TRIC: A Triples Corrupter for Knowledge Graphs

Asara Senaratne(✉), Pouya Ghiasnezhad Omran, Peter Christen, and Graham Williams

School of Computing, The Australian National University, Canberra, Australia
{asara.senaratne,p.g.omran,peter.christen,graham.williams}@anu.edu.au

Abstract. We study the problem of corrupting triples in Knowledge Graphs (KG) for the purpose of assisting anomaly detection and error detection techniques developed for KG quality enhancement. Our goal is to provide users with the highest possible level of control over the triples corruption process, and simultaneously develop a solution that scales to large KGs. Hence, we introduce TRIC, an approach for corrupting triples considering both semantic and type information to generate errors in a KG. In this paper, we discuss how the problem of triples corruption is challenging, and different from existing negative sampling techniques used in link prediction. To the best of our knowledge, there is no approach in the literature dedicated for generating abnormal triples in KGs to support anomaly detection and error detection tasks.

Keywords: Anomalous triples · Erroneous triples · Knowledge Graph quality enhancement · Negative sampling

1 Introduction

Large scale Knowledge Graphs (KG) such as YAGO, DBpedia, and Wikidata are published and widely used at present. However, these KGs are far from perfect and have many quality issues. For example, these KGs may contain inaccurate, or abnormal triples which limit the credibility and further utility of the KGs [8]. The domain of data quality assessment in the field of traditional relational data roots in the literature, and has recently attracted experts from the domain of knowledge representation. Inspired by our recent work in anomaly detection [4, 5], we consider the problem of empirically evaluating anomaly detection and error detection algorithms developed for quality enhancement of KGs, in the absence of labelled data.

The essence of this study is the absence of a standardized technique to systematically corrupt triples in a KG considering both semantics and entity types of triples. As none of the real-world KGs contain labelled data (ground truth data), developers of existing KG quality enhancement techniques adopt impromptu means of corrupting triples for their evaluation purposes [1]. This further demotivates the development of unsupervised and human-independent

C. Pesquita et al. (Eds.): ESWC 2023, LNCS 13998, pp. 117–122, 2023.
https://doi.org/10.1007/978-3-031-43458-7_22

approaches for anomaly and error detection in KGs. As a result, many of the existing KG error detection techniques depend on external sources to validate their proposed approaches during experimental evaluation [6]. However, this is a costly process, and not every KG has a gold standard [2].

To overcome the stated problem, we introduce a triples corrupter for knowledge graphs (TRIC). It is unsupervised, and can introduce a wide range of errors (inaccurate, unusual, contradicting, invalid, redundant triples, and data quality errors) in a KG by corrupting either of subject, predicate, or object, whilst considering type and semantic information. This is in contrast to negative sampling, where negative triples are generated by corrupting a known positive triple by replacing either subject, predicate, or object. TRIC introduces different types of errors by corrupting a set of randomly chosen triples in a KG, where it can corrupt both entity-based and literal-based triples. Furthermore, TRIC allows the user to have complete control over the triples corruption process. While TRIC operates on a default setting, if required, a user can determine the types of errors to be introduced, the percentage of errors from each error type to be introduced, and the overall percentage of triples to be corrupted such that the KG has 10%, 20%, and so on of its triples corrupted.

Our Contribution: We introduce TRIC, a pioneer in generating errors in a KG to support anomaly and error detection techniques. The novelty of TRIC is its ability to corrupt both entity-based and literal-based triples by considering the semantics, data types, and entity types of the associated relations, literals, and entities, respectively. TRIC is automated, and requires no user assistance. Furthermore, the corruptions of TRIC will introduce inaccurate, unusual, contradicting, invalid, redundant triples, and data quality errors in a KG.

2 Related Work

There are many techniques proposed in the literature for KG evaluation. One common evaluation strategy is to use a partial gold standard. In this methodology, a subset of graph entities or relations are selected and labeled manually. Another evaluation strategy is to use the given KG itself as a test dataset, which is known as silver standard evaluation. For retrospective evaluations, the output of a given approach is given to human judges for annotation, who then label identified errors as correct or incorrect [2]. While it is costly and time consuming to perform manual evaluation involving human experts, obtaining a gold standard KG for every KG is infeasible. Hence, most existing KG evaluation approaches depend on silver standard evaluation, where the developers of these these techniques synthetically generate errors. Negative sampling is one such technique used to introduce corrupted triples [2].

Negative sampling techniques [7] generate negative triples by corrupting a known positive triple $(s, p, o) \in G$ by replacing either s, p, or o. Usually, the corruption of relations (predicate) is omitted as the evaluation of KG embedding models on the link prediction task only considers the suitability of head (subject) prediction and tail (object) prediction, but not relation (predicate) prediction. Due to the simplicity in thus generated corrupted triples (as this approach has no specific interest in corrupting the semantics nor the data types of literals), and given that real-world errors and anomalies in KGs are much complex [5], there exists the requirement of a triples corruption approach dedicated for the evaluation of anomaly detection and error detection techniques developed for KGs.

3 Methodology

Even though large scale KGs such as YAGO-4 contains semantic constraints in the form of SHACL to keep data clean [3], such a validation layer is not often available in custom built KGs. Hence, during the designing process of TRIC, we considered all possible quality issues that can exist in such real-world KGs, where there is no adoption of constraints such as SHACL or ShEx.

We consider a directed edge-labelled KG, $G = (V, E)$ containing a set of nodes (or vertices) V, and a set of labelled edges E connecting these vertices. Each edge $e \in E$, together with the connecting nodes are considered as a triple t. A triple (also named as a triplet) contains the three elements subject (head) $s \in S$, predicate (relation) $p \in P$, and object (tail) $o \in O$. A triple t is denoted as (s, p, o), where $(s, o) \in V$. While a subject $s \in t$ is considered as a real-world entity, an object $o \in t$ can either be an entity n, or a literal l (an entity's attribute value) [5]. We refer triples of the form (s, p, n) as entity-based triples, and (s, p, l) as literal-based triples.

Table 1 provides the fifteen types of errors TRIC can generate considering both (s, p, n) and (s, p, l) triples. Furthermore, algorithm 1 provides the pseudocode of TRIC under the default setting (assuming there are no user inputs). To replace an element of t, TRIC extracts the new content within the KG itself. While TRIC randomly selects the triples for corruption based on the percentage of errors required, the entities and predicates to use as replacements are also selected randomly after analyzing the type information associated with the entities. We obtain entity type information via inferencing. Implementation of TRIC is available on our GitHub repository[1].

[1] https://github.com/AsaraSenaratne/SEKA.

Table 1. Types of errors TRIC can generate in a KG.

Error Types	Original Triple/Status	Corrupted/New Triple
(1) Change s while preserving entity type.	<personA, isMarriedTo, personB>	<personC, isMarriedTo, personB>
(2) Change o while preserving the entity type	<personA, isMarriedTo, personB>	<personA, isMarriedTo, personD>
(3) Change p while preserving the predicate type	<personA, isMarriedTo, personB>	<personA, hasChild, personB>
(4) Change both (s, o) while preserving the entity types.	<personA, isMarriedTo, personB>	<personE, isMarriedTo, personF>
(5) Change s while also changing the entity type.	<personA, isMarriedTo, personB>	<moon, isMarriedTo, personB>
(6) Change o while also changing the entity type.	<personA, isMarriedTo, personB>	<personA, isMarriedTo, london>
(7) Change both (s, o) while also changing their entity types.	<personA, isMarriedTo, personB>	<moon, isMarriedTo, london>
(8) Change p while also changing its semantic meaning. That is replace predicates used for a person with a predicate that is not used for a person.	<personA, isMarriedTo, personB>	<personA, livesIn, personB>
(9) Add an edge between two entities, such that there is a type inconsistency in the predicate introduced.	The entities *personP* and *personQ* have no relationship	<personP, produced, personQ>
(10) Add an edge between two entities, such that there is no type inconsistency in the predicate introduced.	The entities *personP* and *personQ* have no relationship	<personP, hasChild, personQ>
(11) Introduce semantically incorrect literals to entities.	Add *DateOfBirth* to a *location* entity	<london, hasDateOfBirth, "10/10/1990" >
(12) Introduce semantically correct literals to entities (avoiding duplicates).	Add *hasWebsite* to a *person*	<personA, hasWebsite, "www.a.com"
(13) Corrupt t such that the literal value changes to a value of a different data type.	A date gets changed to a name.	<personA, hasDateOfBirth, "Sarah">
(14) Corrupt t by removing the literal value, thus generating a triple with a missing literal.	<personA, hasDateOfBirth, "12/02/1989">	<personA, hasDateOfBirth, "">
(15) Corrupt t such that the new literal value is a duplicate of an existing literal value of the same entity under consideration.	<personA, hasDateOfBirth, "12/02/1989">	<personA, hasDateOfBirth, "12/02/1989">

ALGORITHM 1: Error generation steps of TRIC.

Input: G: The KG to be subjected for triples corruption.
Output: $\mathbf{G}_c$: The KG with corrupted triples.

```
    //If no user input, default percentage of errors is 1%
1:  p ← defineAnomalyPercentage()
    //Get total number of triples in G.
2:  sizeG ← getSize(G)
    //Find count of errors required from the percentage
3:  errorscount ← sizeG * p
    //Number of errors required from each error type (from 15 error types)
4:  errorseach ← round(errorscount/15)
    //Select the triples to corrupt
5:  triplestocorrupt ← getTriplesRandomly(G)
    //Remove triples to corrupt from G
6:  reducedG = Gc ← removeTriples(G, triplestocorrupt)
    //Iterate over the types of errors TRIC can generate
7:  for error in errortypes:
8:      count = 0
        //Iterate to generate specified number of errors from each error type.
9:      while count <= errorseach:
            //Add corrupted triples to graph
10:         Gc+ = generateErrors(error, reducedG, triplestocorrupt)
            //Increment count
11:         count + +
12: return Gc
```

4 Conclusion and Future Work

In this paper, we introduced TRIC, a triples corrupter for Knowledge Graphs (KG) that is aimed at supporting the KG quality enhancement tasks; anomaly detection and error detection. Even though there exists negative sampling techniques dedicated for link prediction, the triples corrupted via negative sampling cannot fully exercise the capabilities of anomaly and error detection approaches, as negative sampling does not consider semantic and type information of the predicates, literals, and entities. As future work, we aim to publish TRIC as a Python library for the use of the wider research community.

References

1. Jia, S., Xiang, Y., Chen, X., Wang, K.: Triple trustworthiness measurement for knowledge graph. In: The World Wide Web Conference, pp. 2865–2871 (2019)
2. Paulheim, H.: Knowledge graph refinement: a survey of approaches and evaluation methods. Semant. Web **8**(3), 489–508 (2017)

3. Pellissier Tanon, T., Weikum, G., Suchanek, F.: YAGO 4: a reason-able knowledge base. In: Harth, A., et al. (eds.) ESWC 2020. LNCS, vol. 12123, pp. 583–596. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49461-2_34
4. Senaratne, A., Christen, P., Williams, G., Omran, P.G.: Unsupervised identification of abnormal nodes and edges in graphs. JDIQ **15**(1), 1–37 (2022)
5. Senaratne, A., Omran, P.G., Williams, G., Christen, P.: Unsupervised anomaly detection in knowledge graphs. In: IJCKG, pp. 161–165 (2021)
6. Wang, Y., Ma, F., Gao, J.: Efficient knowledge graph validation via cross-graph representation learning, pp. 1595–1604. ACM, New York (2020)
7. Xie, R., Liu, Z., Lin, F., Lin, L.: Does william shakespeare really write hamlet? knowledge representation learning with confidence. In: AAAI, vol. 32 (2018)
8. Xue, B., Zou, L.: Knowledge graph quality management: a comprehensive survey. In: TKDE (2022)