What is Normal, What is Strange, and What is Missing in a Knowledge Graph: Unified Characterization via Inductive Summarization

Caleb Belth
University of Michigan
cbelth@umich.edu

Jilles Vreeken
CISPA Helmholtz Center for Information Security
jv@cispa.saarland

Xinyi Zheng
University of Michigan
zxycarol@umich.edu

Danai Koutra
University of Michigan
dkoutra@umich.edu

ABSTRACT

Knowledge graphs (KGs) store highly heterogeneous information about the world in the structure of a graph, and are useful for tasks such as question answering and reasoning. However, they often contain errors and are missing information. Vast research in KG refinement has worked to resolve these issues, tailoring techniques to either detect specific types of errors or complete a KG.

In this work, we introduce a unified solution to KG characterization by formulating the problem as an unsupervised KG summarization with a set of inductive, soft rules, which describe what is normal in a KG, and thus can be used to identify what is abnormal, whether it be strange or missing. Unlike first-order logic rules, our rules are labeled, rooted graphs, i.e., patterns that describe the expected neighborhood around a (seen or unseen) node, based on its type, and information in the KG. Stepping away from the traditional support/confidence-based rule mining techniques, we propose KGist, Knowledge Graph Inductive Summarization, which learns a summary of inductive rules that best compress the KG according to the Minimum Description Length principle—a formulation that we are the first to use in the context of KG rule mining.

We apply our rules to three large KGs (NELL, DBpedia, and Yago), and tasks such as compression, various types of error detection, and identification of incomplete information. We show that KGist outperforms task-specific, supervised and unsupervised baselines in error detection and incompleteness identification, (identifying the location of up to 93% of missing entities—over 10% more than baselines), while also being efficient for large knowledge graphs.

ACM Reference Format:
Caleb Belth, Xinyi Zheng, Jilles Vreeken, and Danai Koutra. 2020. What is Normal, What is Strange, and What is Missing in a Knowledge Graph: Unified Characterization via Inductive Summarization. In Proceedings of The Web Conference 2020 (WWW ’20), April 20–24, 2020, Taipei, Taiwan, ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3366423.3380189

1 INTRODUCTION

Knowledge graphs (KGs), such as NELL [9], DBpedia [5], and Yago [46], store collections of entities and relations among those entities.

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution. WWW ’20, April 20–24, 2020, Taipei, Taiwan © 2020 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC-BY 4.0 License. ACM ISBN 978-1-4503-7023-3/20/04. https://doi.org/10.1145/3366423.3380189

Figure 1: KGist summarizes a KG (left) by finding patterns that can be interpreted as rules (right). For instance, the rule that books are written by authors, who are born in countries, which holds in two out of three cases in this KG (Frankenstein is missing an author), correctly describes books in general. However, the opposite pattern does not: while Leo Tolstoy writes books, Emily Dickinson writes poems. The summary of rules characterizes what is normal in a KG, while simultaneously revealing what is strange and missing, such as the erroneous and missing edges around Frankenstein.
to the parts of the world not captured by $G$. With this characterization of the norm, we can also identify what is strange in and what is missing from $G$: the parts of the graph that violate the rules or remain unexplained by the summary. These strange parts of the graph may be genuine exceptions, errors, or missing information. To solve the problem, we propose KGist, an information-theoretic approach that serves as a unified solution to summarization and various KG refinement tasks, which have traditionally been viewed independently.

Our main contributions are summarized as follows:

- **Problem Formulation.** Rather than targeting a specific refinement task (e.g., link prediction), we unify various refinement tasks by joining the problems of refinement and unsupervised summarization, and introduce the notion of inductive summarization with soft rules that plausibly generalize beyond the KG. § 3

- **Expressive rules.** While current methods (§ 2) learn first-order logic rules that have single-element consequences, which predict single edges, our rules are labeled, rooted graphs that are recursively defined, allowing them to describe arbitrary graph structure around a node (i.e., they can have complex consequences). Our formulation of rules takes a step towards treating knowledge graphs as graphs—something often overlooked in KG refinement [35]. § 3

- **MDL-based approach.** We introduce KGist, an unsupervised, information-theoretic approach that identifies rules via the Minimum Description Length (MDL) principle [38], going beyond the support/confidence framework of prior work. § 4

- **Experiments on real KGs.** We perform extensive experiments on large KGs (NELL, DBpedia, Yago), and diverse tasks, including compression, various types of error detection, and identifying the absence of nodes. We show that KGist learns orders of magnitude fewer rules than current methods, allowing KGist to be efficient and effective at diverse tasks. KGist identifies the location of 76-93% of missing entities—over 10% more than baselines. § 5

Our code and data are available at https://github.com/GemsLab/KGist.

2 RELATED WORK

2.1 Knowledge Graph Refinement

KG refinement attempts to resolve erroneous or missing information [35, 36]. Next, we discuss the three most relevant categories of refinement techniques (although other methods exist, such as crowd-sourcing-based methods [23]).

2.1.1 Rule-mining-based Refinement. These approaches are reminiscent of association rule mining [2]. AMIE+ [18] introduces an altered confidence metric based on the partial completeness assumption, according to which, if a particular relationship of an entity is known, then all relationships of that type for that entity are known (as opposed to the open-world assumption, which assumes that an absent relationship could either be missing or not hold in reality). AMIE+ [17] is optimized to scale to larger KGs, and Tanon et al. [47] seek to acquire and use counts of edges to measure the incompleteness of KGs. Other, non-rule-mining-based methods have also been proposed for measuring KG quality [22, 37]. A supervised approach that augments AMIE+ [16] takes example complete and incomplete assertions (e.g., crowd-sourced) as training data, and predicts completeness of predicate types observed during training. These works focus on refinement and find Horn rules on binary predicates. In contrast, we focus on summarization, and our rules can be applied to a node, knowing only its type. Also, we go beyond the support/confidence framework, which treats KGs as a table of transactions, and take a graph-theoretic view instead. One work that does take a graph-theoretic view learns rules in a bottom-up fashion by sampling paths from the KG, but the rules are constrained to be path-based Horn-rules [28]. Graph-Repairing Rules (GRRs) [10] have also been proposed to target the specific problems of identifying incomplete, conflicting, and redundant information in graphs. They focus on simple graphs, whereas KGs contain multi-edges [31], multiple labels per node (Tab 2), and self-loops. GRRs were preceded by less expressive association rules with graph patterns [14] and functional dependencies for graphs [15]. Rule-mining also has applications beyond KG refinement, such as recommender systems [27]. Our rules could potentially be used in these scenarios, but we leave that for future work.

2.1.2 Embedding-based Refinement. KG embedding approaches seek to learn representations of nodes and relations in a latent space [49], spanning from tensor factorization-based methods [32, 33] to translation-based methods such as TransE [8] and semantic matching models such as ComplEx [48]. These works often perform link prediction, which is useful for completing relationships among entities, but only predicts links between entities already in the KG. In contrast, KGist can identify the absence of entities from the KG.

2.1.3 Hybrid Refinement. Recent refinement methods improve link prediction performance by iteratively applying rule mining and learning embeddings. For instance, pre-trained embeddings have been used to more accurately measure the quality of candidate rules [20]. In [52], facts inferred from rules improve embeddings of sparse entities, and in turn embeddings improve the efficiency of rule mining. Unlike these works, we focus on unifying different refinement tasks, going beyond link prediction.

2.2 Graph Summarization

Graph summarization seeks to succinctly describe a large graph in a smaller representation either in the original or a latent space [24, 26]. Much of the work on knowledge graph summarization has focused on query-related summaries, such as query answer-graph summaries [50], patterns that can be used as query views to improve KG search [13, 45], and sparse, personalized KG summaries—based on historical user queries—for use on personal, resource-constrained devices [41]. While our summaries could conceptually be used for query-related problems, we focus on the problem of characterizing what is normal, strange, and missing in a KG. We also construct summaries with patterns that generalize, which is not considered by [45]. Similar to summarization, Bodeck et al. [7] use MDL to assess KG evolution, but they do not target refinement. Beyond KGs, MDL has been used to summarize static and temporal graphs via structures, such as cliques, stars, and chains [19, 25, 30, 42], or frequent subgraphs [34] (also studied from the perspective of subgraph support [12]). Unlike these works, we learn inductive summaries of recursively defined rules or rooted graphs, which incorporate both the KG structure and semantics, and can be used for graph refinement.


3 INDUCTIVE SUMMARIZATION: MODEL

In this section we describe our proposed MDL formulation for inductive summarization of knowledge graphs, after introducing some preliminary definitions. We list the most frequently used symbols in Table 1, along with their definitions.

3.1 Preliminaries

3.1.1 Knowledge Graph (KG) $G$. A KG is a labeled, directed graph $G = (V, E, \hat{V}, \hat{E}, \phi)$, consisting of a set of nodes or entities $V$, a set of relationship types $\hat{V}$, a set of edges or triples $\hat{E} \subseteq V \times \hat{V} \times V$, a set of node labels $\hat{L}$, and a function $\phi : V \to \hat{P}(\hat{L})$ mapping nodes to their labels, the set of which we give the name type of $G$. We provide an example KG in Fig. 1. An edge or triple $t = (s, p, o)$ connects the subject and object nodes $s, o \in V$ via a relationship type (predicate) $p \in \hat{E}$. An example is $\text{War & Peace}$, $\text{writtenBy}$, $\text{Leo Tolstoy}$. Triples encode a unit of information or fact, semantically about the subject. Since a pair of nodes may have multiple edges between them, we represent the connectivity of $G$ with a $|V| \times |V| \times |L|$ adjacency tensor $A$. Similarly, we store the label information in an $|\hat{L}| \times |V|$ binary label matrix, $L$.

3.1.2 Ideal Knowledge Graph $\hat{G}$. An ideal knowledge graph $\hat{G}(V, \hat{E}, \hat{V}, \hat{E}, \hat{\phi})$ contains all the correct facts in the world and no incorrect ones, i.e., $(s, p, o) \in \hat{E}$ if and only if the fact holds in reality. An ideal KG is only a conceptual aid, and does not exist, since KGS have errors and missing information.

3.1.3 Model $M$ of a KG. A model $M$ of a KG is a set of inductive rules, which describe its facts (see formal definition in §3.1.4). In §3.2, we will explain in the context of our work.

3.1.4 Rule $g$. A rule $g \in M$ is defined recursively and compositionally. Specifically, rule $g = (L_g, \chi_g)$ is a rooted directed graph, with a subset of node labels $L_g \subseteq \hat{L}$ defining $g$'s root, and a set of children $\chi_g$. Each child in $\chi_g$ is of the form $(p, \delta, \phi)$ consisting of a predicate $p$ (e.g., writtenBy), the directionality $\delta$ of the rule (i.e., $\rightarrow$ or $\leftarrow$), and a descendant rule $g$. A leaf rule has no children, i.e., $\chi_g = \emptyset$. An atomic rule consists of one root with a single child (e.g., $(\text{Book}, \text{writtenBy}, \leftarrow, (\text{Author}, \emptyset))$), since all rules can be formed from compositions of these. Rule $g$ in Fig. 2 (which reads, Books have fictional family characters and are written by authors who are born in countries.) rooted at Book, consists of three atomic rules, has root $L_g = \{\text{Book}\}$ and two children $\chi_g$ (for clarity we omit the braces for sets): $(\text{writtenBy}, \leftarrow, (\text{Author}, (\text{bornIn}, \rightarrow, (\text{Country}, \emptyset)))$ and $(\text{character}, \leftarrow, (\text{Fictional Family}, \emptyset))$.

3.1.5 Rule Assertion $d_g$. An assertion $d_g$ of a rule $g = (L_g, \chi_g)$ over the KG $G$ is an instantiation of the edges and labels that $g$ asserts around a particular node, and is reminiscent of a rule grounding [28]. The set of all assertions of rule $g$ is $\mathcal{A}^{(g)}$. Formally, $d_g \in \mathcal{A}^{(g)}$ is a subgraph induced by a traversal that starts at a node $s_{a_g} \in V$ with at least the same labels as $L_g$ (i.e., $L_g \subseteq \phi(s_{a_g})$), and that recursively follows $g$’s syntax. For example, War & Peace is the starting node $s_{a_g}$ of one assertion of the rule in Fig. 2. If the traversal fails to match the syntax of the rule at any point, then we call it an exception of $g$, in which case the assertion is just the node $s_{a_g}$ that violates the rule. Otherwise the induced subgraph is called a correct assertion of $g$. Formally, $\mathcal{A}^{c(g)}_g$ and $\mathcal{A}^{e(g)}_g$ are the set of $g$’s correct assertions and exceptions respectively. Every assertion is either a correct assertion or an exception, so $\mathcal{A}^{c(g)}_g$ and $\mathcal{A}^{e(g)}_g$ form a partition of $\mathcal{A}^{(g)}$.

3.1.6 Minimum Description Length (MDL) Principle. In two-part (crude) MDL [39], given a set of models $M$, the best model $M \in M$ minimizes $L(M) + L(D|M)$, where $L(M)$ is the length (in bits) of the description of $M$, and $L(D|M)$ is the length of the description of the data when encoded using $M$. In our work, we leverage MDL to concisely summarize a given KG.

3.1.7 Problem Definition. Because both errors and missing information are instances of abnormalities, we unify KG characterization in terms of what is normal, strange, and missing, as follows:

**Problem 1 (Inductive KG Summarization).** Given a knowledge graph $G$, and an inaccessible ideal knowledge graph $\hat{G}$, we seek to find a concise model $M$ of inductive rules that summarize what is normal in both $G$ and $\hat{G}$. The rules should be (1) interpretable (by which we mean readable in natural language) and, (2) their exceptions should reveal abnormal information in the KG, whether it be erroneous (e.g., some $t \in \hat{E}$ : $t \notin \hat{E}$), missing (e.g., some $t \in \hat{E}$ : $t \notin \hat{E}$), or a legitimate exception (e.g., some $t \in \hat{E}$ : $t \notin \hat{E}$).

The concise set of rules admits efficient performance on follow-up tasks (such as error detection and incompleteness identification).

### Table 1: Description of major symbols.

| Notation | Description |
|----------|-------------|
| $G(V, E)$ | knowledge graph |
| $A, L$ | binary adjacency tensor and label matrix of $G$, resp. |
| $M, M_0$ | model or set of rules, and the empty model, resp. |
| $L(t)$ | # of bits to transmit an object (e.g., a graph or rule) |
| $\phi$ | rule in the form of a pattern graph |
| $\mathcal{A}(g)$, $\mathcal{A}^{c(g)}_g$, $\mathcal{A}^{e(g)}_g$ | assertions, correct assertions, exceptions of $g$, resp. |
| $\ell$ | set cardinality and number of 1s in tensor/matrix |
Although existing rule mining techniques can be adapted to handle variants of this problem (typically they are tailored to either detect a specific type of error or perform completion), they tend to result in a large number of redundant rules (§ 5.1.2) and require heuristics to be adapted to tasks that they were not designed for. In the next section, we formalize our problem definition further and propose a principled, information-theoretic solution that naturally unifies KG characterization.

### 3.2 Inductive KG Summarization: MDL Model

The inductive KG summarization problem (Problem 1) is closely related to the idea of compression in information theory—compression finds patterns in data (what is normal), which in turn can reveal outliers (what is strange or missing). In this work, we leverage MDL (§ 3.1.6) for KG summarization—a formulation that we are the first to use in the context of KG rule mining. Based on our preliminary definitions above, Problem 1 can be restated more formally:

**Problem 2 (Inductive KG Summarization with MDL).** Given a knowledge graph $G$, we seek to find the model $M^*$ (i.e., set of rules) that minimizes the description length of the graph,

$$M^* = \arg \min_{M \in \mathcal{M}} L(G, M) = \arg \min_{M \in \mathcal{M}} \{L(M) + L(G|M)\},$$

where $M$ is a set of rules (§ 3.1.3) describing what is normal in $G$, $L(M)$ is the number of bits to describe $M$, and $L(G|M)$ is the number of bits to describe parts of $G$ that $M$ fails to describe. Thus, expensive parts of $L(M)$ and $L(G)$ reveal abnormal information in $G$ (§ 4.3).

In § 3.2.1 we will define our model space $\mathcal{M}$, how to describe a KG with a model $M \in \mathcal{M}$, and how to encode it in bits. Then, in § 3.2.2 we will describe the KG under the model, $L(G|M)$, which we refer to as the model’s error, since it encodes what is not captured by $M$. All logarithms are base 2.

#### 3.2.1 MDL Models $M$ and Cost $L(M)$

A model $M \in \mathcal{M}$ is a set of rules, and each rule has a set of correct assertions (or guided traversals of a graph $G$, § 3.1.5). The model thus describes $G$’s semantics (labels) and connectivity (edges) through rule-guided traversals over $G$. Each time a node is visited, some of its labels are revealed by the structure of the rule. For instance, arriving at the node Leo Tolstoy while traversing the subgraph in Fig. 2, reveals (i) its author label, since this is implied by the rule on the left, and (ii) its link to where the traversal just came from (viz., War & Peace).

For our model, we consider a classic information theoretic transmitter/receiver setting [43], where the goal is to transmit (or describe) the graph to the receiver using as few bits as possible. In other words, the sender must guide the receiver in how to fill in an empty binary adjacency tensor $A$ and binary label matrix $L$ with the $1$s needed to describe $G$. Since MDL seeks to find the best model, the costs that are constant across all models (e.g., the number of nodes and edges) can be ignored during model selection. At a high level, beyond this preliminary information, we need to transmit the number of rules (upper bounded by the number of possible candidate rules), followed by the rules in $M$ and their assertions, which we discuss in detail next

\[
L(M) = \log(2 \cdot |L_W|^2 \cdot |L_E| + 1) + \sum_{g \in M} \left( L(g) \cdot \frac{|L(A^g)|}{|L|} \right),
\]

\[L(g) = L(L_g) + L(\{x \in L_g \} + 1) + \sum_{g \in X} \left( -\frac{n_p}{|L|} + 1 \right) + L(\hat{g}^{\delta}),
\]

where $n_p$ is the number of times predicate $p$ occurs in $G$. We discuss each term in Eq. (3) next. We encode the root labels $L_g$ by transmitting their number (upper bounded by $|L_W|$) and then the actual labels via optimal prefix code [11], since they may not occur with the same frequency:

\[
L(L_g) = \sum_{\ell \in L_g} -\log n_{\ell}/|L|, \tag{4}
\]

where $n_{\ell}$ is the number of times label $\ell \in L_W$ occurs in $G$. Then, for the children $X_g$, we transmit their number (expected to be small) using the optimal encoding of an unbounded natural number [40] similarly to [3] and denoted $L_{\delta}$; and per child we specify: (i) its predicate $\hat{p}$ using an optimal prefix code as in Eq. (4), (ii) its directionality $\delta$ (i.e., $\rightarrow$ or $\leftarrow$) without making an a priori assumption about which is more likely, and (iii) its descendent rule $\hat{g}$ by recursively applying Eq. (3) until leaf rules (with 0 children) are reached.

We note that while some labels can be inferred from rules (e.g., the author label of Leo Tolstoy), it is possible that all labels will not be revealed by rules. Thus, we transmit the un-revealed labels as negative error—i.e., information needed to make the transmission lossless, but that is not modeled by $M$. We discuss this in § 3.2.2.

So far, in our running example, once the receiver has the information that War & Peace is a book, it can apply the rule in Fig. 2. It knows that War & Peace should have one or more Fictional Families as characters, and one or more Authors who wrote it, but it does not yet know which Fictional Families and Authors. This information will be encoded next in the assertions.

#### Encoding the Rule Assertions

In Eq. (2), the last term encodes the assertions, $A^g$, of each rule $g$. The receiver infers the starting nodes of the traversals from $g$’s $\delta$-nodes root (Eq. 3)) and the node labels (encoded via other rules or $L^{-}$ in Eq. 10). Thus, we transmit the failed traversals (i.e., exceptions) and details needed to guide the correct assertions:

\[
L(A^g) = L(A^g) + L(A^g), \tag{5}
\]

The first term transmits which assertions are exceptions to a rule (e.g., the book Syntactic Structures, which is non-fiction and hence does not have any Fictional Family characters). We transmit the number of exceptions, followed by their IDs (i.e., which assertions they are), chosen from among the assertions:

\[
L(A^g) = \log(|A^g|) + \log(|A^g|), \tag{6}
\]

of the children $X_g$, we transmit their number (expected to be small) using the optimal encoding of an unbounded natural number [40] similarly to [3] and denoted $L_{\delta}$; and per child we specify: (i) its predicate $\hat{p}$ using an optimal prefix code as in Eq. (4), (ii) its directionality $\delta$ (i.e., $\rightarrow$ or $\leftarrow$) without making an a priori assumption about which is more likely, and (iii) its descendent rule $\hat{g}$ by recursively applying Eq. (3) until leaf rules (with 0 children) are reached.

We note that while some labels can be inferred from rules (e.g., the author label of Leo Tolstoy), it is possible that all labels will not be revealed by rules. Thus, we transmit the un-revealed labels as negative error—i.e., information needed to make the transmission lossless, but that is not modeled by $M$. We discuss this in § 3.2.2.

So far, in our running example, once the receiver has the information that War & Peace is a book, it can apply the rule in Fig. 2. It knows that War & Peace should have one or more Fictional Families as characters, and one or more Authors who wrote it, but it does not yet know which Fictional Families and Authors. This information will be encoded next in the assertions.
where \( \log |\mathcal{A}^{(g)}| \) is an upper bound on the number of exceptions.

Intuitively, the bits needed to encode exceptions penalize overly complex rules, which are unlikely to be accurate and generalizable.

The remaining traversals are correct assertions, for which we transmit details as we traverse each \( a_g \in \mathcal{A}_{\hat{g}}^{(g)} \). The encoding cost for \( \mathcal{A}_{\hat{g}}^{(g)} \) is the sum of the cost of all these traversals:

\[
L(\mathcal{A}_{\hat{g}}^{(g)}) = \sum_{a_g \in \mathcal{A}_{\hat{g}}^{(g)}} L(a_g). 
\]

Each traversal is encoded by recursively visiting neighbors according to the recursive structure of \( g \). Formally,

\[
L(a_g) = \sum_{\hat{g} \in \mathcal{X}_g} \left( \sum_{\# \text{ neighbors}} \log |\mathcal{V}| + \sum_{\text{neighbor ids}} \log (|\mathcal{V}| - 1) \right) + \sum_{a_{\hat{g}} \in \mathcal{A}_{\hat{g}}^{(g)}} L(a_{\hat{g}}),
\]

where, for each child of \( g \), we first transmit the number of \( a_g \)'s neighbors with the child's labels (upper-bounded by the number of nodes \(|\mathcal{V}|\) in \( G \)), followed by the neighbors' IDs (which are the starting nodes of the child rule's correct assertions, since the child is itself a rule) using a binomial transmission scheme. Once the neighbors have been revealed, the traversal proceeds recursively to them. For example, the traversal in Fig. 2 begins at \( \text{War & Peace} \) and the rule has two children (characters and authors). For each, we transmit the number of nodes relevant (5 and 1 respectively), followed by their IDs. The traversal then proceeds recursively to each node just specified.

3.2.2 MDL Error \( L(G|M) \). In Eq. (1), along with sending the model \( M \), we also need to send anything not modeled, i.e., the model's negative error. This error consists of the cost of encoding (i) the node labels that are not revealed by the rules and (ii) the unmodeled edges. We denote the modeled labels and edges as \( L_{M} \) and \( A_{M} \) respectively, which contain the subset of 1s in \( A \) and \( L \) that the receiver has been able to fill in via the rules it received in \( M \). We denote the unmodeled labels and edges as the binary matrix \( L^c = L - L_{M} \) and binary tensor \( A^c = A - A_{M} \), and these are what we refer to as negative error. The cost of the model's error is thus

\[
L(G|M) = L(L^c) + L(A^c). 
\]

Specifically, the receiver can infer the number of missing node labels (i.e., 1s in \( L^c \)) given the total number of node labels and the number not already explained by the model (§ 3.2.1). Thus, we send only the position of the 1s in \( L^c \), encoding over a binomial (where \(|| \) denotes set cardinality and the number of 1s in a tensor/matrix):

\[
L(L^c) = \log ({|\mathcal{L}|}) \cdot (|\mathcal{V}| - |L_{M}|). 
\]

We transmit missing edges \( L(A^c) \) analogously:

\[
L(A^c) = \log (|\mathcal{V}|^2 \cdot |\mathcal{L}|_{c} - |A_{M}|). 
\]

4 INDUCTIVE SUMMARIZATION: METHOD

In the previous section, we fully defined the encoding cost \( L(G,M) \) of a knowledge graph \( G \) with a model \( M \) of rules. Here we introduce our method, KGst, which will leverage our KG encoding \( L(G,M) \) to find a concise summary \( M^* \) of inductive rules, with which it will characterize what is normal, what is strange, and what is missing in the KG.

A necessary step to this end is to generate a set of candidate rules \( C \supseteq M^* \) from which MDL will construct the rules that best compress \( G \). However, even given that set, selecting the optimal model \( M^* \in M \) involves a combinatorial search space, since any subset of \( C \) is a valid model, i.e., \( |M| = 2^{|C|} \) (where even \( |C| \) can be in the millions for large KGs). This cannot be searched exhaustively, and our MDL search space does not have easily exploitable structure, such as the anti-monotonicity property of the support/confidence framework. To find a tractable solution, we exploit the compositionality of rules—starting with simple, atomic rules and building from there. We give KGst’s pseudocode in Alg. 1, and describe it by line next.

4.1 Generating and Ranking Candidate Rules

4.1.1 Candidate Generation (line 1). We begin by generating atomic candidate rules—those that assert exactly one thing (§ 3.1.4). The number of possible atomic rules is exponential in the number of node labels, but not all of them need to be generated: rules that never apply in \( G \) do not explain any of the KG, and hence will not be selected by MDL. Thus, we use the graph to guide candidate generation. For each edge in the graph, KGst generates atomic rules that could explain it. For instance, the edge (\( \text{war} & \text{peace}, \text{writtenBy} \), Leo Tolstoy) could be explained by rules such as, “\( \text{books are written by authors} \)” and “\( \text{authors write books} \).” These have the forms \( g_1 = (\{\text{Book}\}, \{\text{writtenBy} \}, \{\text{Author}\}, 0)) \) and \( g_2 = (\{\text{Author}\}, \{\text{writtenBy} \}, \{\text{Book}\}, 0)) \). To avoid candidate explosion from allowing rules to have any subset of node labels, we only generate atomic rules with a single label per node here, and account for more combinations of labels in the next step.

4.1.2 Qualifying Candidate Rules with Labels (line 2). Adding more labels to rules can help make them more accurate and more inductive by limiting the number of places they apply (e.g., Fig. 3), and subsequently their exceptions (which incur a cost in our MDL model). To this end, given a rule \( g \), KGst identifies the labels shared by all the starting nodes of the correct assertions of the rule: \( \phi_g = \cap_{a_g \in A_g^{(g)}} \phi(a_g) \). If this set contains more labels than the rule (i.e., \( L_g \subset \phi_g \)), then it forms a new rule \( g' \) with root \( \phi_g \). If \( L(G, M_0 \cup (g')) \leq L(G, M_0 \cup (g)) \), where \( M_0 \) is the empty model without any rules, KGst replaces \( g \) with \( g' \) in \( C \). It carries this out for all the rules in \( C \). This can be viewed as qualifying \( g \): it qualifies the conditions under which \( g \) applies, to those that contain all the labels rather than its original label alone.

4.1.3 Ranking Candidate Rules (line 3). Considering all possible combinations of candidate rules \( \mathcal{P}(C) \), and finding the optimal model \( M^* \) is not tractable. Moreover, an alternative greedy approach that constructs the model by selecting in each iteration the rule \( g \in C \) that leads to the greatest reduction in the encoding cost, would still be quadratic in \( |C| \) (which is in the order of many millions for large-scale KGs). Instead, for scalability, given the set of (potentially qualified) candidate rules \( C \), we devise a ranking that allows KGst to take a constant number of passes over the candidate rules. Intuitively, our ranking considers the amount of
explanatory power that a rule has—i.e., how much reduction in error it could lead to:
\[
\Delta L(G|M_0 \cup \{ g \}) = L(G|M_0) - L(G|M_0 \cup \{ g \}).
\] (12)

KGist sorts the rules descending on this value, and breaks ties by considering rules with more correct assertions first. If that fails, the final tie-breaker is the lexicographic ordering of rules’ root labels.

4.2 Selecting and Refining Rules

4.2.1 Selecting Rules (lines 4-9). After ranking the candidate rules \( C \), KGist initializes \( M = \emptyset \) and considers each \( g \in C \) in ranked order for inclusion in \( M \). For each rule \( g \), it computes \( L(G,M \cup \{g\}) \), i.e., the MDL objective if \( g \) is added to the current model. If this is less than the MDL cost \( L(G,M) \) without the rule (e.g., rule \( g \) correctly explains new parts of \( G \)), then KGist adds \( g \) to \( M \). If \( g \) has a reverse version (e.g., “books are written by authors” and “authors write books”), KGist considers both at once and picks the one that gives a lower MDL cost. KGist runs a small number of passes over \( C \) until no new rules are added. The resulting model \( M \) approximates the true optimal model \( M^* \).

4.2.2 Refining Rules (line 10). The model at this point only contains atomic rules. To better approximate \( M^* \), we introduce two refinements that compose rules via merging \( \mathcal{R}_m \) and nesting \( \mathcal{R}_n \).

Refinement \( \mathcal{R}_m \) for “rule merging” composes rules that share a root. It identifies all sets of rules, \( \{g_1, \ldots, g_j\} \) with matching roots that correctly apply in the same cases, i.e., \( \mathcal{L}_{g_1} = \cdots = \mathcal{L}_{g_j} \) and \( \{s_{a_{g_1}} : a_{g_1} \in \mathcal{A}^{(g_1)}\} = \cdots = \{s_{a_{g_j}} : a_{g_j} \in \mathcal{A}^{(g_j)}\} \). It then merges these into a single rule \( g' \), consisting of the union of the children \( \mathcal{X}_{g_1} \cup \cdots \cup \mathcal{X}_{g_j} \). For example, if all books that have authors \( g_j \) also have publishers \( g_j \), these would be merged into a single rule.

We refer to this variant as \( \text{KGist} + \mathcal{M} \).

Refinement \( \mathcal{R}_n \) for “rule nesting” considers composing rules where an inner node of one rule \( g_{in} \) matches the root of another rule \( g_{rt} \), possibly creating a more inductive rule. \( \mathcal{R}_n \) begins by computing, between each compatible \( g_{in} \) and \( g_{rt} \), the Jaccard similarity of the correct assertion starting points of the matching inner and root nodes (i.e., it quantifies the ‘fit’ of the nodes). For instance, if a rule asserts that “books have authors” \( g_{in} \), and another rule asserts that “authors have a birthplace” \( g_{rt} \), then the Jaccard similarity is computed over the set of book authors and the set of authors with birthplaces. The refinement then considers nesting the rules in descending order of Jaccard similarity, resulting in rule \( g_{rt} \) being subsumed into rule \( g_{in} \), which becomes its ancestor. If the composed rule \( g_{in} \circ g_{rt} \) leads to lower encoding cost than the individual rules (e.g., \( g_{in} \) qualifies \( g_{rt} \) as in Fig. 3), i.e., \( L(G, (M \cup \{g_{in}, g_{rt}\}) \cup \{g_{in} \circ g_{rt}\}) < L(G, M) \), then the composition replaces \( g_{in} \) and \( g_{rt} \). The Jaccard similarity between rules that were compatible with \( g_{in} \) or \( g_{rt} \) is then recomputed with \( g_{in} \circ g_{rt} \), the list of compatible rules is re-sorted by Jaccard similarity, and the search continues until all pairs are considered with none being composed (i.e., when no composition reduces the encoding cost).

This sorting is done over the set of selected rules \( M \) (§ 4.2.1), where \( |M| < |C| \), and since only few compositions occur (§ 5.1.1), this repeated sorting is tractable. As nesting embeds \( g_{rt} \) into \( g_{in} \), this refinement allows for arbitrarily expressive rules to form. We call our method with both refinements (merging and nesting) \( \text{KGist} + \mathcal{n} \).

4.3 Deriving Anomaly Scores

We now discuss how to leverage a model \( M \) (i.e., a summary of rules) mined by KGist towards identifying what is strange or anomalous in a KG, whether it be erroneous or missing—two key tasks in KG research. Anomaly detection seeks to identify objects that differ from the norm [1, 4]. In our case, the learned summary concisely describes what is normal in a KG.

Intuitively, nodes that violate rules, and edges that are unexplained are likely to be anomalous. Next, we make this intuition more principled by defining anomaly scores for entities (nodes) and relationships (edges) in information theoretic terms.

4.3.1 Entity Anomalies. We define the anomaly score \( \eta \) of an entity or node \( v \) as the number of bits needed to describe it as an exception to the rules in the model:
\[
\eta(v) = \sum_{g \in \mathcal{R}(v), \text{id} \in \mathcal{A}^{(g)}} \left\lfloor \log \frac{|\mathcal{A}^{(g)}|}{|\mathcal{A}^{(g)}\}_{\text{id}}} \right\rfloor, \quad \text{bits to model } v \text{ as an exception}
\] (13)

where \( r : \mathcal{V} \rightarrow \mathcal{P}(M) \) maps each node to the rules that apply to it. We distribute the cost of the exceptions equally over all \( \mathcal{A}^{(g)} \) violating nodes, following Eq. (6).

4.3.2 Relationship Anomalies. We also introduce an anomaly score for a relationship or edge. Intuitively, edges that are not explained by the model \( M \) are anomalous, and their anomaly score is defined as the number of bits describing them as negative error. Since we transmit all unmodeled edges in \( A^{-} \) together (Eq. (11)), we make this intuition more principled by distributing this transmission cost evenly across all unmodeled edges in the anomaly score:
\[
\eta^{(p)}(s, o, p) = \begin{cases} 
\frac{1}{|A^-|} \left\lfloor \log \left( \frac{|\mathcal{V}|^2 + |\mathcal{L}_E| - |A^-|}{|A^-|} \right) \right\rfloor, & \text{if } A_{s, o, p} = 1 \\
0 & \text{otherwise}
\end{cases},
\] (14)

Under the reasonable assumption that our model effectively captures what is normal in the KG, it follows that edges unexplained by the model are likely to be abnormal. Equation (14) captures this
notion, but to prevent the unexplained edges from all receiving equal scores, we add the anomaly score of the endpoints (Eq. (13)):

\[ \eta(s, p, o) = \eta(s) + \eta(o) + \eta'(s, p, o). \]  

(15)

4.4 Complexity Analysis

Generating candidate rules involves iterating over each edge (and its nodes’ labels) as it is encountered. The number of possible atomic rules with a single label that could explain an edge \((s, p, o)\) is \(2 \cdot |\phi(s)| \cdot |\phi(o)|\). Letting \(\phi_{\text{max}}\) denote the max number of labels over all nodes, the overall complexity of candidate generation is \(O(|E| \cdot \phi_{\text{max}}^2)\). The number of candidate rules generated, \(|C|\), is also \(O(|E| \cdot \phi_{\text{max}}^2)\).

Computing the error, \(L(G(M))\), is constant since it only involves computing the log-binomials. The computation of \(L(M)\) depends on the time of traversing and describing the correct assertions. Since the traversals occur in a DFS manner (in linear time) over a subgraph enough smaller than \(G\) to be ignored, \(L(M)\) takes \(O(|M|)\) time. Since ranking only requires computing \(L(G(M))\), it is a small constant, the cost comes only from sorting \(|C|\) items, which is \(O(|C| \cdot \log |C|)\).

\(\text{KGist}\) takes a small number of passes over the candidate set (§4.2.1) in \(O(|C|)\) time. So, the overall complexity is \(O(|C| + |C| \cdot \log |C|)\), which simplifies to \(O(|C| \cdot \log |C|)\), or \(O(|E| \cdot \phi_{\text{max}}^2 \cdot \log(|E| \cdot \phi_{\text{max}}^2))\). We omit the complexity of the refinements for brevity.

5 EVALUATION

Our experiments seek to answer the following questions:

Q1. Does KGist characterize what is normal? How well can KGist compress, in an interpretable way, a variety of KGs?

Q2. Does KGist identify what is strange? Can it identify and characterize multiple types of errors?

Q3. Does KGist identify what is missing?

Q4. Is KGist scalable?

Data. Table 2 gives descriptive statistics for our data: NELL [9] or “Never-Ending Language Learning” continually learns facts via crawling the web. Our version contains 1,115 iterations, each introducing new facts for which the confidence has grown sufficiently large. DBpedia [5] is extracted from Wikipedia data, heavily using the structure of infoboxes. The extracted content is aligned with the DBpedia ontology via crowd-sourcing [35]. Yago [46], like DBpedia, is built largely from Wikipedia. Yago contains 3 orders of magnitude more node labels than the other two graphs (Tab. 2).

5.1 [Q1] What is normal in a KG?

In this section, we demonstrate how KGist characterizes what is normal in a KG by achieving (1) high compression, (2) concise, and (3) interpretable summaries with intuitive rules.

5.1.1 KG Compressibility. Although compression is not our goal, it is our means to evaluate the quality of the discovered rules. Effective compression means that the discovered rules describe the KG accurately and concisely.

| | \(|V|\) | \(|E|\) | \(|L_v|\) \(|L_e|\) | \(\text{avg } \phi(v)\) | \(\text{med } \phi(v)\) |
|---|---|---|---|---|---|
| NELL | 46,682 | 231,634 | 266 | 821 | 1.53 | 1 |
| DBpedia | 976,404 | 2,862,489 | 239 | 504 | 2.72 | 3 |
| Yago | 6,349,336 | 12,027,848 | 629,681 | 33 | 3.81 | 3 |

Setup. We run KGist on all three KGs since each has different properties (Tab. 2). \(M_0\) denotes an empty model with no rules, corresponding to transmitting the graph entirely as error, i.e., \(L(G, M_0) = L(G(M_0))\). We compare compression over this model.

Baselines. We compare to: (i) Freq which, instead of using MDL to select rules from \(C\), selects the top-k rules that correctly apply the most often, where we set \(k\) to be the number selected by the best compressed version of KGist. (ii) Coverage is directly analogous to Freq, replacing the metric of frequency with the number of edges explained by the rule. Both select rules independently, without regard for whether rules explain the same edges. (iii) AMIE+ [17] finds Horn rules, which cannot be encoded with our model, so we do not report compression results, but only the number of rules it finds. While other KG compression techniques exist (§2.2), we are seeking to find inductive rules that are useful for refinement, whereas generic graph compression methods compress the graph, but never generate rules, and are hence not comparable.

Metrics. For each dataset, the first row reports the percentage of bits needed for \(L(G, M)\) relative to the empty model. That is, it reports \(L(G, M)/L(G, M_0)\). Small values occur when \(M\) compresses \(G\) well, and hence smaller values are better. The second row reports the percentage of edges explained: \(|A_M|/|A|\). Lastly, we report how many rules were selected to achieve the results.

Results. We record KG compression in bits in Table 3. In all cases, KGist is significantly more effective than the Freq and Coverage baselines, which ignore MDL. Indeed, Freq and Coverage result in values greater than 100% in the first row, meaning they lead to an increase in encoding cost over \(M_0\), due in part to selecting rules independently from each other, and hence potentially explaining the same parts of the graph with multiple rules. KGist is very effective at explaining the graph, leaving only a small percentage of the edges unexplained. It also explains more edges than Coverage due to rule overlap again. The two refinements, \(Rn\) and \(Rn\), are also effective at refining model \(M\) to more concisely describe \(G\). \(Rn\), which allows arbitrarily expressive rules, refines \(M\) to contain fewer and better compressing rules. KGist\(+\) explains slightly fewer edges than KGist\(+\) because nested rules apply only when their root does (e.g., Fig 3).

5.2.1 Rule Conciseness & Interpretability. We compare the number of rules mined by KGist to that of AMIE+ [17]. For AMIE+, we set min-support to 100 and min PCA confidence to 0.1, as suggested by the authors [17]. When running AMIE+ on graphs larger than NELL, we experienced intolerable runtimes (inconsistent with those in [17]). For Yago we were unable to get results, while for DBpedia we report numbers from [18] on an older, but similarly sized version of DBpedia. In Tab. 3 we see that KGist mines orders of magnitude fewer rules than AMIE+, showing that it is more computationally tractable to apply our concise summary of rules to refinement tasks than the sheer number of rules obtained by other rule-mining methods that operate in a support/confidence framework. This is because redundant rules cost additional bits to describe, so MDL encourages conciseness. While these other methods could use the min-support parameter to reduce the number of rules, it is not clear how to set this parameter \textit{a priori}. Using MDL, we can approximate the optimal number of rules in a parameter-free, information-theoretic way, leading to fewer but descriptive rules.
Table 3: Compression: The small % bits needed (relative to an empty model) and number of rules found by various models demonstrate the effectiveness of KGist variants at finding a concise set of rules in G. AMIE+ [17] finds Horn rules, which cannot be encoded with our model, so we only report the number of rules it finds. Freq and Coverage are baseline models that we introduce by greedily selecting from our candidate set C (without MDL) the top-k rules that (1) correctly apply the most often and (2) cover the most edges, resp. For these, we preset k to the number of rules found by the best-compressed version of our method and report it as top-k to distinguish from the non-preset values.

| Dataset | Metric                  | Horn rules | Rules of the form $g = (L_y, X_g)$ |
|---------|-------------------------|------------|-----------------------------------|
|         |                         | AMIE+      | Freq   | Coverage | KGist | KGist+m | KGist+n |
| NELL    | % Bits needed           | N/A        | 191.46%  | 192.72%  | 73.88%  | 73.00%  | 63.57%  |
| (6,268,200) | Edges Explained        | N/A        | 57.33%   | 50.12%   | 78.52%  | 78.52%  | 74.67%  |
|         | # Rules                | 32,676     | top-k    | top-k    | 1,115   | 647     | 573     |
| DBpedia | % Bits needed           | N/A        | 674.51%  | 718.22%  | 69.88%  | 69.84%  | 68.77%  |
| (119,117,468) | Edges Explained       | N/A        | 80.64%   | 71.70%   | 89.17%  | 89.17%  | 88.51%  |
|         | # Rules                | -6,963 [17]| top-k    | top-k    | 516     | 505     | 498     |
| Yago    | % Bits needed           | N/A        | 896.33%  | 947.64%  | 76.13%  | 75.98%  | 75.04%  |
| (793,027,801) | Edges Explained      | N/A        | 86.54%   | 83.44%   | 88.40%  | 88.40%  | 85.20%  |
|         | # Rules                | failed     | top-k    | top-k    | 60,298  | 34,331  | 32,670  |

Furthermore, we present and discuss in Fig. 3 example rules mined with KGist. These show that our rules are interpretable and intuitively inductive, and that Rn is a useful refinement for improving the inductiveness of rules.

5.2 [Q2] What is strange in a KG?

Here we quantitatively analyze the effectiveness of KGist at identifying a diverse set of anomalies, and demonstrate the interpretability of what it finds. Whereas most approaches focus on exceptional facts [51], erroneous links, erroneous node type information [35], or identification of incomplete information (e.g., link prediction) [17], KGist rules can be used to address multiple of these at once. To evaluate this, we inject anomalies of multiple types into a KG, and see how well KGist identifies them.

Setup. We inject four types of anomalies. For each, we select q percent of G’s nodes uniformly at random to perturb. We sample nodes independently for each type, so it is possible that occasionally a node is chosen multiple times. This is realistic, since there are multiple types of errors in KGs at once [35]. Although we target nodes, our perturbations also affect their incident edges. Thus, we formulate the anomaly detection problem as identifying the perturbed edges. Specifically, we introduce the following anomalies:

- **A1 Missing labels**: We remove one label from each node. Unlike the A2-A4, we only sample nodes with more than one label. E.g., we may remove the entrepreneur label from Bill Gates, leaving the labels billionaire, etc. We consider all the in/out edges of the altered nodes as perturbed.
- **A2 Superfluous labels**: We add to each node a randomly selected label that it does not currently have. E.g., we may add the label Fruit to Taj Mahal.
- **A3 Erroneous links**: We inject 1 or 2 edges incident to each node, choosing the edge’s predicate and destination randomly. E.g., we may inject random edges like (Des Moines, owner, Coca-Cola). We mark injected edges as anomalous.
- **A4 Swapped labels**: For each node, we replace a label with a new random one that it does not yet have.

For this experiment we show results on NELL, since it has confidence values for each of its edges, which we can use to sample negative examples. The perturbed edges are ground truth errors (positive examples), and we randomly sample from NELL an equal number of ground truth correct edges with a confidence value of 1.0 (after filtering out edges that our injected anomalies perturbed). We use a 20/80 validation/test split, and the perturbed graph for training.

Baselines. We compare to (i) ComplEx, an embedding method that we tune as in [48] (ranking edges based on its scoring function), (ii) TransE, an embedding method that we tune as in [8] (ranking edges based on their energy scores), (iii) SDValidate [36], an error detection method deployed in DBpedia (it outputs an edge ranking), and (iv) AMIE+, designed for link prediction, but which we adapt for error detection by ranking based on the sum of the confidences of the rules that predict each test edge (i.e., edges that are predicted by many, high-confidence rules will be low in the ranking, and edges that are not predicted by any rules will be high in the ranking). We also tried PaTyBRED [29], but it had prohibitive runtime.

**KGist variants.** To define edge anomalies for our variants, we use the edge-based anomaly score $\eta$ in Eq. (15). KGist_Freq is the Freq method described in § 5.1.1, but uses KGist’s anomaly scores. While KGist+n learns compositional rules that help with compression, we found that the simpler rules of KGist+m performed better in this task, so we report only its results for brevity. The unsupervised methods do not have hyper-parameters, but are tested on the same test set as ComplEx/TransE, so the validation set errors are additional noise they must overcome.

**Metrics.** Each ranking includes only the test set edges, and we compute the AUC for each ranking, using reciprocal rank as the predicted score for each edge—edges higher in the ranking being closer to 1 (i.e., more anomalous). We also compute Precision@100, Recall@100, and F1@100 for (i) the entire test set of edges; (ii) each type of perturbed edges from the different anomaly types. For ties in the ranking, we extend the list beyond 100 until the tie is broken (e.g., if the 100th and 101st edge have the same score, then we compute over 101 edges). Ties did not often extend much beyond
Table 4: Anomaly detection results on NELL. We mark the best performing method with a gray background and the best unsupervised method with bold text. We mark statistical significance at a 0.05 p-value (paired t-test) with an "*" for KGist_Freq/KGist+m vs. unsupervised methods. The final row shows the average rank of each method. KGist+m performs the most consistently well.

| Task | Metric | Supervised | Unsupervised |
|------|--------|------------|--------------|
|      |        | ComplEx   | TransE       | SDValidate  | AMIE+ | KGist_Freq | KGist+m |
| All anomalies | AUC | 0.5068 ± 0.02 | 0.5779 ± 0.04 | 0.4996 ± 0.00 | 0.4871 ± 0.04 | 0.5739 ± 0.01 | 0.6052 ± 0.03* |
| | P@100 | 0.4820 ± 0.05 | 0.7940 ± 0.06 | 0.5100 ± 0.04 | 0.3980 ± 0.07 | 0.6116 ± 0.10 | 0.7419 ± 0.07* |
| | R@100 | 0.0087 ± 0.00 | 0.0126 ± 0.00 | 0.0092 ± 0.00 | 0.0072 ± 0.00 | 0.0126 ± 0.00 | 0.0139 ± 0.00* |
| | F1@100 | 0.0172 ± 0.00 | 0.0247 ± 0.00 | 0.0181 ± 0.00 | 0.0141 ± 0.00 | 0.0247 ± 0.01 | 0.0273 ± 0.01* |
| A1 missing labels | AUC | 0.5842 ± 0.04 | 0.6021 ± 0.06 | 0.4997 ± 0.00 | 0.4409 ± 0.06 | 0.5149 ± 0.02 | 0.6076 ± 0.03* |
| | P@100 | 0.2640 ± 0.05 | 0.4280 ± 0.15 | 0.3040 ± 0.06 | 0.1200 ± 0.05 | 0.4067 ± 0.11 | 0.4759 ± 0.05* |
| | R@100 | 0.0119 ± 0.00 | 0.0181 ± 0.01 | 0.0134 ± 0.00 | 0.0057 ± 0.00 | 0.0199 ± 0.01 | 0.0244 ± 0.01* |
| | F1@100 | 0.0227 ± 0.01 | 0.0346 ± 0.01 | 0.0257 ± 0.01 | 0.0109 ± 0.01 | 0.0377 ± 0.01 | 0.0463 ± 0.02* |
| A2 superfluous labels | AUC | 0.5502 ± 0.02 | 0.5659 ± 0.03 | 0.4989 ± 0.01 | 0.4946 ± 0.03 | 0.4997 ± 0.04 | 0.5115 ± 0.03 |
| | P@100 | 0.1780 ± 0.05 | 0.3160 ± 0.16 | 0.2160 ± 0.07 | 0.1040 ± 0.09 | 0.2081 ± 0.06 | 0.2485 ± 0.09 |
| | R@100 | 0.0112 ± 0.00 | 0.0219 ± 0.01 | 0.0152 ± 0.00 | 0.0070 ± 0.01 | 0.0169 ± 0.01 | 0.0175 ± 0.01 |
| | F1@100 | 0.0229 ± 0.00 | 0.0408 ± 0.02 | 0.0283 ± 0.01 | 0.0131 ± 0.01 | 0.0311 ± 0.01 | 0.0326 ± 0.01 |
| A3 erroneous links | AUC | 0.2493 ± 0.03 | 0.4126 ± 0.08 | 0.4966 ± 0.01 | 0.8902 ± 0.08 | 0.7383 ± 0.00 | 0.8423 ± 0.00 |
| | P@100 | 0.1020 ± 0.04 | 0.0920 ± 0.00 | 0.0480 ± 0.02 | 0.1860 ± 0.08* | 0.0331 ± 0.01 | 0.0317 ± 0.01 |
| | R@100 | 0.0374 ± 0.02 | 0.0067 ± 0.00 | 0.0176 ± 0.01 | 0.0679 ± 0.03* | 0.0051 ± 0.01 | 0.0052 ± 0.01 |
| | F1@100 | 0.0548 ± 0.02 | 0.0011 ± 0.00 | 0.0257 ± 0.01 | 0.0995 ± 0.05* | 0.0074 ± 0.01 | 0.0075 ± 0.01 |
| A4 swapped labels | AUC | 0.5369 ± 0.03 | 0.5527 ± 0.02 | 0.4991 ± 0.00 | 0.4891 ± 0.03 | 0.6904 ± 0.01* | 0.6633 ± 0.07 |
| | P@100 | 0.2160 ± 0.08 | 0.4200 ± 0.09 | 0.2080 ± 0.08 | 0.1240 ± 0.06 | 0.5360 ± 0.15* | 0.4768 ± 0.10 |
| | R@100 | 0.0136 ± 0.00 | 0.0269 ± 0.01 | 0.0128 ± 0.00 | 0.0079 ± 0.00 | 0.0379 ± 0.01* | 0.0320 ± 0.01 |
| | F1@100 | 0.0256 ± 0.01 | 0.0055 ± 0.01 | 0.0241 ± 0.01 | 0.0148 ± 0.01 | 0.0705 ± 0.01* | 0.0599 ± 0.01 |
| Avg rank | 4.10 | 2.90 | 4.15 | 5.00 | 2.90 | 1.95 |

Figure 4: Example anomalies in NELL (left two) and DBpedia (right) that violate many rules. The 1st and 3rd are missing information. While most states are the headquarters of sports teams, Pennsylvania does not have any teams listed. However, the Steelers, Eagles, etc. are all teams located in PA. Also, unlike most music genres, 4-beat has no persons listed who play it. The 2nd exception may not capture missing information, since hippopotamuses were, until recently, considered herbivores; this node is anomalous as it differs from many carnivorous and omnivorous mammals.

100, but for KGist_Freq they tended to extend farthest. Positives are considered perturbed edges and negatives un-perturbed edges. When computing scores for a particular anomaly type, we first filter the ranking to only contain the edges perturbed by that anomaly type and the un-perturbed edges to ensure that edges perturbed by other anomaly types are not considered false negatives.

Results. In Table 4 we report results identifying anomalies generated with sampling probability $q = 0.5\%$ and 5 random seeds. We report avg and stdev over the 5 perturbed graphs. Across all anomaly types, KGist+m is most effective at identifying anomalous edges, demonstrating its generality. This is further evidenced by its top average ranking: it ranks 1.95 on average across all anomaly types and metrics. Furthermore, as discussed in Fig. 4, not only can it identify anomalies, but its interpretable rules allow us to reason about why something is anomalous.

In most cases, KGist+m even outperformed ComplEx and TransE, supervised methods requiring validation data for hyper-parameter tuning. A2 is the only anomaly type where supervised methods outperform unsupervised methods, but the difference is not statistically significant. KGist_Freq performs better than most other baselines, demonstrating that our formulation of anomaly scores and rules are effective at finding anomalies. However, as KGist+m usually outperforms KGist_Freq, we conclude that MDL leads to improvement over simpler rule selection approaches. AMIE+ only performed well on A3. We conjecture that this is because randomly injected edges are likely to be left un-predicted by all of AMIE+’s rules. On the other hand, edges with perturbed endpoints may be left un-predicted by some rules, but, out of the large number of rules that AMIE+ mines (§ 5.1.2), some rule is likely to still predict it. The results for $q = 1.0\%$ were overall consistent, with a few fluctuations between KGist+m and KGist_Freq. We omit the results for brevity.

5.3 [Q3] What is missing in a KG?

In this section, we evaluate KGist’s ability to find missing information. Most KG completion methods target link prediction, which seeks to find missing links between pairs of nodes that are present in a KG. If either node is missing, then link prediction cannot provide any information. We focus on this task: revealing where entities are missing. Since KGist’s rules apply to nodes, rather than edges, the rule exceptions can reveal where links to seen or unseen entities are missing (but cannot predict which specific entity the link should be to). Thus, our task and link prediction are complementary.

Setup. We assume the commonly used partial completeness assumption (PCA) [16, 18, 35], according to which, if an entity has
one relation of a particular type, then it has all relations of that type in the KG (e.g., a movie with at least one actor listed in the KG has all its actors listed). We generate a perturbed KG with ground-truth incomplete information via the following steps: (1) we randomly remove q% of nodes (and their adjacent edges) from G, and (2) we enforce the PCA (e.g., if we removed one actor from a movie, then we remove all the movie’s actor edges). Our goal is to identify that the neighbors of the removed nodes are missing information, and what that information is. We run KGist on the perturbed KG, and identify the exceptions, \( A^{(n)} \), of each rule \( g \in M \). If a rule asserts the removed information, then this is a true positive. For example, if we removed Frankenstein’s author and KGist mines the rule that books are written by authors, then that rule asserts the removed information. We use NELL and DBpedia for this experiment as their sizes permit several runs over different perturbed KG variants.

**Baselines.** Link prediction methods are typically used for KG completion, but they do not apply to our setting: they require that both endpoints of an edge be in G to predict the edge, while our setup assumes that one endpoint is missing from G. Thus, we compare KGist to Freq and AMIE+C. Freq, as before (§ 5.1.1), selects the top-k rules with the most correct assertions, where k is set to the number of rules KGist mines. AMIE+C is what we name the method from [16]. AMIE+C requires training data comprised of examples of \((u, \text{incomplete}, p)\) triples where \( u \in V \) is an entity, \( p \in L \) is a predicate, and the triple specifies that node \( u \) is missing its links of type \( p \) (e.g., a movie is missing actors). We use 80% of the removed data as training data for AMIE+C and test all methods on the remaining 20%. We tune AMIE+C’s parameters as in [16].

**Metrics.** We report only recall, R, since information that we did not remove but was reported missing could be either a false positive, or real missing information that we did not create [35]. We compute recall as the number of nodes identified as missing, divided by the total number of nodes removed. In addition, we compute a more strict recall variant, \( R_L \), which requires that the missing node’s label also be correctly identified (e.g., not only do we need to predict the absence of a missing writtenBy edge, but also that the edge should be connected to an author node). KGist can return this label information, but AMIE+C only predicts the missing link, not the label. Thus, we omit \( R_L \) for AMIE+C.

**Results.** We report results in Table 5 with \( q = 5\% \) (the results are consistent with other q values). KGist outperforms all baselines by a statistically significant amount (10-11% on R, 13-27% on \( R_L \), and paired t-test p-values \(< 0.01\)), which demonstrates its effectiveness in finding information missing from a KG. We conjecture that KGist outperforms AMIE+C because AMIE+C requires all training data be focused around a small number of predicates (e.g., 10-11 in [16]) in order to learn effective rules, while MDL encourages KGist to explain as much of the KG as possible, allowing it to find missing information over more diverse regions of the KG. This is further evidenced by the fact that KGist outperforms Freq, which only applies to frequent regions of the KG. Not only does KGist effectively reveal where the missing nodes are, it also usually correctly identifies their labels (small drop in \( R_L \) compared to R). AMIE+C is not able to do this, and Freq can only report the label sometimes, by taking advantage of the rules we formulated in this work.

### Table 5: KG completion results on NELL and DBpedia (averages and stds of 10 runs).

| Dataset | Metric | Supervised | Unsupervised |
|---------|--------|------------|--------------|
| NELL    | R      | 0.6587 ± 0.03 | N/A          |
|         | R_L    | 0.4589 ± 0.02 | 0.7598 ± 0.02 |
| DBpedia | R      | 0.8187 ± 0.01 | 0.6589 ± 0.02 |
|         | R_L    | 0.8049 ± 0.01 | 0.6636 ± 0.01 |

**5.4 Scalability**

In this section, we evaluate KGist’s performance as the number of edges in the KG grows. We perform this evaluation on an Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz with 1TB RAM, using a Python implementation. NELL, DBpedia, and Yago have from 231,634 to 12,027,848 edges. We run KGist on each KG three times. Since we aim to analyze the runtime with respect to the number of edges, but Yago has three orders of magnitude more labels, we run KGist with an optimization that only generates candidate rules with the 300 most frequent labels (approximately equal to NELL and DBpedia), allowing us to fairly investigate the effect of the number of edges. Figure 5 shows the number of edges vs. runtime in seconds. In practice, KGist is near-linear in the number of edges. Even on Yago, it mines summaries in only a few minutes, and on NELL in seconds.

### 6 CONCLUSION

This paper proposes a unified, information theoretic approach to KG characterization, KGist, which solves our proposed problem of inductive summarization with MDL. KGist describes what is normal in a KG with a set of interpretable, inductive rules, which we define in a new, graph-theoretic way. The rule exceptions, and the parts of the KG that the summary fails to describe reveal what is strange and missing in the KG. KGist detects various anomaly types and incomplete information in a principled, unified manner, while scaling nearly linearly with the number of edges in a KG—this property allows it to be applied to large, real-world KGs. Future work could explore using KGist’s rules to guide KG construction.

### ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. IIS 1845491, Army Young Investigator Award No. W911NF1810397, an Adobe Digital Experience research faculty award, and an Amazon faculty award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties.
[51] Gensheng Zhang, Damian Jimenez, and Chengkai Li. Maverick: Discovering exceptional facts from knowledge graphs. In Proceedings of the 27th ACM Conference on Information and Knowledge Management (CIKM), Torino, Italy, 2018.

[52] Wen Zhang, Bibek Paudel, Liang Wang, Jiaoyan Chen, Hai Zhu, Wei Zhang, Abraham Bernstein, and Huajun Chen. Iteratively learning embeddings and rules for knowledge graph reasoning. In Proceedings of the 28th International Conference on World Wide Web (WebConf), San Francisco, CA, pages 2366–2377, 2019.

[53] Mussab Zneika, Dan Vodislav, and Dimitris Kotzinos. Quality metrics for rdf graph summarization. Semantic Web, pages 1–30, 2019.