Constraint Rules in Knowledge Graphs

Pablo Mollá Chárlez

February 9, 2025

Contents

1 Rule Evaluation Metrics

1.1 Support

Support of a rule R indicates how many facts in the knowledge base K the rule correctly accounts for. Formally:

$$\text{support}(R) = \left|\left\{ p \mid (K \wedge R \vDash p) \wedge p \in K \right\}\right|.$$

As rules become more specialized (adding more conditions in the premise), support often decreases because fewer facts match. In practice, a rule with very low support may be discarded (its applicability is minimal). Support addresses the **relevance of the rule** (how many facts it covers).

1.2 Support of rule in sets

The coverage of a rule R in a given set S (Support of the rule in set S) is defined as:

$$\text{support(R,S)} = |p : (K \wedge R \vDash p) \wedge p \in S|$$

1.3 Confidence

Confidence of a rule R measures the proportion of the rule's predictions that are actually true in K:

$$\text{confidence}(R) = \frac{\text{support}(R)}{\text{support}(R) + |\,\text{cex}(R)\,|},$$

where cex(R) is the set of counterexamples-predictions the rule makes that are not found in K. Confidence represents the **accuracy of the rule** (fraction of correct predictions). Because knowledge graphs typically lack explicit negative facts, we need assumptions to infer counterexamples:

- Closed World Assumption (CWA): Anything not stated is considered false. If the rule predicts a fact not in K, that prediction counts as a counterexample.

$$\text{cex\_CWA} = |\{(x,y) \in B : (x,y) \notin H\}| \quad \text{where} \quad R : B \longrightarrow H$$

- Open World Assumption (OWA): Anything not stated is unknown. A missing fact is not necessarily a counterexample; it could still be true but not yet recorded.

$$\text{cex\_OWA} = |B \cap \neg H| \quad \text{where} \quad R : B \longrightarrow H$$

- Partial Completeness Assumption (PCA): If any relationship r is known for a subject s, then all such r-relationships for s are expected to be recorded. This assumption tries to balance the extremes of CWA and OWA, treating some parts of the knowledge base as "closed" while leaving others "open."

$$\text{cex\_PCA} = |\{(x,y) \in B : x \in H \wedge (x,y) \notin H\}| \quad \text{where} \quad R : B \longrightarrow H$$

1.4 Head Coverage

Head coverage for a rule $B \Rightarrow r(x,y)$ captures what fraction of the known facts for the predicate r (in the knowledge base K) is "explained" by that rule. Formally:

$$\text{hc}(B \Rightarrow r(x,y)) = \frac{\text{support}(B \Rightarrow r(x,y))}{|\{(x,y) \mid r(x,y) \in K\}|}.$$

Where support$(B \Rightarrow r(x,y))$ is the number of true pairs (x,y) for which B holds and $r(x,y)$ is actually in K, and $|\{(x,y) \mid r(x,y) \in K\}|$ is the total count of true instances of r in K.

Suppose we have a knowledge base about people and their living locations, and we are examining the rule:

$$\texttt{marriedTo}(x,y),\ \texttt{livedIn}(x,z) \Rightarrow \texttt{livedIn}(y,z).$$

In K, there are 10 known facts of the form `livedIn`(y, z). By checking the premises (`marriedTo`(x, y) and `livedIn`(x, z)), we find that the rule correctly predicts 4 of those 10 facts. In other words, the rule's support is 4 (it matches 4 existing "livedIn" facts). So the head coverage is:

$$\text{hc} = \frac{\text{support}}{\text{total known facts for } r} = \frac{4}{10} = 0.4.$$

Thus, 40% of the `livedIn`(y, z) relationships in the KB are "covered" (i.e., explained) by that rule.

1.5 Weight of a Rule

The weight of a rule $w(R)$ is a composite metric that tries to balance how well R covers the true facts in the knowledge graph (akin to recall) and how many non-facts or potential negatives it predicts (related to precision). One way to define it is:

$$w(R) = \alpha\Big(1 - \frac{\text{support}(R, \mathcal{G})}{|\mathcal{G}|}\Big) + \beta\Big(\frac{\text{support}(R, \mathcal{V})}{|\mathcal{V}|}\Big),$$

where

- $\mathcal{G}$ is the set of grounded facts (true statements) in the knowledge graph (finite).
- $\mathcal{V}$ is a set representing potential negative or unobserved facts (which can be extremely large). We can also use $|U(R, \mathcal{V})|$, $U(R, \mathcal{V}) \subseteq \mathcal{V}$ is chosen so that $\text{support}(R, \mathcal{V}) \subseteq U(R, \mathcal{V})$, ensuring proper normalization.
- $\text{support}(R, \mathcal{G})$ counts how many true facts R covers, and $\text{support}(R, \mathcal{V})$ counts how many "negative" or unobserved facts R covers.
- α, β let you tune how much each term contributes.

In words, the first term $\big(1 - \frac{\text{support}(R,\mathcal{G})}{|\mathcal{G}|}\big)$ decreases when R correctly covers more real facts (increasing recall), whereas the second term $\frac{\text{support}(R,\mathcal{V})}{|\mathcal{V}|}$ increases if R also covers many negative or unobserved facts (worsening precision).

For instance, suppose that $\mathcal{G}$ has 100 known true facts relevant to R. The rule covers 30 of them: $\text{support}(R, \mathcal{G}) = 30$. Moreover, $\mathcal{V}$ has 300 potential negative facts; the rule covers 45 of those: $\text{support}(R, \mathcal{V}) = 45$. We choose $\alpha = 0.5$ and $\beta = 0.5$. We would get:

$$w(R) = 0.5\Big(1 - \frac{30}{100}\Big) + 0.5\Big(\frac{45}{300}\Big) = 0.5\big(0.70\big) + 0.5\big(0.15\big) = 0.35 + 0.075 = 0.425.$$

A lower weight in this scheme would typically mean the rule either fails to cover enough true facts or covers too many negatives-hence is less desirable.

2 SHACL: Shape-Based Constraint Language

SHACL (Shapes Constraint Language) is a **W3C Recommendation (2017)** that provides a standard schema language for RDF. It allows one to define validation constraints on classes, nodes, properties, and literals by expressing them directly in RDF. The validation process takes two inputs-a data graph (the RDF dataset to be validated) and a shape graph (the RDF-based definitions of the constraints)-and produces a validation report. A shape graph can contain multiple shapes, each described by RDF triples using the SHACL vocabulary (sh:). Shapes come in two forms: **Node Shapes**, which apply constraints directly to a node, and **Property Shapes**, which specify constraints on the nodes reached via a particular property path. The goal of SHACL is thus to enforce and verify that RDF data conforms to the intended structure and semantics.

2.1 SHACL Constraints

Table 1: SHACL Constraint Forms

Constraint	Shape Type	Description
Node Kind and Datatype Constraints		
`sh:class c`	Node / Property	Target nodes must be instances of class `c`. Multiple `sh:class` values are conjunctive.
`sh:datatype t`	Node / Property	Target nodes must be literals of datatype `t`; at most one `sh:datatype`.
`sh:nodeKind k`	Node / Property	At most one `sh:nodeKind` declaration for the shape.
Numeric Constraints		
`sh:minInclusive / sh:minExclusive /` `sh:maxInclusive / sh:maxExclusive`	Node / Property	All target nodes must satisfy the numeric comparison (often combined with `sh:datatype xsd:integer` or other numeric types).
String Length Constraints		
`sh:minLength n; sh:maxLength m`	Node / Property	Target nodes must be literals or IRIs. One `sh:minLength` and one `sh:maxLength` allowed.
Regular Expression Constraints		
`sh:pattern regex; sh:flags flag`	Node / Property	Target nodes must match the given `regex`; `flag` often includes `"i"` (case-insensitive).
Language Tag Constraints		
`sh:languageIn (lang_list)`	Node / Property	All target nodes must have a language tag in the given list.
`sh:uniqueLang true`	Property shape	Ensures at most one value per language tag among value nodes.
Cardinality Constraints		
`sh:maxCount n; sh:minCount m`	Property	On `sh:path p`, bounds the number of value nodes from `p`. `n` and `m` must be `xsd:integer`.
Fixed Value Constraints		
`sh:hasValue v`	Property	All values from the path must be equal to `v`.
`sh:in (v1 v2 ... vn)`	Property	All value nodes must be one of `v1, v2, ..., vn`.
Property Pair Constraints		
`sh:equals r`	Property	Values from `p` must match values from `r` exactly.
`sh:disjoint r`	Property	Values from `p` must not overlap values from `r`.
`sh:lessThan r`	Property	`max`(Vp) < `min`(Vr).
`sh:lessThanOrEquals r`	Property	`max`(Vp) ≤ `min`(Vr).
Logical Operators		
`sh:not [shape]`	Node / Property	Negates the constraints defined in the nested shape.
`sh:and (shapes...)`	Node / Property	Target node must satisfy all listed shapes.
`sh:or (shapes...)`	Node / Property	Target node must satisfy at least one listed shape.
`sh:xone (shapes...)`	Node / Property	Target node must satisfy exactly one listed shape.

2.2 SHACK's Validation Steps

1. Identify Target Nodes. Determine which nodes the constraint applies to (e.g., via **sh:targetClass**, **sh:targetSubject** or **sh:targetObjectsOf**).

2. Define the Constraint Condition in SPARQL. Typically a SELECT query that returns any nodes violating the rule.

3. Check for Violations. If the query returns any results, the graph fails to conform. Otherwise, it passes.

This SPARQL-based method is quite flexible—constraints can incorporate arbitrary graph patterns and even aggregations. Additional Constraints Expressible in SHACL include:

- Unary Inclusion Dependencies: Restricting which nodes can appear as property values.

- (Conditional) Inclusion Dependencies (IND/CIND): Ensuring that values in one property subset must appear in another, potentially under certain conditions.

- Relation Functionality: Enforcing a property can have at most one value (i.e., functional).

- Inverse Relation Functionality: Ensuring that the property's inverse is functional (each value node points back to at most one subject).