECOMMEND]` relationship. In Table 7, the resulting association rules are reported.

Body	Head	Support	Confidence
AnyLinkOfLengthUpTo2ToX	BuyY		
Dress	Jeans	0.5	0.5
Dress	Shoes	0.5	0.5
Dress	Shorts	0.25	0.25
Dress	T-shirt	0.25	0.25
Jeans	Dress	0.5	0.5
Jeans	Shoes	0.25	0.25
Jeans	Shorts	0.25	0.25
Jeans	T-shirt	0.25	0.25
Shoes	Dress	0.5	0.5
Shoes	Jeans	0.5	0.5
Shoes	Shorts	0.25	0.25
Shoes	T-shirt	0.25	0.25
Shorts	Jeans	0.25	0.25
Shorts	T-shirt	0.25	0.5
T-shit	Shorts	0.25	1

Table 7 Output of AnyLongerPathAssociationRules

Consider the seventh association rule *"People with any link (of length up to 2) to Jeans also buy Shorts"*, which is not included among the extracted rules in the previous example. We still have $C(A) = 4$, but while Sofia, Chiara, and Fabio directly buy or recommend Jeans, Luca follows Sofia, who in turns buys Jeans; therefore, $C(B_7) = 4$ instead of 3. Among them, only Luca buys Shorts, so $C(R_7) = 1$. The resulting support of the rule is $C(R_7)/C(A) = 0.25$, and the confidence of the rule is $C(R_7)/C(B_7) = 0.25$.

4.7 Association Rules with Relationships' Chains

Next, we consider that `<relPattern>`s can be chained, in the body, in the head, or in both of them; as the grammar has a recursive production (`<patternTail>`), the length of the chain is not bound.

We consider a path in the body of the `MINE GRAPH RULE` operator, linking purchased items to their categories, so as to include also categories in resulting association rules. Consider the following operator:

```
MINE GRAPH RULE PathAssociationRules1
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[:BUY]->(X:Item)
                            -[:OF]->(C:Category)
         HEAD AS (p)-[:BUY]->(Y:Item)
EXTRACTING RULES WITH SUPPORT > 0.1
                   AND CONFIDENCE > 0.1
```

Note that the `(:Category)` nodes are reached from the `(:Person)` anchor nodes along a chain of relationships, which uses the `[:BUY]` and `[:OF]` relationships; the interpretation of the generated association rules is *"People who buy X of category C, also buy Y"*. The corresponding output is shown in Table 8.

Body		Head	Support	Confidence
ItemSetB1		ItemSetH1		
BuyX	OfC	BuyY		
Dress	Casual	Jeans	0.25	0.5
Dress	Casual	Shoes	0.5	1
Jeans	Casual	Dress	0.25	0.5
Shoes	Casual	Dress	0.5	1
Shoes	Sportswear	Dress	0.5	1
Shoes	Casual	Jeans	0.25	0.5
Shoes	Sportswear	Jeans	0.25	0.5
Shorts	Sportswear	T-shirt	0.25	1
T-shirt	Sportswear	Shorts	0.25	1

Table 8 Output of PathAssociationRules1

If we consider the first association rule, *"People who buy Dress of category Casual also buy Jeans"*, we note that Sofia and Chiara buy Dress of category Casual but just Sofia buys Jeans; hence, with $C(A) = 4$, we have $C(B_1) = 2$ and $C(R_1) = 1$. Given these the counters, the rule support is $C(R_1)/C(A) = 0.25$, the rule confidence is $C(R_1)/C(B_1) = 0.5$.

4.8 Association Rules with Ignore

The `IGNORE` construct designates a list of variables -i.e., nodes of the pattern- that shall not be included in the body or head of the rule. We denote as *visible variables* those appearing in the operator but not in the ignore clause; only visible variables will appear in the body and head of association rules.

For example, consider the following `MINE GRAPH RULE` operator:

```
MINE GRAPH RULE IgnoreAssociationRules
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[:BUY]->(X:Item)
                            -[:OF]->(CX:Category)
         HEAD AS (p)-[:BUY]->(Y:Item)
                            -[:OF]->(CY:Category)
IGNORE X,Y
EXTRACTING RULES WITH SUPPORT > 0.1
                  AND CONFIDENCE > 0.1
```

Extracted rules take the form *"People who buy items of category CX, also buy items of category CY"*. The corresponding output is shown in Table 9.

Body	Head	Support	Confidence
ItemSetB1	ItemSetH1		
BuyItemsOfCategoryCX	BuyItemsOfCategoryCY		
Casual	Sportswear	0.5	0.67
Sportswear	Casual	0.5	0.67

Table 9 Output of IgnoreAssociationRules

Using `IGNORE` adds a second aggregation level to the normal procedure: in particular, after having extracted the anchor nodes for body and head, they are further grouped only with respect to the *visible variables*, thus hiding the information of the *ignored variables*. Consider rule 1, *"People who buy items of category Casual, also buy items of category Sportswear"*. Sofia, Chiara, and Fabio buy at least one item in the category Casual, resulting in $C(B_1) = 3$, but Sofia and Chiara also buy items in the category Sportwear, whereas Fabio does not buy any item in that category. As Fabio is not paired to both the body and the head, Fabio is not a valid anchor for the association rule, and $C(R_1) = 2$. As usual, with $C(A) = 4$ the rule support is $C(R_1)/C(A) = 0.5$, the rule confidence is $C(R_1)/C(B_1) = 0.67$.

4.9 Association Rules with Conjunctions

Next, we consider that both the body and the head can have conjunctive expressions, with an arbitrary number of conjuncts. We consider the body of the `MINE GRAPH RULE` operator as a conjunction of two different `<pattern>`s. Consider the following operator:

```
MINE GRAPH RULE ComplexBodyAssociationRules
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[:FOLLOW]->(X:Person),
                              (p)-[:BUY]->(Y:Item)
         HEAD AS (p)-[:BUY]->(Z:Item)
EXTRACTING RULES WITH SUPPORT >  0.1
                   AND CONFIDENCE > 0.1
```

The association rules are interpreted in this way: *"People who follow X and buy Y, also buy Z"*. The corresponding output is shown in Table 10. Considering the first association rule, *"People who follow Sofia and buy Dress, also buy Shoes"*, we note that Chiara is the only one who both follows Sofia and buys Dress, and she also buys Shoes; hence, with $C(A) = 4$, we have $C(B_1) = 1$ and $C(R_1) = 1$ and with these counters, the rule support is $C(R_1)/C(A) = 0.25$ and the rule confidence is $C(R_1)/C(B_1) = 1$.

4.10 Other Complex Association Rules

4.10.1 First Example.

We consider a body `<pattern>`, with two conjuncts both consisting of chains of relationships. The following operator extracts rules whose interpretation is *"People who follow more than two people who recommend products of category CX and buy products of category CY, also buy Z"*. The corresponding output is shown in Table 11. Considering the first association rule, "People who follow more than two persons who recommend

Body		Head	Support	Confidence
ItemSetB1	ItemSetB2	ItemSetH1		
FollowX	BuyY	BuyZ		
Sofia	Dress	Shoes	0.25	1
Sofia	Shoes	Dress	0.25	1
Sofia	Shorts	T-shirt	0.25	1
Sofia	T-shirt	Shorts	0.25	1
Chiara	Dress	Jeans	0.25	1
Chiara	Dress	Shoes	0.25	1
Chiara	Jeans	Dress	0.25	1
Chiara	Jeans	Shoes	0.25	1
Chiara	Shoes	Dress	0.25	1
Chiara	Shoes	Jeans	0.25	1
Chiara	Shorts	T-shirt	0.25	1
Chiara	T-shirt	Shorts	0.25	1
Fabio	Shorts	T-shirt	0.25	1
Fabio	T-shirt	Shorts	0.25	1

Table 10 Output of ComplexBodyAssociationRules

an item of category Sportswear and also buy items of category Sportswear, also buy T-shirt"; we note that Luca is the only one who both follows more than two persons -Chiara and Fabio- recommending Shoes and Shorts of category Sportswear, and also buys items of category Sportswear; therefore, with $C(A) = 4$, we have $C(B_1) = 1$ and $C(R_1) = 1$ and, with these counters, the rule support is $C(R_1)/C(A) = 0.25$ and the rule confidence is $C(R_1)/C(B_1) = 1$.

```
MINE GRAPH RULE MoreComplexBodyAssociationRules
GROUPING ON (p:Person)
DEFINING BODY AS (p)-[:FOLLOW>=3]->(PX:Person)
                             -[:RECOMMEND]->(X:Item)
                             -[:OF]->(CX:Category),
                  (p)-[:BUY]->(Y:Item)
                             -[:OF]->(CY:Category)
         HEAD AS (p)-[:BUY]->(Z:Item)
IGNORE PX, X, Y
EXTRACTING RULES WITH SUPPORT > 0.1
                   AND CONFIDENCE > 0.1
```

Body		Head	Support	Confidence
ItemSetB1	ItemsSetB2	ItemSetH1		
FollowPersonRecommendItemOfCX	BuyItemOfCY	BuyZ		
Sportswear	Sportswear	T-shirt	0.25	1
Sportswear	Sportswear	Shorts	0.25	1
Casual	Sportswear	T-shirt	0.25	1
Casual	Sportswear	Shorts	0.25	1

Table 11 Output of MoreComplexBodyAssociationRules

4.10.2 Second Example.

Lastly, we consider a complex structure of the body `<pattern>` with `WHERE` conditions on both anchors and nodes. The following operator extracts rules whose interpretation is *"People from Rome who follow Chiara, who recommends items of category CX, also buy items of the same category"*. The corresponding output is shown in Table 12. Considering the first association rule, *"People from Rome who follow Chiara who recommends items of category Casual, also buy Jeans, which are of the same category Casual"*, we note that there is only one person, Sofia, who satisfies all these constraints, but also Luca follows Chiara recommending Casual items; hence, with $C(A) = 3$, we have $C(B_1) = 2$ and $C(R_1) = 1$ and, with these counters, the rule support is $C(R_1)/C(A) = 0.33$ and the rule confidence is $C(R_1)/C(B_1) = 0.5$.

```
MINE GRAPH RULE VerifyInfluencerEffectivenessInRome
GROUPING ON (p:Person) WHERE p.city = "Rome"
DEFINING BODY AS (p)-[:FOLLOW]->(PX:Person)
                             -[:RECOMMEND]->(X:Item)
                             -[:OF]->(CX:Category)
         HEAD AS (p)-[:BUY]->(Y:Item)
                             -[:OF]->(CY:Category)
WHERE PX.name = "Chiara" AND CX.name = CY.name
IGNORE PX, X
EXTRACTING RULES WITH SUPPORT > 0.1
                   AND CONFIDENCE > 0.1
```

Body	Head		Support	Confidence
ItemSetB1	ItemSetH1			
FollowChiaraRecommendCX	BuyItem	OfCY		
Casual	Jeans	Casual	0.33	0.5
Casual	Dress	Casual	0.33	0.5
Casual	Shoes	Casual	0.33	0.5
Sportswear	Shoes	Sportswear	0.33	0.5
Sportswear	T-shirt	Sportswear	0.33	0.5
Sportswear	Shorts	Sportswear	0.33	0.5

Table 12 Output of VerifyInfluencerEffectivenessInRome

5 Implementation

Since the GQL standard was released only in April 2024, mature products that fully adopt the standard are in progress. We chose to base our implementation on Cypher [28], the query language of Neo4j, due to its close affinity to GQL. We selected the Neo4j Graph database as a development platform since it provides an easily expandable architecture. Still, our proposed implementation does not strictly depend on any specific Neo4j component or Cypher construct. Implementations for other graph databases and frameworks, like Memgraph, JanusGraph, or Apache TinkerPop, can be developed with no particular efforts.

Our implementation of the `MINE GRAPH RULE` operator has been developed in Java as a user-defined procedure; it is available as a Neo4j plugin like other add-ons of the Awesome Procedures on Cypher (APOC) [8] library. Libraries providing the data structures exploited in the current implementation are also available for most of the other main graph database systems.

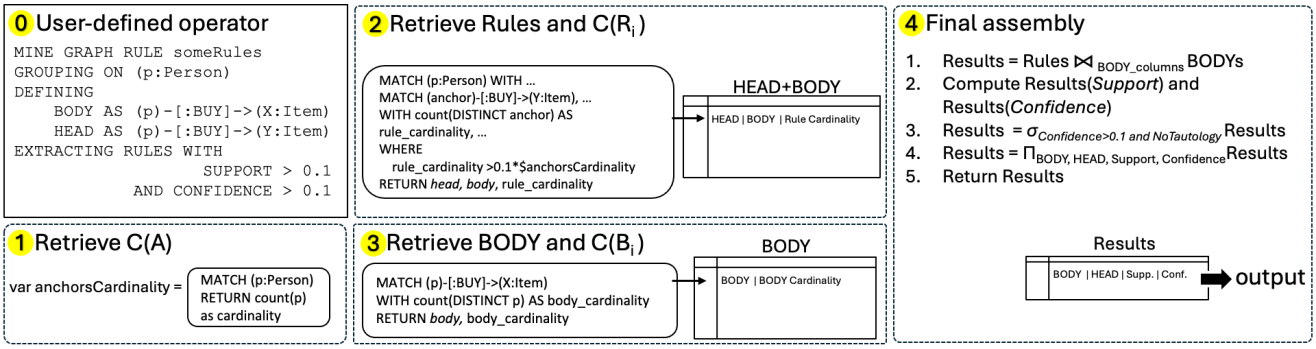

Fig. 4 Phases of the `MINE GRAPH RULE` algorithm applied to a simple user-defined operator. `Phase(1)` counts the anchor nodes with the label 'Person' $C(A)$. In `Phase(2)`, a query generating head and body for the operator is issued, and the query results, which satisfy the minimum support threshold (expressed using absolute counts), are entered into a table whose entries include the HEAD, the BODY, and the rule cardinality $C(R_i)$ In `Phase(3)`, a query generating just the body for the operator is issued and the query results are entered into a table whose entries include the BODY and the BODY cardinality $C(B_i)$. In the final `Phase(4)` the output table is produced along a progression of five steps: first, the two tables produced by phases 3 and 4 are joined on the BODY columns; then, support and confidence for each record of the resulting table are computed; then, rows that create tautologies or do not satisfy the minimum confidence threshold are removed; then, the result is projected upon the chosen schema; finally, resulting rows are streamed in output.

Once the plugin is installed, the procedure can be invoked by its name "`mineGraphRule`" in the Neo4j Cypher shell; it supports nine mandatory parameters that map one-to-one to the variable elements of the operator contained in the tuple $(a, A, W_a, H, B, W, I, s, c)$: the anchor's variable a, the anchor's label A, the conditions W_a on the anchor, the map structure H for the `HEAD` itemSet, the map structure B for the `BODY` itemSet, the list of predicates W for the `WHERE` clause, the list of `IGNORE`d variables I, the minimum support threshold s, and the minimum confidence threshold c. A detailed specification of the parameters' format can be found in the project's repository [33]. Collectively, these nine parameters provide a semi-structured version of the `MINE GRAPH RULE` operator. In our implementation, we associated each node with a system-generated identifier, to ensure the uniqueness of each node.

Once executed, the procedure outputs a stream of records (each representing one association rule) with a fixed format, consisting of four fields:

- `body`: a map of key-value pairs for the `BODY` columns. The keys are the column names , whereas the values are the identifiers (i.e., properties) of the nodes matched by the specified patterns;
- `head`: a map of key-value pairs for the `HEAD` columns, with the same format of the `BODY`;
- `support`: the support value in double floating-point precision;
- `confidence`: the confidence value in double floating-point precision.

By default, the APOC procedure uses each internal node identifier to match the nodes and generate the output table for both the `BODY` and `HEAD` columns. However, with an optional input parameter, the user can provide a list of identifying attributes as key-value pairs (<NodeLabel>: <IdentifyingPropertyList>) specifying, for each or some node label, which property should be used instead of the internal node identifier.

The result can be transformed with the Cypher `YIELD` sub-clause [43], which can select any column of the output record and apply any supported Cypher or APOC operation on them, such as reordering, regrouping, or string concatenation.

We next describe our algorithm by first considering a base case, not including conjunctions or multiple items in the BODY and in the HEAD. Then, we consider the general case.

5.1 Algorithm for Simple Cases

In the translation, we use a syntax-directed approach in which, for every production of the grammar, we generate an appropriate Cypher expression. Therefore, chains are directly translated as Cypher `MATCH` blocks targeting alternatively: a simple relationship, a relationship filtered based on the number of edges, or a bounded path with free-type relationships. In this way, our `MINE GRAPH RULE` language, which includes a rich collection of orthogonal features, can be expressed in a compact Cypher query, whose execution takes advantage of the optimization capabilities of graph database engines. In this simple case, we only need the following queries:

- a query for retrieving the anchors' cardinality $C(A)$ (see Section 3);
- a query for retrieving HEAD, BODY, and rule's cardinality $C(R_i)$. Since this is the most selective query,

we obtain the rules' confidence at an early stage, so that an optimized execution can discard all rules below the confidence threshold;
– for the remaining rules, a query for retrieving BODY and body's cardinality $C(B_i)$. At this point, the rules' support can be computed and the optimized query execution can discard all rules below the support threshold.

The first query follows a simple structure and produces a variable counting the number of anchors. The other two queries have a similar structure, which is described as follows:

1. `MATCH` subquery for anchors' selection based on labels and on predicates that are specific to anchors' properties;
2. `MATCH` on all BODY (and possibly HEAD) patterns in conjunction;
3. optional `WHERE` predicates, when operators include conditions on the variables;
4. `WITH` clause to group by anchors and visible variables, to compute the rule cardinality;
5. `WHERE` predicate to guarantee minimum support;
6. `RETURN` clause with the final names returned to the user at the end of the procedure.

Cypher queries produced for bodies only include BODY's patterns, whereas queries produced for rules also include elements from heads.

The actual association rules, with their support and confidence, are produced after a final assembly of the results of the three described Cypher queries. Fig. 4 shows the complete flow of execution of the graph association rule mining algorithm when applied to a simple operator.

5.2 Algorithm for General Cases

General cases include conjunctions and multiple patterns in the BODY and/or in the HEAD. Our algorithm is inspired by the fast Apriori algorithm implementation [10], where the mining starts by exploring the smaller (i.e., most common) sets of frequent items and combines them into bigger sets, until the minimum support threshold is met. We start by considering the operators with one pattern in the BODY and one in the HEAD and then consider operators with more patterns, thus, performing a body-head pair expansion. At each step, exactly one pattern is added either to the BODY or to the HEAD, up to the maximum number of patterns specified in the input operator. The body-head pairs are organized in a directed acyclic graph (DAG) from the simplest to the most complete one, as exemplified in Fig. 5, Phase (2), for a particular example where the BODY presents two patterns in conjunction, the first ranging from 1 to 3 and the second ranging from 1 to 2.

Continuing our analogy with the Apriori algorithm, the objective of organizing the evaluation with a DAG of pairs is to enable the early exclusion of pairs when their support and confidence does not meet the minimum conditions set by the users. As usual, our antimonotonicity property only refers to support. (see Section 5.3 for the related theorem and demonstration). Thus, pairs are considered in an order created by a breadth-first traversal of the DAG and, as soon as one pair does not meet the support threshold, that pair, together with all its descendants, is removed from the DAG. The generation of queries corresponding to each given pair is performed with the mechanism defined for simple input operators, as illustrated in Fig. 4, Phases (3) and (4).

Note that tables HEAD+BODY and BODY accumulate the instances for all the pairs. As discussed in Section 5.1, a final assembling step produces all the association rules with their HEAD, BODY, support, and confidence.

5.3 Demonstration of the Antimonotonicity Property of the DAG

To demonstrate the antimonotonicity property of the `BODY-HEAD` pairs of the DAG (used in the algorithm described in Section 5.2), we first introduce the DAG of BODY-HEAD pairs, where N is the set of its nodes and A is the set of its arcs, then we define and prove our notion of antimonotonicity.

Definition 1 Each node $n \in N$ represents a fixed `BODY-HEAD` pair of `MINE GRAPH RULE` applied to an instance G of a property graph database. Each pair includes a finite set of joint patterns P_H of the HEAD and a finite set of joint patterns P_B of the BODY. For each pattern $p_{Xi} \in P_H \cup P_B$, its *arity* (i.e., the number of items in each conjunct) is described by the function $k(n, p_{Xi}) : (N, P_B \cup P_H) \rightarrow \mathbb{N}^+$.

Definition 2 Each arc $a \in A : u \xrightarrow{a} v$ between nodes u and v denotes that the function $k(v, *)$ has the same output values of $k(u, *)$ for any pattern p_{Xi}, except for one specific pattern, denoted as $\widetilde{p_{Xi}}$, for which we impose $k(v, \widetilde{p_{Xi}}) = k(u, \widetilde{p_{Xi}}) + 1$ during the construction of the DAG.

Theorem 1 *For any node u in the DAG and all its `BODY-HEAD` pair descendants v, the support count $S(v)$ is at most equal to the support count $S(u)$: $S(v) \leq S(u)$.*

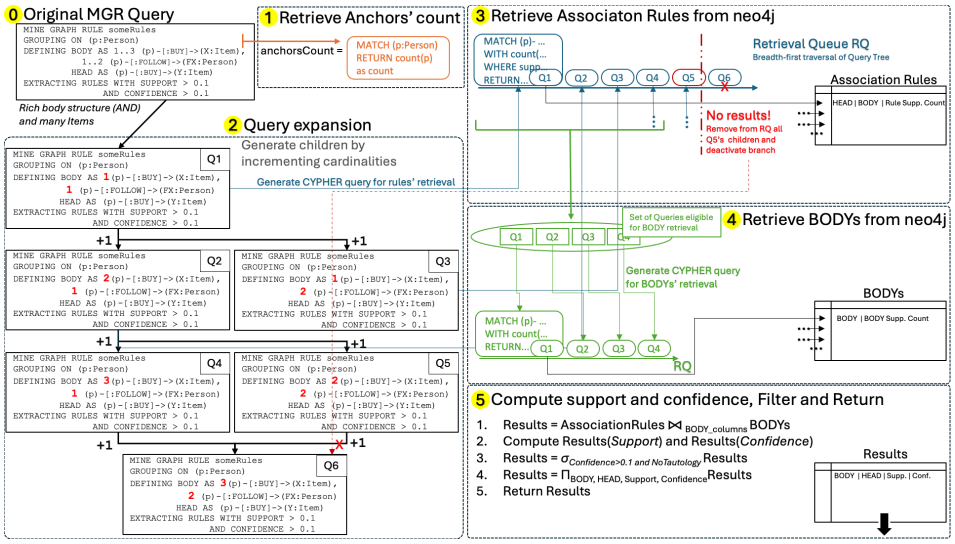

Fig. 5 Phases of the `MINE GRAPH RULE` algorithm applied to an operator with two patterns in conjunction in the body, each with many items, ranging respectively from 1 to 3 and 1 to 2. This specific operator has been chosen to explain how queries are expanded, queued, and possibly removed from the queue. `Phase(1)` counts the anchor nodes with the label 'Person'. `Phase(2)` generates various nonredundant *body-head pairs*, named P1..P6, arranged in a directed acyclic graph (DAG), each with a fixed number of items in the body's conjuncts; for each pair, the number of items of the body's conjuncts are highlighted (in red in the figure) - while descending along the DAG levels, item numbers increase by 1, along the progression: [P1:(1,1)]; [P2:(2,1),P3:(1,2)]; [P4:(3,1),P5:(2,2)]; [P6(3,2)]. In `Phase(3)`, the pairs generated at Phase (2) are queued, by performing a breadth-first traversal of the DAG; thus, the queue PQ: P1..P6 is generated. For each pair, a query producing head and body is issued, and the query results are entered into a table whose entries include the HEAD, the BODY, and the rule cardinality. As explained in Fig. 4, Phase (2), only the rules that satisfy the minimum support threshold are selected. At this stage, it is possible to remove some pairs from the queue; if we assume that no rule extracted for pair P5 satisfies the constraint on the minimum support, then P5 can be removed from the queue together with all the successors in the DAG (in this case, P6). In `Phase(4)`, for each remaining pair P1..P4, a query producing just the body for that pair is generated, and the query results are entered into a table whose entries include the BODY and the BODY cardinality. The final `Phase(5)` occurs in the same way as described in Fig. 4, Phase (4).

Proof We prove the theorem by contradiction. Suppose that for any pair u such that $u \to v$, it holds that $S(v) > S(u)$. Then, the `BODY-HEAD` pair of v is less selective than the pair in u, which is excluded by Definition 2, imposing by construction an increased arity. We recall from the GQL semantics that any additional path – as an added item in the result – increases the selectivity of the `MATCH` expression. Thus, the initial assumption $S(v) > S(u)$ is false. □

6 Evaluation

We assessed the performance of the `MINE GRAPH RULE` operator by conducting a series of experiments over a combination of synthetic datasets and query complexities of the operator. Furthermore, to validate the results obtained from the operator, we also tested it on real-world datasets.

Configuration We ran the experiments on our dedicated server machine with a 56-core Intel E5-2660 v4 CPU and 384 GB of RAM. We deployed a Neo4j database instance from its `5.15.0-community-bullseye` Docker image. To avoid memory bottlenecks, we configured the database with the following settings:

```
server.memory.heap.initial_size=230G
server.memory.heap.max_size=230G
dbms.memory.pagecache.size=200G
```

6.1 Synthetic Databases Generation

We generated three types of synthetic datasets, all adhering to the schema of the running example:

- The first one has a *Uniform* distribution of all the edges among random nodes of the graph, ensuring that relationships are evenly spread.
- The second one, named *Coarse Scale-free*, uses the power-law distribution for the `BUY` relationship, represents the real-case scenario of an online shop; all the other relationships are generated with a uniform distribution among all nodes.

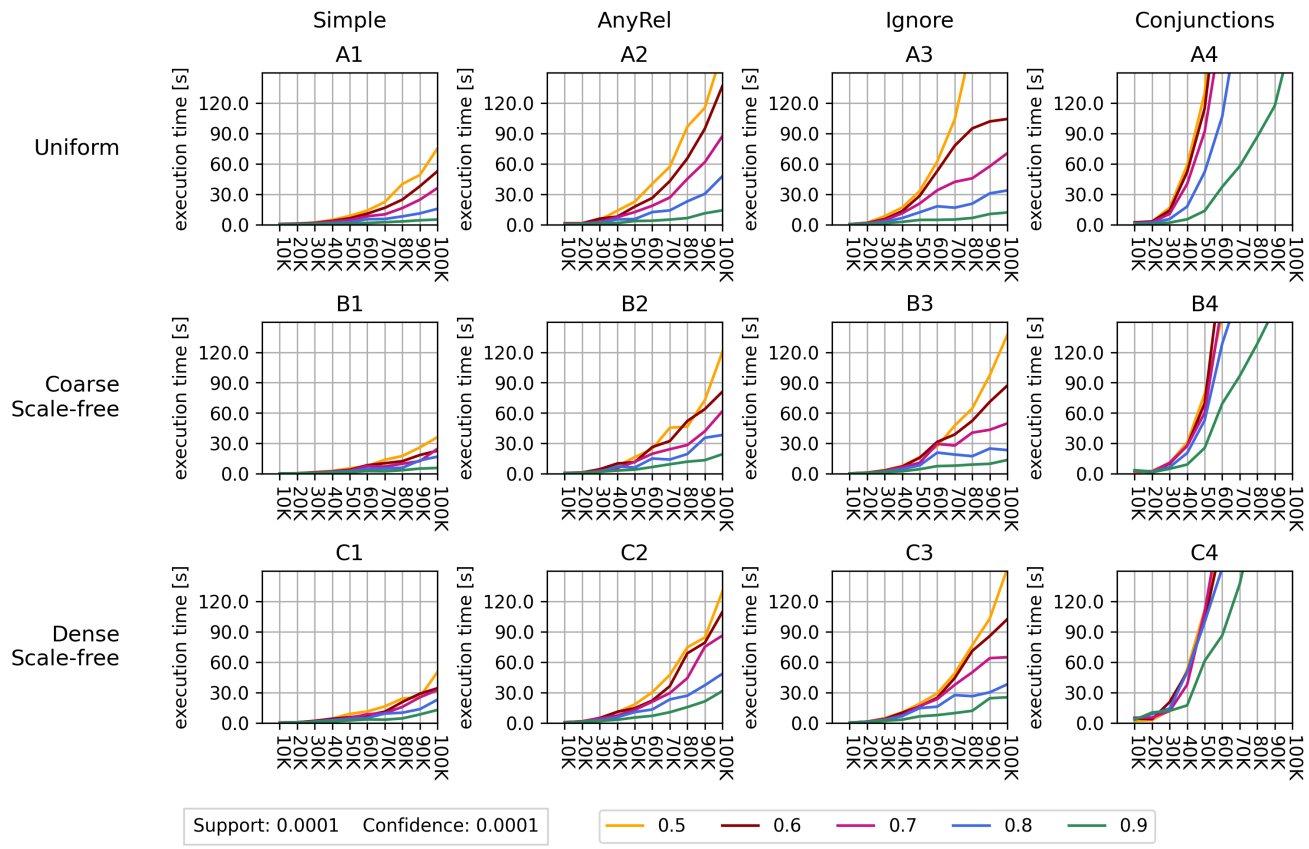

Fig. 6 Performance analysis for synthetic datasets. Each graph reports the execution time required for executing four operators with ingresing complexity on the used patterns (respectively named Simple, AnyRel, Ignore and Many Items) over graphs with different syntethic edge generations (uniform, coarse scale-free, dense scale free); in the graph generation, we vary the total dimension of the graph (up to 100K nodes) and use five different ratios (ranging from 0.5 to 0.9) of the number of anchor nodes over the total nodes.

– The third one, named *Dense scale-free*, also uses a power-law distribution for the `BUY` relationship, bwith a higher density of relationships.

For each type of synthetic dataset, we generated multiple versions by varying the total number of nodes of the graph to test how the performance scales with respect to the dimensions of the graph. In addition, for each graph instance, we varied the ratio of anchor nodes (in our example, nodes with label `Person`), which directly impacts the absolute count of `BUY` relationships within a graph. In *Uniform* graphs, the total count of `BUY` is proportional to the product of the number of `Person` nodes and the number of `Item` nodes, which is maximum when the graph is evenly split between `Persons` and `Items`. Therefore, increasing the ratio over 0.5 would produce less `BUY` relationships. For the *Scale-free* distribution, which mimics the distribution of items' purchases in real life [38,26], graphs have a fixed smaller portion of `Items` that are way more likely to be bought, and increasing the percentage of `Persons` results in a higher number of `BUY` relationships; therefore, here the ratio of anchor nodes has a nonlinear relationship with the total count of `BUY` edges.

6.2 Experimental Results

We evaluated the scalability of our operator in multiple settings.

General Scalability To assess the general scalability of the operator, we evaluated the execution time of four different expressions: *Simple*, *AnyRel*, *Ignore*, and *Conjunctions* referencing the examples respectively reported in Sections 4.2, 4.6, 4.8 and 4.9. In Fig. 6, we compared their performance with fixed values of support and confidence (set to 0.0001) on different synthetic graphs, varying not only the distribution of relationship `BUY`, but also the graph dimension (from 10K nodes to 100K nodes) and the ratios between anchor and total number nodes (from 0.5 to 0.9). In general, as the complexity of the operators grow, we obtain higher

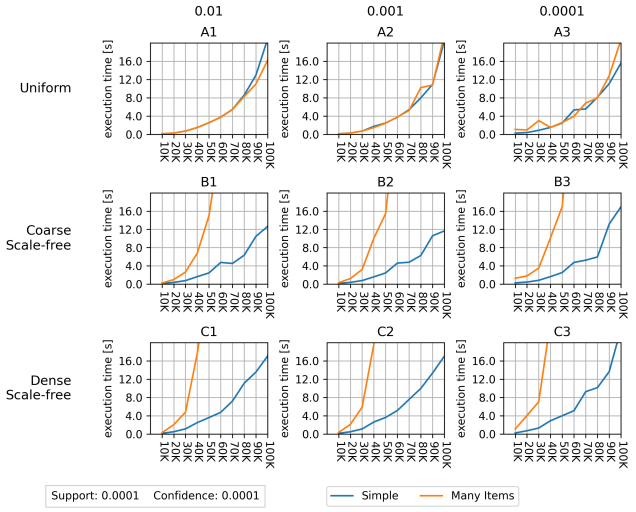

Fig. 7 Impact of support in the various configurations. Each graph reports the execution time required by *Simple* and *Many Items* operators over graphs generated with increasing numbers of nodes, by fixing the anchor nodes over the total nodes ratio (equal to 0.8). Each column corresponds to increasing values of support (respectively 0.01, 0.001, and 0.0001); confidence is set at 0.0001.

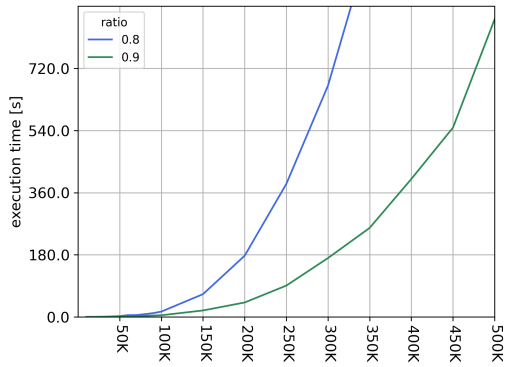

Fig. 8 High Volume Scalability. For the *Simple* operator, we consider a high the number of nodes (up to 500K) and diferent ratios (0.8 vs 0.9) of item nodes over anchor nodes.

execution times while moving from Colimn 1 to Column4, as expected.

With *Uniform* graphs, very few assovciation ruiles are extracted, as the topology does not encourage anomalous associations. In *Scale-free* graphs, having some `Items` with higher probability to be bought results in extracting many association rules, thus activating the branch of the algorithm that would perform additional queries for conjuctive branches, ultimately making the Conjunction query slower (see Section 5.2). For instance, in B3 at 50K nodes, *Simple* extracts 2,630 rules while *Conjuctions* extracts 10,697 rules. Thus, execution times of "Conjunctions" (cases B4/C4) are much higher tham execution times of "Simple" (cases B1/C1); performance does not seem to be too affected by the difference between dense and coarse scale-free relationship generation.

Impact of Support Selection To evaluate the effect of the support on the performance of the operator, in Fig. 7 we focused on two expressions of the operator, *Simple* and *Many Items* (respectively reported in Sections 4.2 and 4.3), testing them with fixed values of confidence (0.0001) on synthetic graphs with different distributions and increasingly large number of nodes (from 10K to 100K). In the uniform case, as no association rules are produced, performances of Simple and Many Items are identical; instead, they differ in the scale-free case, with worse performance in the *Many Items* case for higher density and for reduced support costraints (as more association rules are produced in output).

High Volume Scalability Lastly, focusing on the operator *Simple* (see Section 4.2), we evaluated the high volume scalability with a fixed value for support and confidence (both at 0.0001) and an increasing number of nodes (up to 500K). In Fig. 8, we compared two different values of ratios of anchor nodes over item nodes (0.8 in blue and 0.9 in green), showing that performances worsen when anchor nodes increase over item nodes.

6.3 Real-World Datasets

We evaluated the results of our operator from two real-world datasets.

Spotify Dataset [53] This dataset comprises information from over 6,000 Spotify playlists, including details such as the music genre of each playlist, the user who created it, the list of songs, the artists of the songs, and their respective genres. The resulting graph contains over 380,000 nodes and over 923,000 relationships. Consider the following `MINE GRAPH RULE` operator:

```
MINE GRAPH RULE ArtistPlaylist
GROUPING ON (p:Playlist)
DEFINING BODY AS (p)-[:With]->(s1:Song)
                             -[:SungBy]->(X:Artist)
         HEAD AS (p)-[:With]->(s2:Song)
                             -[:SungBy]->(Y:Artist)
WHERE X.popularity > 70 and Y.popularity > 70
IGNORE s1, s2
EXTRACTING RULES WITH SUPPORT > 0.001
                  AND CONFIDENCE > 0.001
```

The interpretation of the generated association rules is *"Playlists with songs sung by X (with over 70 popularity), have also songs sung by Y (with over 70 popularity)"* and with these thresholds values for support and confidence the operator extracts more than 60,000

rules. Table 13 reports the rules with the 10 highest support values. The top positions are mainly occupied by paired rules (i.e., where the artists are swapped between the head and body). Based on the mathematical definition of support, these paired rules must have the same support value. However, their confidence values can vary significantly, which suggests interesting implications. For instance, *"Playlists with songs sung by Harry Styles, have also songs sung by Taylor Swift"* has support of 0.0181 and confidence of 0.491, while *"Playlists with songs sung by Taylor Swift, have also songs sung by Harry Styles"* has the same support of 0.0181 but confidence of 0.348, suggesting that Harry Styles fans enjoy Taylor Swift songs more than her fans enjoy his songs. This can be interpreted as a greater attention to Taylor's Swift's personal history, which is typically in the text of her songs.

Body	Head	Support	Confidence
PlaylistWithSongSungByX	PlaylistWithSongSungByY		
Kanye West	Drake	0.0216	0.475
Drake	Kanye West	0.0216	0.446
Kendrick Lamar	Drake	0.0204	0.582
Drake	Kendrick Lamar	0.0204	0.422
21 Savage	Drake	0.0183	0.816
Drake	21 Savage	0.0183	0.377
Harry Styles	Taylor Swift	0.0181	0.491
Taylor Swift	Harry Styles	0.0181	0.348
Kendrick Lamar	Kanye West	0.0176	0.502
Drake	Future	0.0176	0.364

Table 13 Output of ArtistPlaylist rules with highest 10 values of support

ArXiv Dataset [9] This dataset comprises information from scientific papers, including details on both their authors and their categories/macro-categories. The resulting graph contains over 5 million nodes and over 15 million relationships. Consider the following `MINE GRAPH RULE` operator:

```
MINE GRAPH RULE AuthorCategory
GROUPING ON (a:Author)
DEFINING BODY AS (a)-[:Publish]->(ar1:Article)
                        -[:Labelled]->(CX:Category)
                        -[:Of]->(m1:Macrocategory)
                        AS "ComputerScience"
         HEAD AS (a)-[:Publish]->(ar2:Article)
                        -[:Labelled]->(CY:Category)
                        -[:Of]->(m2:Macrocategory)
                        AS "Physics"
WHERE m1.name = "Computer Science"
        and m2.name = "Physics"
IGNORE ar1, ar2, m1, m2
EXTRACTING RULES WITH SUPPORT > 0.001
                  AND CONFIDENCE > 0.001
```

The generated association rules can be interpreted as *"Authors who publish articles labelled CX of Computer Science, also publish articles labelled CY of Physics"*. In this case, we opted for simpler column names (using the aliases), and named columns just *ComputerScience* and *Physics*). The operator extracted over 1,000 rules using these threshold values for support and confidence, demonstrating its capability to handle millions of nodes, and the rules with the highest 10 values of support are listed in Table 14. For instance, the pair *Machine Learning, Quantum Physics* has support of 0.0042 and confidence of 0.039.

7 Brief Comparison with Path Association Rules Mining (PARM)

Among the graph pattern mining algorithms in the existing literature, PARM, by Sasaki and Karras [49] (Nov. 2024) exploits some of the characteristics of property graphs to extract association rules. This section provides a deep analysis of the similarities and differences between PARM and our approach, highlighting their respective strengths.

PARM explores association rule mining by leveraging paths within the graph; path patterns are classified either as *simple paths*, i.e. paths of a given length consisting of alternating node labels (attributes) and edge types, or *reachability path patterns*, where only the labels of the first and last nodes are specified, and these nodes are connected by a path of edges of the same type, constrained to a maximum length. PARM defines a path as matching a path pattern when it is composed exactly of the alternating sequence of nodes and edges specified in the pattern; it also states that a node matches a path pattern if it originates a path that matches to the same pattern. PARM also includes the dominance between pairs of path patterns, stating that one pattern dominates another if it is longer and matches the other pattern along its entire length. These premises allow the definition of Path Association Rules (PAR) as pairs of path patterns, as well as various notions of support and confidence. In particular, the authors focus on association rules between paths with support above a given threshold, shorter than a maximum length, and such that neither of them dominates the other. Rule extraction is performed by the PIONEER algorithm, which takes advantage of anti-monotonicity to prune infrequent patterns and of auxiliary data structures to store intermediate results; several extensions of the base version of the method are discussed, for approximation and parallelization.

In comparison, while `MINE GRAPH RULE` provides syntax and semantics for declaratively describing association rules, PARM extracts association rules from the graph in a bottom-up manner. A few aspects are similar, e.g., the use of path expressions; `MINE GRAPH RULE` does not support reachability paths but it supports

Body	Head	Support	Confidence
ComputerScience	Physics		
Machine Learning	Material Science	0.0042	0.040
Machine Learning	Quantum Physics	0.0042	0.039
Machine Learning	Computational Physics	0.0032	0.030
Computer Vision and Pattern Recognition	Materials Science	0.0031	0.040
Artificial Intelligence	Materials Science	0.0029	0.040
Machine Learning	Optics	0.0029	0.027
Machine Learning	Mesoscale and Nanoscale Physics	0.0029	0.027
Artificial Intelligence	Quantum Physics	0.0028	0.038
Social and Information Networks	Physics and Society	0.0027	0.196
Machine Learning	Physics and Society	0.0027	0.025

Table 14 Output of AuthorCategory rules with highest 10 values of support

the orthogonal and recursive construction of patterns with much higher expressive power, as these include aggregation, alternative paths, arbitrary composition of itemSets, conjunctive expressions of paths, and arbitrary predicates over any semantic feature supported by property graphs (e.g., arbitrary labels and properties of nodes and edges) – thanks to the richness of GQL. Algorithms are not comparable, as PIONEER is implemented in C++ over structures optimized for the purpose.

8 Discussion and Conclusion

After our progressive illustration of the expressive power of the operator in Section 4, we summarize here the benefits of our approach in terms of readability.

- Our language uses clauses mimicking GQL expressions, employing standard GQL variables and predicates over properties and labels of entities and relationships; as such, it exploits the richness of the property graph data model.
- Resulting association rules are extracted as rows of (structured) tables, associating the Body and Head itemSets to their support and confidence. These can be inspected by any non-technical user in the form of a CSV file, e.g., by using spreadsheets.
- We also developed a recursive mechanism, driven by a grammar, to generate semantically rich column names, e.g., for Body: **FollowPersonRecommendItemOfCX** and for Head **BuyItemOfCY** (see Table 11); when names are considered as too complex, they can be renamed using alias clauses (see Table 14).

Some recent papers have attempted to translate complex formal expressions into readable examples, such as *two users are likely colleagues if they follow the same organization and have over k friends in common* (see Example 1 in [23]) or *if a television station v is part of a company employing a CEO, then it is also part of a company employing a professor* (see Example 'Nell' in [49]). However, these rules are not equipped with their support and confidence, and no indication is given about how to extract these specific rules from the large collection of extracted rules.

Our approach exploits the use of a GQL engine; in particular, the fragment of GQL that we use is also compatible with Cypher [28], a widely used graph database language. Therefore, in our implementation, we map `MINE GRAPH RULE` patterns to Cypher and execute them on a Neo4j engine; we use the Apriori approach at a set-oriented level, and exploit the query optimizer of the graph database engine, without resorting to schema-specific tunings or heuristics. Although our prototype implementation is specific to Neo4j, the syntax and semantics of the operator are general, and the principles followed in the implementation can be applied to any graph database.

In this work, our focus has been on defining the syntax and semantics of the `MINE GRAPH RULE` operator; we deliberately left out the optimization of the execution over a data mining engine. Clearly, the *a priori* knowledge of the queries makes them amenable to efficient optimizations. For what concerns query expansion, we could design a special index that expands one group if and only if it has enough support, mimicking with a physical structure an optimized breadth-first DAG traversal. We intend to dedicate follow-up work to deepening query optimization for property graph databases, using suitable physical database ingredients, and performance-oriented benchmarks.

In the current design, `MINE GRAPH RULE` is a declarative, top-down operator; another interesting direction for future work is to embed the operator into higher-level abstractions, so as to give more space to bottom-up identification of most frequent patterns.

In conclusion, `MINE GRAPH RULE` is a significant step forward along the directions discussed in [11], to which we recently contributed in [32,19,18].

Resources

The source code of the implementation of the mineGraphRule plugin for Neo4j, examples of usage, and the scripts to generate the evaluation datasets are available on GitHub [33].

Acknowledgements This paper is supported by the FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, Investment 1.3, Line on Artificial Intelligence).

References

1. Apyori. `https://github.com/ymoch/apyori`, 2019.
2. Db-engines ranking of graph dbms. `https://db-engines.com/en/ranking/graph+dbms`, 2024. Last accessed online: January 22nd, 2025.
3. 39075, I. Information technology - database languages - gql - fist edition 2024-04, 2024.
4. Abiteboul, S., and Bidoit, N. Non first normal form relations to represent hierarchically organized data. Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (1984), 191–200.
5. Aggarwal, C. C. An introduction to social network data analytics. Springer, 2011.
6. Agrawal, R., Imieliński, T., and Swami, A. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data (1993), pp. 207–216.
7. Agrawal, R., Srikant, R., et al. Fast algorithms for mining association rules. In Proc. 20th int. conf. very large data bases, VLDB (1994), vol. 1215, Santiago, pp. 487–499.
8. APOC Library. Awesome Procedures for Neo4j 5.18.x. `https://github.com/neo4j/apoc`, 2024. Last accessed online: January 22nd, 2025.
9. arXiv.org submitters. arxiv dataset, 2024.
10. Bodon, F. A fast APRIORI implementation. In FIMI (2003), vol. 3, Citeseer, p. 63.
11. Bonifati, A., Ozsu, M. T., Tian, Y., Voigt, H., Yu, W., and Zhang, e. A roadmap to graph analytics. ACM SIGMOD Record 53, 4 (2025), 43–51.
12. Boulicaut, J.-F., Klemettinen, M., and Mannila, H. Querying inductive databases: A case study on the mine rule operator. In Principles of Data Mining and Knowledge Discovery: Second European Symposium, PKDD'98 Nantes, France, September 23–26, 1998 Proceedings 2 (1998), Springer, pp. 194–202.
13. Braga, D., Campi, A., Klemettinen, M., and Lanzi, P. Mining association rules from xml data. In Data Warehousing and Knowledge Discovery: 4th International Conference, DaWaK 2002 Aix-en-Provence, France, September 4–6, 2002 Proceedings 4 (2002), Springer, pp. 21–30.
14. Bringmann, B., and Nijssen, S. What is frequent in a single graph? In Advances in Knowledge Discovery and Data Mining: 12th Pacific-Asia Conference, PAKDD 2008 Osaka, Japan, May 20-23, 2008 Proceedings 12 (2008), Springer, pp. 858–863.
15. Campi, A., and Palese, C. Twitter association rule mining using clustering and graph databases. In 2021 the 5th International Conference on Information System and Data Mining (2021), pp. 90–95.
16. Changchien, S. W., and Lu, T.-C. Mining association rules procedure to support on-line recommendation by customers and products fragmentation. Expert systems with applications 20, 4 (2001), 325–335.
17. Ciftçi, O., Tenekeci, S., et al. Artist recommendation based on association rule mining and community detection. In Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management-KDIR (2021), SCITEPRESS.
18. Colombo, A., Bernasconi, A., and Ceri, S. An llm-assisted etl pipeline to build a high-quality knowledge graph of the italian legislation. Information Processing & Management 62, 4 (2025), 104082.
19. Colombo, A., Cambria, F., and Invernici, F. Legislative knowledge management with property graphs. In Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference (2025).
20. Deutsch, A., Francis, N., Green, A., Hare, K., Li, B., Libkin, L., Lindaaker, T., Marsault, V., Martens, W., Michels, J., et al. Graph pattern matching in gql and sql/pgq. In Proceedings of the 2022 International Conference on Management of Data (2022), pp. 2246–2258.
21. Dhiman, A., and Jain, S. Optimizing frequent subgraph mining for single large graph. Procedia Computer Science 89 (2016), 378–385.
22. Fan, W., Fu, W., Jin, R., Lu, P., and Tian, C. Discovering association rules from big graphs. Proc. VLDB Endow. 15, 7 (mar 2022), 1479–1492.
23. Fan, W., Han, Z., Wang, Y., and Xie, M. Parallel rule discovery from large datasets by sampling. In Proceedings of the 2022 international conference on management of data (2022), pp. 384–398.
24. Fan, W., Wang, X., Wu, Y., and Xu, J. Association rules with graph patterns. Proceedings of the VLDB Endowment 8, 12 (2015), 1502–1513.
25. Fan, W., Wu, Y., and Xu, J. Functional dependencies for graphs. In Proceedings of the 2016 international conference on management of data (2016), pp. 1843–1857.
26. Fenner, T., Levene, M., and Loizou, G. Predicting the long tail of book sales: Unearthing the power-law exponent. Physica A: Statistical Mechanics and Its Applications 389, 12 (2010), 2416–2421.
27. Fernandes, D., Bernardino, J., et al. Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb. Data 10 (2018), 0006910203730380.
28. Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., and Taylor, A. Cypher: An evolving query language for property graphs. In Proceedings of the 2018 International Conference on Management of Data (New York, NY, USA, 2018), Association for Computing Machinery, pp. 1433–1445.
29. Gudes, E., Shrimony, E., and Vanetik, N. Discovering frequent graph patterns using disjoint paths. IEEE Transactions on Knowledge and Data Engineering 18, 11 (2006), 1441–1456.
30. Hegland, M. The Apriori algorithm - A tutorial. pp. 209–262.

31. Huynh, B., Nguyen, L., Duc, N., Nguyen, N., Nguyen, H.-S., Pham, T., Pham, T., Nguyen, L., and Vo, B. Mining association rules from a single large graph. Cybernetics and Systems 55, 3 (2023), 693–707.
32. Invernici, F., Bernasconi, A., and Ceri, S. Searching covid-19 clinical research using graph queries: Algorithm development and validation. Journal of Medical Internet Research 26 (2024), e52655.
33. Invernici, F., Cambria, F., Bernasconi, A., and Ceri, S. GitHub repository of mineGraphRule. `https://github.com/FrInve/mine_graph_rule`, 2025. Last accessed online: January 22nd, 2025.
34. Jouili, S., and Vansteenberghe, V. An empirical comparison of graph databases. In 2013 International Conference on Social Computing (2013), IEEE, pp. 708–715.
35. Kaur, M., Garg, U., and Kaur, S. Advanced eclat algorithm for frequent itemsets generation. International Journal of Applied Engineering Research 10, 9 (2015), 23263–23279.
36. Lin, W., Alvarez, S. A., and Ruiz, C. Efficient adaptive-support association rule mining for recommender systems. Data mining and knowledge discovery 6 (2002), 83–105.
37. Liu, X., Dong, B., Fu, W., Wu, N., Wang, X., and Wang, W. Extending graph rules with oracles. Proc. VLDB Endow. 17, 7 (may 2024), 1775–1787.
38. Mahanti, A., Carlsson, N., Mahanti, A., Arlitt, M., and Williamson, C. A tale of the tails: Power-laws in internet measurements. IEEE Network 27, 1 (2013), 59–64.
39. Meo, R., Psaila, G., and Ceri, S. An extension to sql for mining association rules. Data mining and knowledge discovery 2 (1998), 195–224.
40. Meo, R., Psaila, G., Ceri, S., et al. A new SQL-like operator for mining association rules. In VLDB (1996), vol. 96, Citeseer, pp. 122–133.
41. Moniruzzaman, A., and Hossain, S. A. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. arXiv preprint arXiv:1307.0191 (2013).
42. Nahar, J., Imam, T., Tickle, K. S., and Chen, Y.-P. P. Association rule mining to detect factors which contribute to heart disease in males and females. Expert systems with applications 40, 4 (2013), 1086–1093.
43. Neo4j Cypher Manual. CALL Procedure. `https://neo4j.com/docs/cypher-manual/current/clauses/call`, 2024. Last accessed online: January 22nd, 2025.
44. Osadchiy, T., Poliakov, I., Olivier, P., Rowland, M., and Foster, E. Recommender system based on pairwise association rules. Expert Systems with Applications 115 (2019), 535–542.
45. Parviainen, P., Tihinen, M., Kääriäinen, J., and Teppola, S. Tackling the digitalization challenge: how to benefit from digitalization in practice. International journal of information systems and project management 5, 1 (2017), 63–77.
46. Raschka, S. Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack. The Journal of Open Source Software 3, 24 (Apr. 2018).
47. Robinson, I., Webber, J., and Eifrem, E. Graph databases: new opportunities for connected data. "O'Reilly Media, Inc.", 2015.
48. Santoso, M. H. Application of Association Rule Method Using Apriori Algorithm to Find Sales Patterns Case Study of Indomaret Tanjung Anom. Brilliance: Research of Artificial Intelligence 1, 2 (2021), 54–66.
49. Sasaki, Y., and Karras, P. Mining path association rules in large property graphs. In Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (2024), pp. 1994–2003.
50. Secretary, I. C. Information Technology - Database Languages - GQL. Standard ISO/IEC WD 39075. `https://www.iso.org/standard/76120.html`, 2024.
51. Sen, S., Mehta, A., Ganguli, R., and Sen, S. Recommendation of influenced products using association rule mining: Neo4j as a case study. SN Computer Science 2 (2021), 1–17.
52. Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D. J., and Naughton, J. F. Relational databases for querying xml documents: Limitations and opportunities. 302–314.
53. Shkurenko, V. GitHub repository of mineGraphRule. `https://www.kaggle.com/datasets/viktoriiashkurenko/278k-spotify-songs/data?select=music_genres.txt`, 2023.
54. Si, H., Zhou, J., Chen, Z., Wan, J., Xiong, N. N., Zhang, W., and Vasilakos, A. V. Association rules mining among interests and applications for users on social networks. IEEE Access 7 (2019), 116014–116026.
55. Srikant, R., and Agrawal, R. Mining generalized association rules. Future generation computer systems 13, 2-3 (1997), 161–180.
56. Standl, B., and Schlomske-Bodenstein, N. A pattern mining method for teaching practices. Future Internet 13, 5 (2021), 106.
57. Stilou, S., Bamidis, P. D., Maglaveras, N., and Pappas, C. Mining association rules from clinical databases: an intelligent diagnostic process in healthcare. Studies in health technology and informatics, 2 (2001), 1399–1403.
58. Tandan, M., Acharya, Y., Pokharel, S., and Timilsina, M. Discovering symptom patterns of covid-19 patients using association rule mining. Computers in biology and medicine 131 (2021), 104249.
59. Vanetik, N., Shimony, S. E., and Gudes, E. Support measures for graph data. Data Mining and Knowledge Discovery 13, 2 (2006), 243–260.
60. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., and Wilkins, D. A comparison of a graph database and a relational database: a data provenance perspective. In Proceedings of the 48th annual Southeast regional conference (2010), pp. 1–6.
61. Wang, X., and Xu, Y. Mining graph pattern association rules. In International Conference on Database and Expert Systems Applications (2018), Springer, pp. 223–235.
62. Zaki, M. J. Scalable algorithms for association mining. IEEE transactions on knowledge and data engineering 12, 3 (2000), 372–390.