Learning SHACL Shapes from Knowledge Graphs

Pouya Ghiasnezhad Omran [a,*], Kerry Taylor [a] Sergio Rodriguez Mendez [a] and Armin Haller [a]
[a] *School of Computing, The Australian National University , ACT, Australia*
E-mails: P.G.omran@anu.edu.au, kerry.taylor@anu.edu.au, Sergio.RodriguezMendez@anu.edu.au, armin.haller@anu.edu.au

Abstract. Knowledge Graphs (KGs) have proliferated on the Web since the introduction of knowledge panels to Google search in 2012. KGs are large data-first graph databases with weak inference rules and weakly-constraining data schemes. SHACL, the Shapes Constraint Language, is a W3C recommendation for expressing constraints on graph data as shapes. SHACL shapes serve to validate a KG, to underpin manual KG editing tasks and to offer insight into KG structure.

We introduce Inverse Open Path (IOP) rules, a predicate logic formalism which presents specific shapes in the form of paths over connected entities. Although IOP rules express simple shape patterns, they can be augmented with minimum cardinality constraints and also used as a building block for more complex shapes, such as trees and other rule patterns. We define quality measures for IOP rules and propose a novel method to learn high-quality rules from KGs. We show how to build high-quality tree shapes from the IOP rules. Our learning method, SHACLEARNER, is adapted from a state-of-the-art embedding-based open path rule learner (OPRL).

We evaluate SHACLEARNER on some real-world massive KGs, including YAGO2s (4M facts), DBpedia 3.8 (11M facts), and Wikidata (8M facts). The experiments show SHACLEARNER can learn informative and intuitive shapes from massive KGs effectively. Our experiments show the learned shapes are diverse in both structural features such as depth and width, and in quality measures.

Keywords: SHACL Shape Learning, Shapes Constraint Language, Knowledge Graph, Inverse Open Path Rule

1. Introduction

While public knowledge graphs (KGs) became popular with the development of DBpedia [1] and Yago [2] more than a decade ago, interest in enterprise knowledge graphs [3] has only taken off since the inclusion of knowledge panels on the Google Search engine, driven by its internal knowledge graph, in 2012. Although these KGs are massive and diverse, they are typically incomplete. Regardless of the method that is used to build a KG (e.g. collaboratively vs individually, manually vs automatically), it will be incomplete because of the evolving nature of human knowledge, cultural bias [4] and resource constraints. Consider Wikidata [5], for example, where there is more complete information for some types of entities (e.g. pop stars), while less for others (e.g. opera singers). Even for the same type of entity, for example, computer scientists, there are different depths of detail depending on the country of origin of the scientist.

However, the power of KGs comes from their data-first approach, enabling contributors to extend a KG in a relatively arbitrary manner. By contrast, a relational database typically employs not-null and other constraints that require some attributes to be instantiated at all times. Large KGs are typically populated by automatic and semi-automatic methods using non-structured sources (e.g. Wikipedia) that are prone to errors of omission and commission.

SHACL[6] was formally recommended by the W3C in 2017 to express constraints on a KG as *shapes*. For example, SHACL can be used to express that a person needs to have a name, birth date, and place of birth, and that these attributes have particular types: a string;

*Corresponding author. E-mail: P.G.omran@anu.edu.au.

a date; a location. The shapes are used to guide the population of a KG, although they are not necessarily enforced. Typically, SHACL shapes are manually-specified and the methods to do this are well-studied. However, as for multidimensional relational database schemes [7], shapes could, in principle, be inferred from KG data. As frequent patterns, the shapes characterise a KG and can be used for subsequent data cleaning or ongoing data entry. There is scant previous research on this topic [8].

While basic SHACL [6] and its advanced features [9] allows the modelling of diverse shapes including rules and constraints, most of these shapes are previously well known as expressed by alternative formalisms, including closed rules [10], trees, existential rules [11], and graph functional dependencies [12]. We claim that the common underlying form of all these shapes is the *path*, over which additional constraints induce the various versions. For example in DBpedia we discover the following path, $< dbo : type > \_ < db : Song > (x) \rightarrow < dbo : album > (x,y) \wedge < dbo : recordLabel > (y,z)$. which expresses that if an entity x is a song, then x is in an album y which has a record label z.

Since the satisfaction of a less-constrained shape is a necessary condition for satisfaction of a more complex shape (but not a sufficient condition), in this paper we focus on learning paths, the least constrained shape for our purposes. Paths can serve as the basis for more complex shapes. We also investigate the process of constructing one kind of more complex shape, that is a *tree*, out of paths. For example, we discover a tree about an entity which has song as its type as we show in Fig. 1. In a KG context, the tree suggests that if we have an entity of type song in the KG, then we would expect to have the associated facts as well.

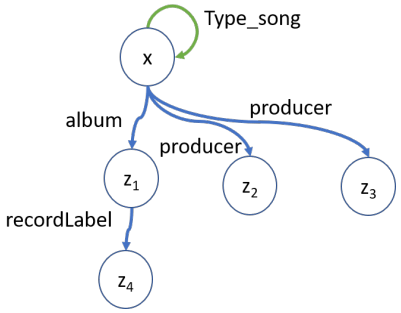

Fig. 1. A tree shape for the Song concept from DBpedia.

In this paper, we present a system, SHACLEARNER, that mines shapes from KG data. For this purpose we propose a predicate calculus formalism in which rules have one body atom and a chain of conjunctive atoms in the head with a specific variable binding pattern. Since these rules are an inverse version of *open path rules* [13], we call them *inverse open path* (IOP) rules. To learn IOP rules we adapt an embedding-based open path rule learner, OPRL [13]. We define quality measures to express the validity of IOP rules in a KG. SHACLEARNER uses the mined IOP rules to subsequently discover more complex tree shapes. Each IOP rule or tree is a SHACL shape, in the sense that it can be syntactically rewritten in SHACL. Our mined shapes are augmented with a novel numerical confidence measure to express the strength of evidence in the KG for each shape.

2. Preliminaries

An *entity* e is an identifier for an object such as a place or a person. A *fact* (also known as a *link*) is an RDF triple (e,P,e'), written here as as $P(e,e')$, meaning that the *subject* entity e is related to an *object* entity e' via the binary *predicate* (also known as a *property*), P. In addition, we admit *unary* predicates of the form $P(e)$, also written as the fact $P(e,e)$. We model unary predicates as self-loops to make the unary predicate act as the label of an link in the graph, just as for binary predicates. Unary predicates may, but are not limited to, represent class assertions expressed in an RDF triple as $(e, \texttt{rdf:type}, P)$ where P is a class or a datatype. A *knowledge graph* (*KG*) is a pair $K = (E,F)$, where E is a set of entities and F is a set of facts and all the entities occurring in F also occur in E.

2.1. Closed-Path Rules

KG rule learning systems employ various rule languages to express rules. RLvLR [14] and SCALEKB [15] use so-called *closed path* (CP) rules that are a kind of closed rule as they have no free variables. Each consists of a *head* at the front of the implication arrow and a *body* at the tail. We say the rule is *about* the predicate of the head. The rule forms a closed path, or single unbroken loop of links between the variables. It has the following general form.

$$P_t(x,y) \leftarrow P_1(x,z_1) \wedge P_2(z_1,z_2) \wedge ... \wedge P_n(z_{n-1},y). \quad (1)$$

We interpret these kinds of rules with quantification of all variables at the outside, and so we can infer a fact that instantiates the head of the rule by finding an instantiation of the body of the rule in the KG. For example, from the rule $citizenOf(x,y) \leftarrow livesIn(x,z) \wedge locatedIn(z,y)$, if we have the facts, $livesIn(Mary, Canberra)$ and $locatedIn(Canberra, Australia)$ in the KG, then we can infer and assert the following new fact, $citizenOf(Mary, Australia)$.

Rules are considered of more use if they generalise well, that is, they explain many facts. To quantify this idea we recall measures *support*, *head coverage* and *standard confidence* that are used in some major approaches to rule learning including [15] and [10].

Definition 1 (satisfies, support)**.** *Let r be a CP rule of the form (1). A pair of entities (e,e')* satisfies *the body of r, denoted $body_r(e,e')$, if there exist entities $e_1,...,e_{n-1}$ in the KG such that all of $\{P_1(e,e_1), P_2(e_1,e_2),...,P_n(e_{n-1},e')\}$ are facts in the KG. Further (e,e')* satisfies *the head of r, denoted $P_t(e,e')$, if $P_t(e,e')$ is a fact in the KG. Then the* support *of r counts the rule instances for which the body and the head are both satisfied in the KG.*

$$supp(r) = |\{(e,e') : body_r(e,e') \text{ and } P_t(e,e')\}|$$

Standard confidence (SC) and *head coverage* (HC) offer standardised measures for comparing rule quality. SC describes how frequently the rule is true, i.e. of the number of entity pairs that satisfy the body in the KG, what proportion of the inferred head instances are satisfied? It is closely related to *confidence* widely used in association rule mining [16]. HC measures the explanatory power of the rule, i.e. what proportion of the facts satisfying the head of the rule could be inferred by satisfying the rule body? It is closely related to *cover* which is widely used for rule learning in inductive logic programming [17]. A non-recursive rule that has both 100% SC and HC is redundant with respect to the KG, and every KG fact that is an instance of the rule head is redundant with respect to the rule.

Definition 2 (standard confidence, head coverage)**.** *Let $r, e, e', body_r$ be as given in definition 1. Then* standard confidence*,*

$$SC(r) = \frac{supp(r)}{|\{(e,e') : body_r(e,e')\}|}$$

and head coverage*,*

$$HC(r) = \frac{supp(r)}{|\{(e,e') : P_t(e,e')\}|}$$

2.2. Open-Path Rules: Rules with Free Variables

Unlike all earlier work in rule mining for KG completion, *active* knowledge graph completion [13] defines *open path* (OP) rules of the form:

$$P_t(x,z_0) \leftarrow P_1(z_0,z_1) \wedge P_2(z_1,z_2) \wedge ... \wedge P_n(z_{n-1},y). \quad (2)$$

Here, P_i is a predicate in the KG and each of $\{x, z_i, y\}$ are entity variables. Unlike CP rules, OP rules do not necessarily form a loop, but a straightforward variable unification transforms an OP rule to a CP rule, and every entity instantiation of a CP rule is also an entity instantiation of the related OP rule (but not vice-versa).

To assess the quality of open path rules, *open path standard confidence (OPSC)* and *open path head coverage (OPHC)* are derived in [13] from the closed path forms (Definition 2).

Definition 3 (open path: OPsupp, OPSC, OPHC)**.** *Let r be an OP rule of the form (2). Then a pair of entities (e,e')* satisfies *the body of r, denoted $body_r(e,e')$, if there exist entities $e_1,...,e_{n-1}$ in the KG such that $P_1(e,e_1)$, $P_2(e_1,e_2),...,P_n(e_{n-1},e')$ are facts in the KG. Also (e',e)* satisfies *the head of r, denoted $P_t(e',e)$, if $P_t(e',e)$ is a fact in the KG. The* open path support*,* open path standard confidence*, and* open path head coverage *of r are given respectively by*

$$OPsupp(r) = |\{e : \exists e', e'' \text{ s.t. } body_r(e,e') \text{ and } P_t(e'',e)\}|$$

$$OPSC(r) = \frac{OPsupp(r)}{|\{e : \exists e' \text{ s.t. } body_r(e,e')\}|}$$

$$OPHC(r) = \frac{OPsupp(r)}{|\{e : \exists e' \text{ s.t. } P_t(e',e)\}|}$$

2.3. SHACL Shapes

A KG is a schema-free database and does not need to be augmented with schema information natively. However, many KGs are augmented with type information that can be used to understand and validate data and can also be very helpful for inference processes on the KG. In 2017 the Shapes Constraint Language (SHACL) [6] was introduced as a W3C recommendation to define schema information for KGs stored as RDF datasets. SHACL defines constraints for graphs

as *shapes*. KGs can then be validated against a set of shapes.

Shapes serve two main purposes: validating the quality of a KG and characterising the frequent patterns in a KG. In Fig. 2, we illustrate an example of a shape from Wikidata [1], where x and z_is are variables that are instantiated by entities. Although the shape is originally expressed in ShEx [18], we translate it to SHACL here.

SHACL, together with SHACL advanced features [9] is extensive. Here we focus on the core of SHACL in which *node* shapes constrain a *target* predicate (e.g. the unary predicate *human* in Fig. 2), with *property* shapes expressing constraints over facts related to the target predicate. We particularly focus on property shapes which act to constrain an argument of the target predicate. Continuing the example of Fig. 2, the shape expresses that each entity x which satisfies $human(x)$ should satisfy the following paths: (1) $citizenOf(x,z_1) \wedge country(z_1)$, (2) $father(x,z_2) \wedge human(z_2)$, and (3) $nativeLanguage(x,z_3) \wedge language(z_3)$.

Many kinds of shapes, including tree and closed rule, can be expressed by SHACL. We focus on path shapes for our shape learning system. A *path* is a sequence of predicates with chained intermediate variables but permitting unbound variables at both ends. Although shapes in the form of a path are not as constrained as more complex shapes like closed rules, they are a necessary condition for the more complex shapes like closed rule or tree, which are also paths (with further restrictions). We will define *Inverse Open Path* rules induced from paths that have a straightforward interpretation as shapes, and also propose a method to mine such rules from a KG. To demonstrate the potential for these kind of shapes to serve as building blocks for more complex shapes, we then propose a method that builds trees out of mined rules, and discuss the application of such trees to KG completion.

```
Shape:human
a sh:NodeShape;
sh:targetclass class:human;
sh:property[
    sh:path: citizenOf;
    sh:path[citizenOf
        sh:nodekind country;]
        ];
sh:property[
    sh:path: father;
    sh:path[father
        sh:nodekind human;]
        ];
sh:property[
    sh:path: nativeLanguage;
    sh:path[nativeLanguage
        sh:nodekind language;]
        ];
```

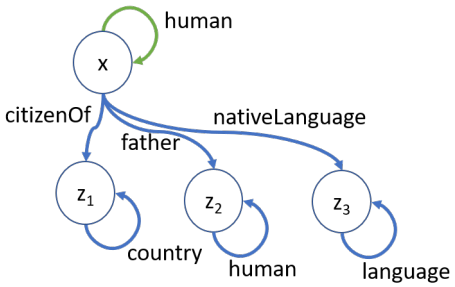

Fig. 2. An example of a SHACL shape from Wikidata.

3. SHACL learning

3.1. Rules with Free Variables or Uncertain Shapes

We observe that the converse of OP rules, that we call *inverse open path rules* (IOP), correspond to SHACL shapes. For example, the shape of Fig.2 has the following three IOP rules as building blocks:

$$human(x,x) \rightarrow citizenOf(x,z_1) \wedge country(z_1,z_1).$$

$$human(x,x) \rightarrow father(x,z_2) \wedge human(z_2,z_2).$$

$$human(x,x) \rightarrow nativeLanguage(x,z_3) \wedge language(z_3,z_3).$$

The general form of an IOP rule is given by

$$P'_t(x,z_0) \rightarrow \exists(z_1,..z_{n-1},y)P'_1(z_0,z_1) \wedge P'_2(z_1,z_2) \wedge ... \wedge P'_n(z_{n-1},y). \quad (3)$$

where each P'_i is either a predicate in the KG or its reverse with the subject and object bindings swapped. These are not Horn rules. In an IOP rule the body of the rule is P_t and its head is the sequence of predicates, $P_1 \wedge P_2 \wedge ... \wedge P_n$. Hence we instantiate the atomic body to predict an instance of the head. This pattern of existential quantification in the head and free variables in the body of a rule has been investigated in the literature as existential rules [11].

[1] https://www.wikidata.org/wiki/EntitySchema:E10

To assess the quality of IOP rules we follow the quality measures for OP rules [13].

Definition 4 (inverse open path: IOPsupp, IOPSC, IOPHC)**.** *Let r be an IOP rule of the form (3). Then a pair of entities* (e, e') satisfies *the head of r, denoted* $head_r(e, e')$*, if there exist entities* $e_1, ..., e_{n-1}$ *in the KG such that* $P_1(e, e_1), P_2(e_1, e_2), ..., P_n(e_{n-1}, e')$ *are facts in the KG. A pair of entities* (e'', e) satisfies *the body of r, denoted* $P_t(e'', e)$*, if* $P_t(e'', e)$ *is a fact in the KG. The* inverse open path support*,* inverse open path standard confidence*, and* inverse open path head coverage *of r are given respectively by*

$$IOPsupp(r) =\mid \{e : \exists e', e'' \text{ s.t. } head_r(e, e') \text{ and } P_t(e'', e)\} \mid$$

$$IOPSC(r) = \frac{IOPsupp(r)}{\mid \{e : \exists e'' \text{ s.t. } P_t(e'', e)\} \mid}$$

$$IOPHC(r) = \frac{IOPsupp(r)}{\mid \{e : \exists e' \text{ s.t. } head_r(e, e')\} \mid}$$

Notably, because a single open path induces both an OP and IOP rule, the support for an IOP rule is the same as the support for its corresponding OP form, IOPSC is the same as the corresponding OPHC, and IOPHC is the same as the corresponding OPSC. This close relationship between OP and IOP rules helps us to mine both OP and IOP rules in the one process.

We show the relationship between an OP rule and its converse IOP version in the following example. Consider the OP rule, $P_1(x, z_0)$
$\leftarrow P_2(z_0, z_1) \wedge P_3(z_1, z_2)$. Assume we have three entities ($\{e_3, e_4, e_5\}$) which can instantiate z_0 and satisfy both $P_1(x, z_0)$ and $P_2(z_0, z_1) \wedge P_3(z_1, z_2)$. Assume the number of entities that can instantiate z_0 to satisfy the head part is 5 ($\{e_1, e_2, e_3, e_4, e_5\}$) and the number of entities that can instantiate z_0 to satisfy the body part is 7 ($\{e_3, e_4, e_5, e_6, e_7, e_8, e_9\}$). Hence, we have the following for this rule, $OPsupp = 3$, $OPSC = 3/7$ and $OPHC = 3/5$. For the IOP version of the rule over the same KG, $P_1(x, z_0) \rightarrow P_2(z_0, z_1) \wedge P_3(z_1, z_2)$, we have the same entities to instantiate z_o while the predicate terms corresponding to the bodies and heads are swapped. Hence, we have the following for the IOP version of the rule, $IOPsupp = 3$, $IOPSC = 3/5$ and $IOPHC = 3/7$.

In many cases we need the rules to express not only the necessity of a chain of facts (the facts in the head of the IOP rule) but the number of different chains which should exist. For example, we may need a rule to express that each human has at least two parents. Hence, we introduce IOP rules annotated with a cardinality, Car. This gives the following annotated form for each IOP rule.

$$IOPSC, IOPHC, Car : P'_t(x, z_0) \rightarrow P'_1(z_0, z_1) \wedge P'_2(z_1, z_2) \wedge ... \wedge P'_n(z_{n-1}, y) \quad (4)$$

where IOPSC and IOPHC belong to $[0, 1]$ and denote those qualities of the rule. Car is an integer $\geqslant 1$.

Definition 5 (Cardinality of an IOP rule, Car)**.** *Let r be an annotated IOP rule of the form (4) and let* $Car(r)$ *be the cardinality annotation for r. Then r* satisfies $Car(r)$ *iff for each entity* $e \in \{e | \exists e'' s.t.\ P_t(e'', e)\}$, $Car(r) \leqslant \mid \{e' | head_r(e, e')\} \mid$.

The cardinality expresses a lower bound on the number of head paths which are satisfied in the KG for every instantiation of the variable that joins the body to the head. For example, if we have $0.8, 0.1, 2 : P_t(x, y) \rightarrow P_1(y, z) \wedge P_2(z, t)$ and have $P_t(e_1, e_2)$ satisfied, we should have at least two paths starting from e_2 that instantiate two distinct entities for variable t, of the form $P_1(e_2, z) \wedge P_2(z, e_3)$ and $P_1(e_2, z) \wedge P_2(z, e_4)$. There is no limitation on the instantiations for variable z which is scoped inside the head, so these two pairs of instantiations both satisfy cardinality of 2: (1) $P_1(e_2, e_5) \wedge P_2(e_5, e_3)$ and $P_1(e_2, e_5) \wedge P_2(e_5, e_4)$ and (2) $P_1(e_2, e_5) \wedge P_2(e_5, e_3)$ and $P_1(e_2, e_6) \wedge P_2(e_6, e_4)$.

Rules with the same head and the same body may have different cardinalities. In the given example we might have the following rule as well, $0.9, 0.1, 1 : P_t(x, y) \rightarrow P_1(y, z) \wedge P_2(z, t)$. While we have a rule with a cardinality, c_1, we also have rules with lower cardinalities $1, ..., (c_1 - 1)$ but their IOPSCs should be as good or better than the rule with c_1 cardinality. This statement is quite intuitive since cardinality expresses the lower bound number of occurrences of the head.

Lemma 1 (IOPSC is non-increasing with length)**.** *Let r be an IOP rule of the form (3) with* $n \geqslant 2$ *and let* r' *be an IOP rule of the form* $P'_t(x, z_0) \rightarrow P'_1(z_0, z_1) \wedge P'_2(z_1, z_2) \wedge ... \wedge P'_n(z_{n-2}, y)$*, being r shortened by the removal of the last head predicate. Then* $IOPSC(r) \leqslant IOPSC(r')$.

Proof. Observe that by definition 4, the denominator of $IOPSC()$ is not affected by the head of the rule, and so has the same value for both. Now, looking at the

Algorithm 1 SHACLEARNER

Input: a KG K, a target predicate P_t
Parameters: max rule length l, max rule cardinality $MCar$, $MinIOPSC$, $MinIOPHC$, and $MinTreeSC$
Output: a set of IOP rules R and $Tree$

1: $K' :=$ Sampling(K, P_t)
2: $(\mathcal{P}, \mathcal{A}) :=$ Embeddings(K')
3: $R' := \emptyset$
4: **for** $2 \leqslant k \leqslant l$ **do**
5: Add PathFinding$(K', P_t, \mathcal{P}, \mathcal{A}, k)$ to R'
6: **end for**
7: $R :=$ Evaluation$(R', K, MCar, MinIOPSC, MinIOPHC)$
8: $Tree :=$ GreadySearch$(R, MinTreeSC)$
9: **return** $Tree$ and R

numerators, we have that $\forall e(\exists e_1, .., e_n(P'_1(e, e_1) \wedge P'_2(e_1, e_2) \wedge ... \wedge P'_n(e_{n-2}, e_{n-1}) \wedge P'_n(e_{n-1}, e_n)) \implies \exists e_1, .., e_{n-1}(P'_1(e, e_1) \wedge P'_2(e_1, e_2) \wedge ... \wedge P'_n(e_{n-2}, e_{n-1})))$. Therefore $\{e : \exists e' \; s.t. \; head_r(e, e')\} \subseteq \{e : \exists e' \; s.t. \; head_{r'}(e, e')\}$. Therefore $\{e : \exists e', e'' \; s.t. \; head_r(e, e') \; and \; P_t(e'', e)\} \subseteq \{e : \exists e', e'' \; s.t. \; head_{r'}(e, e') and \; P_t(e'', e)\}$. So $IOPsupp(r) \leqslant IOPsupp(r')$ as required.

□

3.2. IOP Learning through Representation Learning

We start with the open path rule learner OPRL [13], and adapt its embedding-based OP rule learning to learn annotated IOP rules. We call this new learner, SHACLEARNER, shown in Algorithm 1.

Our SHACLEARNER uses a sampling method Sampling() which prunes the entities and predicates that are less relevant to the target predicate to obtain a sampled KG. The sample is fed to embedding learners such as RESCAL [19, 20] in Embeddings(). Then in PathFinding() SHACLEARNER uses the computed embedding representations of predicates and entities in heuristic functions that inform the generation of IOP rules bounded by a maximum length. Then, potential IOP rules evaluated, annotated, and filtered in Evaluation() to produce annotated IOP rules. Eventually, a tree is discovered for each argument of each target predicate by aggregating mined IOP rules.

In more detail, in line 1, the Sampling() method computes a fragment of the KG (K') consisting of a bounded number of entities that are related to the goal predicate (i.e., P_t). This sampling is essential since embedding learners (e.g. HOLE [19] and RESCAL) cannot handle massive KGs with millions of entities (e.g. YAGO2).

After sampling, in line 2 Embeddings(), we compute predicate embeddings as well as subject and object argument embeddings for all predicates which exist in the K' after sampling, as is done in RLvLR [14].

In a nutshell, we use RESCAL [19] to embed each entity e_i to a vector $\mathbf{E}_i \in \mathbb{R}^d$ and each predicate P_k to a matrix $\mathbf{P}_k \in \mathbb{R}^{d \times d}$ where $\mathbb{R}$ is the set of real numbers and d is an integer (a parameter of RESCAL). For each given fact $P_0(e_1, e_2)$, the following scoring function is computed:

$$f(e_1, P_0, e_2) = \mathbf{E}_1^T \cdot \mathbf{P}_0 \cdot \mathbf{E}_2 \tag{5}$$

The scoring function indicates the plausibility of the fact that e_1 has relation P_0 with e_2. The two sets of embeddings, $\{\mathbf{E}_i\}$ and $\{\mathbf{P}_k\}$ are learned by minimizing a a loss function.

The argument embeddings are also computed in Embeddings() according to the method proposed in [14]. To compute the subject (respectively object) argument embeddings of a predicate P_k, we aggregate the embeddings of entities that occur as the subject (respectively object) of P_k in the KG. Hence for each predicate P_k we have two vectors, $\mathbf{P}_k^1$ and $\mathbf{P}_k^2$ that present the subject argument and object argument of P_k respectively.

After that, in line 3 to line 7, PathFinding() produces potential IOP rules based on the embedding representation of the predicates which are involved in each rule. The potential rules are pruned by the scoring function heuristic proposed in [13] for OP rules. A high-scoring potential path suggests the existence of both OP and IOP rules.

An IOP rule $P_t(x, y) \rightarrow P_1(y, z) \wedge P_2(z, t)$., acts to connect entities satisfying the subject argument of the body predicate, P_t, to entities forming the object argument of the last predicate, P_2, along a path of entities that satisfy a chain of predicates in the rule. There is a relationship between the logical statement of the rule and the following properties in embedding space [13]:

1. The predicate arguments that have same variable in the rule should have similar argument emdeddings. For example we should have the following similarities, $\mathbf{P}_t^2 \sim \mathbf{P}_1^1$ and $\mathbf{P}_1^2 \sim \mathbf{P}_1^2$.
2. The whole path forms a composite predicate, like $P^*(x, t) = P_t(x, y) \wedge P_1(y, z) \wedge P_2(z, t)$.

We compute the embedding representation of the composite predicate based on its components, $\mathbf{P}^*(x,t) = \mathbf{P}_t(x,y) \cdot \mathbf{P}_1(y,z) \cdot \mathbf{P}_2(z,t)$. Now we could check the plausibility of $P^*(x,t)$ for any pair of entities by equation 5. However, since we are interested in the existence of an entity-free rule, the following similarity will hold: $1 \approx \mathbf{P}_t^1 \cdot \mathbf{P}_t \cdot \mathbf{P}_1 \cdot \mathbf{P}_2 \cdot \mathbf{P}_2^2$.

Based on the above two properties, [13] defines two scoring functions that help us to heuristically mine the space of all possible IOP rules to produce a reduced set of candidate IOP rules.

The ultimate evaluation of an IOP rule will be done in the next step as we will describe.

In line 8, we use a greedy search to aggregate the discovered IOP rules for each argument of each target predicate and form a tree (as illustrated in Figure 3). The detail of this process will be described in section 3.3.

3.2.1. Efficient Computation of Quality Measures

The structure of SHACLEARNER follows OPRL [13] very closely, so now we focus our attention on the efficient matrix-computation of the quality measures that are novel for SHACLEARNER. Evaluation() in Algorithm 1 first evaluates candidate rules on the small sampled KG, and selects only the rules with $IOPsupp(r) \geqslant 1$. They may still include a large number of redundant and low quality rules and so are further downselected based on their IOPSC and IOPHC calculated over the full KG. We show in the following how to efficiently compute the measures over massive KGs using an *adjacency matrix* representation of the KG.

Let $K = (E,F)$ with $E = \{e_1,\ldots,e_n\}$ be the set of all entities and $\mathbb{P} = \{P_1,\ldots,P_m\}$ be the set of all predicates in F. Like RESCAL [19, 20], we represent K as a set of square $n \times n$ adjacency matrices by defining the function $\mathcal{A}$. Specifically, the $[i,j]$th element $\mathcal{A}(P_k)[i,j]$ is 1 if the fact $P_k(e_i,e_j)$ is in F; and 0 otherwise. Thus, $\mathcal{A}(P_k)$ is a matrix of binary values and the set $\{\mathcal{A}(P_k) \mid k \in \{1,\ldots,m\}\}$ represents K.

We illustrate the computation of IOPSC and IOPHC through an example. Consider the IOP rule r : $P_t(x,z_0) \rightarrow P_1(z_0,z_1) \wedge P_2(z_1,y)$. Let $E = \{e_1,e_2,e_3\}$ and

$$F = \{P_1(e_1,e_2), P_1(e_2,e_1), P_1(e_2,e_3), P_1(e_3,e_1), P_2(e_1,e_2), P_2(e_3,e_2), P_2(e_3,e_3), P_t(e_1,e_3), P_t(e_3,e_2), P_t(e_3,e_3)\}$$

The adjacency matrices for the predicates P_1, P_2 and P_t are:

$$\mathcal{A}(P_1): \begin{bmatrix}0&1&0\\1&0&1\\1&0&0\end{bmatrix}, \mathcal{A}(P_2): \begin{bmatrix}0&1&0\\0&0&0\\0&1&1\end{bmatrix},$$

$$\mathcal{A}(P_t): \begin{bmatrix}0&0&1\\0&0&0\\0&1&1\end{bmatrix}$$

For IOPSC and IOPHC (definition 4) we need to calculate (1) the number of entities that satisfy the body of the rule, i.e. $\#e : \exists e''$ *s.t.* $P_t(e'',e)$, (2) the number of entities that satisfy the head of a rule i.e. $\#e : \exists e'$ *s.t.* $head_r(e,e')$ and, (3) the number of entities that join the head of a rule to its body i.e. $\#e : \exists e', e''$ *s.t.* $head_r(e,e')$ *and* $P_t(e'',e)$.

For (1) we can read the pairs (e'',e) directly from the matrix $\mathcal{A}(P_t)$. To find distinct es we sum each *column* (corresponding to each value for the second argument) and transpose to obtain the vector $\mathcal{V}^2(P_t)$. Each non-zero element of this vector indicates a satisfying e and the number of distinct es is given by simply counting the number of non-zero elements. In the example, the only non-zero element in $\mathcal{A}(P_t)$ is $\mathcal{A}(P_t)[1,3]$ and after summing the columns and transposing we have

$$\mathcal{V}^2(P_t) = \begin{bmatrix}0\\1\\2\end{bmatrix}$$

so we have only $\mathcal{V}^2(P_t)[3]$ non-zero and $\{e_2,e_3\}$ satisfies the head with count 2.

For (2) the pairs (e,e') satisfying the head are connected by the path $P_1,P_2,\ldots P_m$, and can be obtained, as for (1), directly from the matrix product $\mathcal{A}(P_1)\mathcal{A}(P_2)\ldots\mathcal{A}(P_m)$, being the elements with a value $\geqslant$ *Car* (for rules with cardinality *Car*). To find distinct es we sum each *row* (corresponding to each value for the first argument) to obtain the vector $\mathcal{V}^1(\mathcal{A}(P_1)\mathcal{A}(P_2)\ldots\mathcal{A}(P_m))$. Each element with $\geqslant$ *Car* value of this vector indicates a satisfying e and the number of distinct es is given by counting the number of elements in $\mathcal{V}^1(\mathcal{A}(P_1)\mathcal{A}(P_2)\ldots\mathcal{A}(P_m))$.

In the example we have

$$\mathcal{A}(P_1)\mathcal{A}(P_2) = \begin{bmatrix}0&0&0\\0&2&1\\0&1&0\end{bmatrix}, \mathcal{V}^1(P_1,P_2) = \begin{bmatrix}0\\3\\1\end{bmatrix}$$

with satisfying entities e_2 and e_3 and count of 2 for $Car = 1$. For $Car = 2$, e_2 satisfies the rule so the count is 1.

Computing (3) is now straightforward. We have that the row index of non-zero elements of $\mathcal{V}^2(P_t)$ indicates entities that satisfy the second argument of the body and that row index of elements with a value $\geqslant Car$ of $\mathcal{V}^1(\mathcal{A}(P_1)\mathcal{A}(P_2)\dots\mathcal{A}(P_m))$ indicates entities that satisfy the first argument of the head. Therefore we can find the entities that satisfy both of these conditions by pairwise multiplication. That is, the entities we need are $\{e_i \mid (\mathcal{V}^2(P_t)[i] > 0 \wedge \mathcal{V}^1(\mathcal{A}(P_1)\mathcal{A}(P_2)\dots\mathcal{A}(P_m))[i]) \geqslant Car \text{ and } 1 \leqslant i \leqslant n\}$, and the count of entities is the size of this set. For the example, for $Car = 1$, we have $\{e_2, e_3\}$ in the set with count 2 and for $Car = 2$, we have $\{e_2\}$ in the set with count 1.

Hence, we could have three versions of r, i.e., r^1, r^2 and r^3 with three different Car of 1, 2, and 3 respectively. For $Car = 1$, $IOPsupp(r^1) = |\{e_2, e_3\}| = 2$. From Definition 4 we can obtain $IOPHC(r^1) = 2/2$ and $IOPSC(r^1) = 2/2$. For $Car = 2$, $IOPsupp(r^2) = |\{e_2\}| = 1$. We can obtain $IOPHC(r^2) = 1/1$ and $IOPSC(r^2) = 1/2$. In this case, we have the same qualities for $Car = 3$ as we have for $Car = 2$.

3.3. From IOP Rules to Tree Shapes

Now we turn to deriving SHACL trees from annotated IOP rules. For each target predicate we learn two sets of IOP rules, one binding the subject argument and the other binding the object argument, i.e. rules about the predicate and rules about its reverse. We aggregate the rules about a target predicate as a tree rooted at the predicate as illustrated in Fig. 3.

For example, the shape of Fig. 2 has the following tree:

$$\begin{aligned} human(x,x) \rightarrow\ & citizenOf(x,z_1) \wedge country(z_1,z_1) \\ & \wedge father(x,z_3) \wedge human(z_3,z_3) \\ & \wedge nativeLanguage(x,z_4) \wedge \\ language(z_4,z_4).& \end{aligned}$$

The general form of a tree is given by

$$P'_t(x,z_0) \rightarrow \exists(z_*^*, y^*)$$

$$\begin{aligned} & P'^1_1(z_0,z^1_1) \wedge P'^1_2(z^1_1,z^1_2) \wedge ... \wedge P'^1_n(z^1_{n-1},y^1) \\ \wedge & P'^2_1(z_0,z^2_1) \wedge P'^2_2(z^2_1,z^2_2) \wedge ... \wedge P'^2_m(z^2_{m-1},y^2) \\ & ... \\ \wedge & P'^q_1(z_0,z^q_1) \wedge P'^q_2(z^q_1,z^q_2) \wedge ... \wedge P'^q_t(z^q_{t-1},y^q) \end{aligned} \tag{6}$$

where each P'^j_i is either a predicate in the KG or its reverse with the subject and object bindings swapped. In a tree we say the body of the shape is P_t and its head is the sequence of paths or branches, $Path^1 \wedge Path^2 \wedge ... \wedge Path^q$. Hence we instantiate the atomic body to predict an instance of the head. All head branches and the body joint in one variable, z_0.

To assess the quality of a path we follow the quality measures for IOP rules.

Definition 6 (Tree: Treesupp, TreeSC)**.** *Let r be a tree of the above form. Then a set of pairs of entities $(e,e^1),...,(e,e^q)$* satisfies *the head of r, denoted $head_r(e)$, if there exist the following sequences of entities $e^1_1,...,e^1_{n-1}$, $e^2_1,...,e^2_{m-1}$ and $e^q_1,...,e^q_{t-1}$ in the KG such that $P^1_1(e,e^1_1), P^1_2(e^1_1,e^1_2),...,P^1_n(e^1_{n-1},e^1)$, $P^2_1(e,e^2_1), P^2_2(e^2_1,e^2_2)$, $...,P^2_m(e^2_{m-1},e^2)$ and $P^q_1(e,e^q_1)$, $P^q_2(e^q_1,e^q_2)$ $,...,P^q_t(e^q_{t-1},e^q)$ are facts in the KG. A pair of entities (e'',e)* satisfies *the body of r, denoted $P_t(e'',e)$, if $P_t(e'',e)$ is a fact in the KG. The* tree support *and* tree standard confidence *of r are given respectively by*

$$Treesupp(r) = |\ \{e : \exists e'' \text{ s.t. } head_r(e) \text{ and } P_t(e'',e)\}\ |$$

$$TreeSC(r) = \frac{Treesupp(r)}{|\ \{e : \exists e'' \text{ s.t. } P_t(e'',e)\}\ |}$$

To learn each tree we deploy a greedy search (GreadySearch). To do so, we sort all rules that bind the subject argument (for the left hand tree in Fig. 3) in a non-increasing order with respect to IOPSC. Then we iteratively try to add the first rule in the list to the tree and compute the TreeSC. If the TreeSC drops below the defined threshold (i.e. $TreeSCMIN$) we dismiss the rule otherwise we add it to the tree. For the right hand tree we do the same with the rules that bind the object argument of the target predicate. Since a conjunction of IOP rules form a tree, TreeSC is bounded above by the minimum IOPSC of its constituent IOP rules.

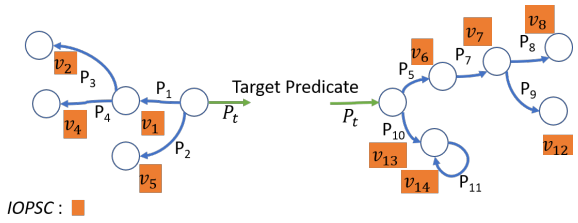

Fig. 3. Trees for a target predicate P_t. Each v_i indicates a TreeSC.

3.3.1. *Tree Shapes are Useful for Human Interaction*

Shapes offer KG documentation as readable patterns and also provide a way to validate a KG. Our novel tree shapes can additionally be used for KG-completion. While there are several methods proposed to complete KGs automatically (e.g. [14, 19, 20]) by predicting missing facts (also called *links*), all these methods traverse the KG in a breadth-first manner. In other words, they sequentially answer a set of independent questions such as $birthPlace(Trump, ?)$ or $playsIn(Naomi\_Watts, ?)$. Our proposed tree shapes instead provide an opportunity to work sequentially along a path of dependent questions such as $birthPlace(Trump, ?_1)$ followed by $capitalOf(?_1, ?)$. The latter question cannot even be asked until we have an answer for the former question, and the existence of an answer to the former gives us the confidence to proceed to the next question along the path. This completion strategy is *depth-first* as it works through a shape tree. Importantly, when we want to ask such completion questions of a human, this depth-first questioning strategy will reduce the cognitive load due to the contextual connections between successive questions. This strategy for human-KG-completion is applied in the smart KG editor *Schímatos* [21].

Tree shapes can also help a human expert to extract a more intuitive concise sub-tree out of a longer, more complex tree when desired for explainablity. If a tree with confidence $TreeSC_{orig}$ is pruned either by removing branches (width-wise) or by reducing the length of branches (depth-wise), it remains a valid tree with confidence $TreeSC_{new}$ with the property that $TreeSC_{new} \geqslant TreeSC_{orig}$. Hence, by pruning a tree we got a tree with higher degree of confidence based on the data.

4. Related Work

There are some exploratory attempts to address learning SHACL shapes from KGs [8, 22–25]. They are procedural methods without logical foundations and are not shown to be scalable to handle real-world KGs. They work with a small amount of data and the representation formalism they use for their output is difficult to compare with the well-defined IOP rules which we use in this paper. [25] carries out the task in a semi-automatic manner: it provides a sample of data to an of-the-shelf graph structure learner and provides the output in an interactive interface for a human user to create SHACL shapes.

There are some works [26, 27] that use existing ontologies for KGs to generate SHACL shapes. [26] uses two different kinds of knowledge to automatically generate SHACL shapes: ontology constraint patterns as well as input ontologies. In our work we use the KG itself to discover the shapes, without relying on external modelling artefacts.

From an application point of view, there are papers which investigate the application of SHACL shapes to the validation of RDF databases including [28, 29], but these do not contribute to the discovery of shapes.

[30] proposes an extended validation framework for the interaction between rules and SHACL shapes in KGs. When a set of inference rules and SHACL shapes are provided, a method is proposed to detect which shapes could be violated by applying a rule.

There are some partial attempts to provide logical foundations for the semantics of the SHACL language including [31] that presents the semantics of recursive SHACL shapes. By contrast, in our work we approach SHACL semantics in the reverse direction. We start with logical formalisms with both well-defined semantics and motivating use cases to derive shapes that can be trivially expressed in a fragment of SHACL.

5. Experiments

We have implemented our SHACLearner[2] based on Algorithm 1, and conducted experiments to assess it. Our experiments are designed to prove the effectiveness of our SHACLearner at capturing shapes with varying confidence, length and cardinality from various real-world massive KGs. Since our proposed system is the first method to learn shapes from massive KGs automatically, we have no benchmark with which to compare. However, the performance of our system

[2]Detailed experimental results can be found (anonymously) at https://www.dropbox.com/sh/dn1kujeg0b6o9bw/AAAbKG9KfZ7e0eaNONz2zE93a?dl=0

shows that it can handle the task satisfactorily and can be applied in practice. We demonstrate that

1. SHACLEARNER is scalable so it can handle real-world massive KGs including DBpedia with over 11M facts.
2. SHACLEARNER can learn several shapes each for various target predicates.
3. SHACLEARNER can discover diverse shapes with respect to qualities of IOPSC and IOPHC.
4. SHACLEARNER discovers shapes of varying complexity, and diverse with respect to length and cardinality.
5. SHACLEARNER discovers more complex shapes (trees) by aggregating learnt IOP rules efficiently.

Our three benchmark KGs are described in Table 1. All three are common KGs and have been used in rule learning experiments previously [10, 14].

Table 1
Benchmark specifications

KG	# Facts	# Entities	# Predicates
YAGO2s	4,122,438	2,260,672	37
Wikidata	8,397,936	3,085,248	430
Wikidata +UP	8,780,716	3,085,248	587
DBpedia 3.8	11,024,066	3,102,999	650
DBpedia 3.8 +UP	11,498,575	3,102,999	1,005

All experiments were conducted on an Intel Xeon CPU E5-2650 v4 @2.20GHz, 66GB RAM and running CentOS 8.

5.0.1. *Transforming KGs with type predicates for experiments*

In real-world KGs, concept or class membership is modeled as entity instances of a binary fact, for example DBpedia contains "<dbo:type>(x,y)" and "<dbo:class>(x,y)" predicates where the second arguments of these predicates are types (e.g. "<db:Album>" or "<db:City>") or classes (e.g. "<db:Reptile>" or "<db:Bird>"). Instead, we choose to model types and classes with *unary predicates*. To do so we make new predicates like "<dbo:type>_<db:Album>(x)" if we have facts in the form "<dbo:type> (x,<db:Album>)", where x is the name of an album. Then we produce new unary facts based on the new predicate and related facts. For example for "<dbo:type>(Revolver,<db:Album>)", we produce the new fact "<dbo:type>_<db:Album>(Revolver)". We manually choose two predicates from DBpedia 3.8 "<dbo:type>" and "<dbo:class>" and one from Wikidata "<occupation_P106>" to generate those unary predicates and facts. These predicates each have a class as their second argument. To prune the classes with few instances, we consider only the new unary predicates which have at least 100 facts. We do not remove the original predicates and facts from the KG but extend the KG with the new ones. In Table 1 we report the specifications of two benchmarks where we have added the unary predicates and facts (denoted as +UP).

5.1. *Learning IOP Rules*

We follow the established approach for evaluating KG rule-learning methods, that is, measuring the quantity and quality of distinct rules learnt. Rule quality is measured by Inverse open path standard confidence (IOPSC) and Inverse open path head coverage (IOPHC). We randomly selected 50 target predicates from Wikidata and DBPedia unary predicates (157 and 355 respectively). We used all binary predicates of YAGO2s (i.e. 37) as target predicates. Each binary target predicate serves as two target predicates, once in the straight form and secondly in its reverse form. In the straight form the object argument of the predicate is the joining variable to connect the body and head, while in the reverse form the subject argument serves to join. In this manner, we ensure that the results of SHACLEARNER on YAGO2s with its binary predicates as targets is comparable with the results for Wikidata and DBpedia that have unary predicates as targets. Hence for YAGO2s we have 74 target predicates. A 10 hour limit was set for learning each target predicate. Table 2 shows #Rules, the average numbers of quality IOP rules found; %SucTarget, the proportion of target predicates for which at least one IOP rule was found; and the running times in hours, averaged over the targets. Only high quality rules meeting minimum quality thresholds are included in these figures, that is, with IOPSC$\geqslant 0.1$ and IOPHC$\geqslant 0.01$, thresholds established in comparative work [10].

Table 2
Performance of SHACLEARNER on benchmark KGs

Benchmark	%SucTarget	#Rules	Time (hours)
YAGO2s	80	42	0.6
Wikidata+UP	82	67	1.8
DBpedia+UP	98	157	2.4

SHACLEARNER shows satisfactory performance in terms of both the runtime and the numbers of quality rules mined. Note that rules found have a variety of lengths and cardinalities. To better present the quality performance of rules we illustrate the distribution of rules with respect to the features, IOPSC, IOPHC, cardinality and length, and also IOPSC vs length. In the following, the *proportion* of mined rules having the various feature values is presented, to more evenly demonstrate the quality of performance over the three very different KGs.

The distribution of mined rules with respect to their IOPSC and IOPHC is shown in Figure 4. In the left-hand chart we observe a consistent decrease in the proportion of quality rules as the IOPSC increases. In the right hand chart we see a similar pattern for increasing IOPHC, but the decrease is not as consistent because there must be frequent relevant rules with many covered head instances.

The distribution of mined rules with respect to their cardinalities is shown in Figure 5. We observe that the largest proportion of rules has cardinality of 1, as expected, as they have the least stringent requirements to be met in the KG. We observe an expected decrease with greater cardinalities as they demand tighter restrictions to be satisfied.

The distribution of mined rules with respect to their lengths is shown in Figure 6. One might expect that as the length increases, the number of rules would increase since the space of possible rules grows and this is what we see.

For a concrete example of SHACL learning, we show the following three IOP rules mined from DBpedia in the experiments.

$$0.64, 0.01, 1 : < \text{dbo : type} > \_ < \text{db : Song} >(x) \rightarrow$$
$$< \text{dbo : album} >(x, z_1) \wedge < \text{dbo : recordLabel} >(z_1, y).$$
$$0.63, 0.01, 1 : < \text{dbo : type} > \_ < \text{db : Song} >(x) \rightarrow$$
$$< \text{dbo : producer} >(x, y).$$
$$0.41, 0.01, 2 : < \text{dbo : type} > \_ < \text{db : Song} >(x) \rightarrow$$
$$< \text{dbo : producer} >(x, y).$$

The numbers prefixing each IOP rule are the corresponding IOPSC, IOPHC, and Cardinality annotations respectively.
$< \text{dbo : type} > \_ < \text{db : Song} >(x)$ expresses x is a song. The first rule indicates x should belong to an album (z_1) that has y as record label. The second rule requires a song (x) to have at least one producer while the third rule requires a song to have at least two producers, and these two rules are distinguished by the cardinality annotation. As we discussed in 3.1, the third rule is more constraining than the second, so the confidence of the third rule is lower than the confidence of the second, based on the KG data.

The rules can be trivially rewritten (through a script that we developed) in SHACL syntax as follows.

```
:s1
a sh:NodeShape ;
sh:targetclass class:<dbo:type>_<db:Song>;
sh:property [ sh:minCount 1 ;
              sh:path :<dbo:album> ;
              [sh:Path
              :<dbo:recordLabel>; ]
              ] .
:s2
a sh:NodeShape ;
sh:targetclass class:<dbo:type>_<db:Song>;
sh:property [ sh:minCount 1 ;
              sh:path :<dbo:producer>; ] .
:s3
a sh:NodeShape ;
sh:targetclass class:<dbo:type>_<db:Song>;
sh:property [ sh:minCount 2 ;
              sh:path :<dbo:producer>; ] .
```

5.1.1. IOPSC vs IOPHC

Using rules found in the experiments, we further illustrate the practical meaning of the IOPSC and IOHC qualities. While IOPSC determines the confidence of a rule based on counting the proportion of target predicate instances for which the rule holds true in the KG, IOPHC indicates the proportion of rule consequent instances that are justified by target predicate instances in the KG, thereby indicating the relevance of the rule to the target. In Wikidata, all unary predicates are occupations such as singer or entrepreneur, so all the entities which have these types turn out to be persons even though there is no explicit person type in our KG. Thus, the occupations all have very similar IOP rules about each of them with high IOPSC and low IOPHC, like the following one, for example.

$$0.47, 0.01, 1 : < \text{occupation} > \_ < \text{singer} >(x) \rightarrow$$
$$< \text{country\_of\_citizenship} >(x, y).$$

On the other hand, for these unary occupation predicates there are also some IOP rules with high IOPHC that apply only to one specific unary predicate, like the

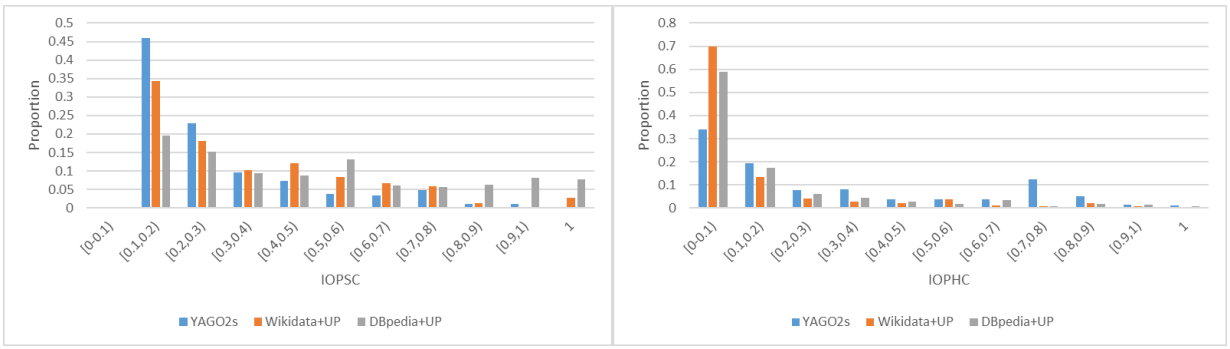

Fig. 4. Distribution of mined rules with respect to IOPSC and IOPHC

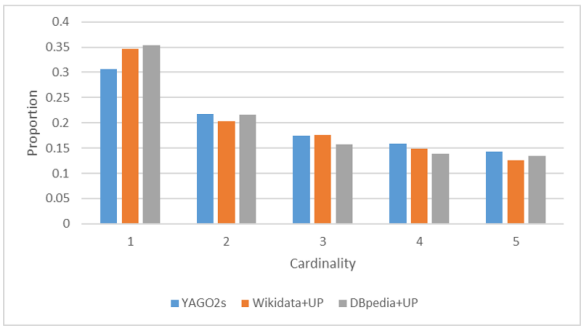

Fig. 5. Distribution of mined rules with respect to their cardinalities

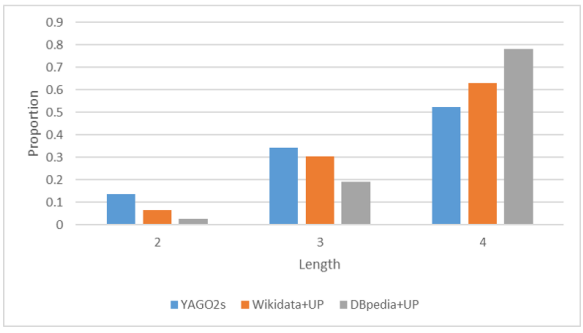

Fig. 6. Distribution of mined rules with respect to their lengths

following one.

$$0.15, 0.16, 5 : < \text{occupation} > \_ < \text{singer} > (x) \rightarrow$$
$$< \text{performer\_musical\_artist} > (x, y).$$

5.2. *Learning Trees from IOP Rules*

Now we turn to presenting results for the trees that are built based on the IOP rules discovered in the experiments. We report the characteristics of discovered trees in Table 3. We use a value of 0.1 for the *TreeSCMIN* parameter and show average TreeSC for each KG, along with the average number of branches in the trees and the average runtime building each tree. The number of trees for each KG is defined by the number of target predicates for which we have at least one IOP rule (see %SucRate in Table 2).

Table 3
Tree-learning performance of SHACLEARNER on benchmark KGs

Benchmark	TreeSC	#branches	Time (hours)
YAGO2s	0.13	21	0.06
Wikidata+UP	0.19	21	0.1
DBpedia+UP	0.2	88	0.14

The results show the running time for aggregating IOP rules into trees is lower than the initial IOP mining time by a factor greater than 10. If, on the other hand, we wanted to discover such complex shapes from scratch it would be exhaustively time consuming due to sensitivity of rule learners to the maximum length of rules. The number of potential rules in the search space grows exponentially with regards to the maximum number of predicates of the rules. The average number of branches in the mined trees are 50%, 31%, and 56% of the corresponding number of mined rules respectively from Table 2. Hence, by imposing the additional tree-shaped constraint over the basic IOP-shaped constraint, at least 44% of IOP rules are pruned.

For an example of tree shape learning, in the following, we show a fragment of a 39-branched tree mined

from DBpedia by aggregating IOP rules in the experiments.

$$\begin{aligned} 0.13 : <\text{dbo}:\text{type}>\_<\text{db}:\text{Song}>(x) \rightarrow \\ 1 : <\text{dbo}:\text{album}>(x, z_1) \wedge \\ <\text{dbo}:\text{recordLabel}>(z_1, y_1) \wedge \\ 2 : <\text{dbo}:\text{album}>(x, z_2) \wedge \\ <\text{dbo}:\text{producer}>(z_2, y_2) \wedge \\ 1 : <\text{dbo}:\text{album}>(x, z_3) \wedge \\ <\text{dbo}:\text{genre}>(z_3, y_3) \wedge \\ 3 : <\text{dbo}:\text{artist}>(x, z_4) \wedge \\ <\text{dbo}:\text{musicalArtist}>(z_4, y_4). \end{aligned}$$

Here, the first annotation value (0.13) presents the SC of the tree and the subsequent values at the beginning of each branch indicate the branch cardinality. This tree can be read as saying that a song has an album with a record label, and an album with two producers, and an album with a genre, and an artist who is a musical artist.

As can be seen here, there remains an opportunity for further simplification and explanatory value by unifying additional variables occurring in predicates shared over multiple branches. We plan to investigate this potential post-processing step in future work.

6. Conclusion

In this paper we propose a method to learn SHACL shapes from KGs as a way to describe KG patterns, to validate KGs, and also to support new data entry. For entities that satisfy target predicates, our shapes describe conjunctive paths of constraints over properties, enhanced with minimum cardinality constraints.

We reduce the SHACL learning problem to learning a novel kind of rules, Inverse Open Path rules (IOP). We introduce rule quality measures IOPSC, IOPHC and Car which augment the rules. IOPSC effectively extends SHACL with shapes, representing the quantified uncertainty of a candidate shape to be selected for interestingness or for KG verification. We also propose a method to aggregate learnt IOP rules in order to discover more complex shapes, trees.

The shapes support efficient and interpretable human validation in a depth-first manner and are employed in an editor *Schímatos* [21] for manual knowledge graph completion. The shapes can be used to complete information triggered by entities with only a type or class declaration by automatically generating dynamic data entry forms. In this manual mode, they can also be used more traditionally to complete missing facts for a target predicate, as well as other predicates related to the target, while enabling the acquisition of facts about entities that are entirely missing from the KG.

To learn such rules we adapt an embedding-based Open Path rule learner (OPRL) by introducing the following novel components: (1) we propose IOP rules which allow us to mine rules with free variables with one predicate forming the body and a chain of predicates as the head, while keeping the complexity of the learning phase manageable; (2) we introduce tree shapes that are built from the IOP rules for more expressive patterns; and (3) we propose an efficient method to evaluate IOP rules and trees by exactly computing the quality measures of each rule using fast matrix and vector operations.

Our experiments show that SHACLEARNER can mine IOP rules of various lengths, cardinalities, and qualities from three massive real-world benchmark KGs including Yago, Wikidata and DBpedia.

In future work we will validate the shapes we learn with SHACLEARNER via formal human-expert evaluation and further extend the expressivity of the shapes we can discover. Another future work will be to extend the SHACLearner algorithm through MapReduce model to handle extremely massive KGs with tens of billions of facts.

Acknowledgments

The authors acknowledge the support of the Australian Government Department of Finance and the Australian National University for this work.

References

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z.G. Ives, DBpedia: A Nucleus for a Web of Open Data, in: *ISWC*, Lecture Notes in Computer Science, Vol. 4825, Springer, 2007, pp. 722–735.

[2] F.M. Suchanek, G. Kasneci and G. Weikum, Yago: a core of semantic knowledge, in: *WWW*, ACM, 2007, pp. 697–706.

[3] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson and J. Taylor, Industry-Scale Knowledge Graphs: Lessons and Challenges, *Commun. ACM* **62**(8) (2019), 36–43–.
[4] E.S. Callahan and S.C. Herring, Cultural bias in Wikipedia content on famous persons, *Journal of the American Society for Information Science and Technology* **62**(10) (2011), 1899–1915.
[5] D. Vrandecic and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Commun. ACM* **57**(10) (2014), 78–85.
[6] H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL), 2017. https://www.w3.org/TR/shacl/.
[7] Z. Huo, K. Taylor, X. Zhang, S. Wang and C. Pang, Generating multidimensional schemata from relational aggregation queries, *World Wide Web* **23** (2020).
[8] P. Ghiasnezhad Omran, K. Taylor, S. Rodríguez Méndez and A. Haller, Towards SHACL Learning from Knowledge Graphs, in: *{ISWC2020} Posters & Demonstrations*, CEUR Proceedings, 2020, To appear..
[9] H. Knublauch, D. Allemang and S. Steyskal, SHACL Advanced Features, 2017. https://www.w3.org/TR/shacl-af/.
[10] L. Galárraga, C. Teflioudi, K. Hose and F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE+, *The International Journal on Very Large Data Bases* (2015), 707–730. ISBN 1066-8888.
[11] L. Bellomarini, E. Sallinger and G. Gottlob, The Vadalog system: Datalogbased reasoning for knowledge graphs, in: *VLDB*, Vol. 11, 2018, pp. 975–987. ISSN 21508097.
[12] W. Fan, C. Hu, X. Liu and P. Lu, Discovering graph functional dependencies, in: *SIGMOD*, ACM, 2018, pp. 427–439. ISSN 07308078. ISBN 9781450317436.
[13] P. Ghiasnezhad Omran, K. Taylor, S. Rodríguez Méndez and A. Haller, Active Knowledge Graph Completion, Technical Report, ANU Research Publication, 2020.
[14] P.G. Omran, K. Wang and Z. Wang, Scalable Rule Learning via Learning Representation, in: *IJCAI*, 2018, pp. 2149–2155. ISBN 9780999241127.
[15] Y. Chen, D.Z. Wang and S. Goldberg, ScaLeKB: scalable learning and inference over large knowledge bases, *The International Journal on Very Large Data Bases* (2016), 893–918.
[16] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, in: *VLDB*, Vol. 1215, 1994, pp. 487–499.
[17] K. Taylor, Generalization by absorption of definite clauses, *The Journal of Logic Programming* **40**(2–3) (1999), 127–157.
[18] S. Staworko, I. Boneva, J.E. Labra Gayo, S. Hym, E.G. Prud'hommeaux and H. Solbrig, Complexity and Expressiveness of ShEx for RDF, in: *ICDT*, 2015, pp. 195–211. http://linkeddata.org/.
[19] M. Nickel, L. Rosasco and T. Poggio, Holographic Embeddings of Knowledge Graphs, in: *AAAI*, 2016, pp. 1955–1961.
[20] M. Nickel, V. Tresp and H.-P. Kriegel, A three-way model for collective learning on multi-relational data, in: *ICML*, 2011, pp. 809–816.
[21] J. Wright, S. Rodríguez Méndez, A. Haller, K. Taylor and P. Omran, Schimatos: a SHACL-based Web-Form Generator for Knowledge Graph Editing, in: *ISWC*, 2020.
[22] N. Mihindukulasooriya, M. Rifat, A. Rashid, G. Rizzo, R. García-Castro, O. Corcho and M. Torchiano, RDF Shape Induction using Knowledge Base Profiling, in: *Annual {ACM} Symposium on Applied Computing, {SAC}*, Vol. 8, 2018, p. pages. ISBN 9781450351911.
[23] D. Fernández-Álvarez, H. García-González, J. Frey, S. Hellmann and J.E.L. Gayo, Inference of latent shape expressions associated to DBpedia ontology, in: *ISWC Posters*, Vol. 2180, 2018. ISSN 16130073.
[24] B. Spahiu, A. Maurino and M. Palmonari, Towards improving the quality of knowledge graphs with data-driven ontology patterns and SHACL, in: *Workshop on Ontology Design and Patterns*, Vol. 2195, 2018, pp. 52–66. ISSN 16130073.
[25] I. Boneva, J. Dusart, D.F. Álvarez and J.E. Labra Gayo, Shape designer for ShEx and SHACL constraints, in: *ISWC Posters*, Vol. 2456, 2019, pp. 269–272. ISSN 16130073.
[26] A. Cimmino, A. Fernández-Izquierdo and R. García-Castro, Astrea: Automatic Generation of SHACL Shapes from Ontologies, in: *ESWC*, Vol. 12123 LNCS, 2020, pp. 497–513. ISSN 16113349. ISBN 9783030494605.
[27] H.J. Pandit, D. O'Sullivan and D. Lewis, Using ontology design patterns to define SHACL shapes, in: *CEUR Workshop Proceedings*, Vol. 2195, 2018, pp. 67–71. ISSN 16130073.
[28] J.-E. Labra-Gayo, E. Prud'hommeaux, H. Solbrig and I. Boneva, Validating and describing linked data portals using shapes, *CoRR* (2017).
[29] I. Boneva, J.E. Labra Gayo and E.G. Prud'hommeaux, Semantics and validation of shapes schemas for RDF, in: *ISWC*, Vol. 10587 LNCS, 2017, pp. 104–120. ISSN 16113349. ISBN 9783319682877. http://www.w3.org/Submission/spin-modeling/.
[30] P. Pareti, G. Konstantinidis, T.J. Norman and S. Murat, SHACL Constraints with Inference Rules, in: *ISWC*, 2019.
[31] J. Corman, J.L. Reutter and O. Savković, Semantics and validation of recursive SHACL, in: *ISWC*, Vol. 11136 LNCS, 2018, pp. 318–336. ISSN 16113349. ISBN 9783030006709.
[32] Anonymised, Active Knowledge Graph Completion, in: *IJCAI(Submitted/Under review)*, 2020.
[33] Authors suppressed, *Schímatos*: A SHACL-based Web-Form Generator for Knowledge Graph Editing, 2020, submitted to ISWC2020.