

A Systematic Survey on Synthetic Knowledge Graph Generators

Ana Alexandra Morim da Silva¹ ✉ 

Data Science Group (DICE), Heinz Nixdorf Institute, Paderborn University, Germany

Michael Röder ✉ 

Data Science Group (DICE), Heinz Nixdorf Institute, Paderborn University, Germany

Axel-Cyrille Ngonga Ngomo ✉ 

Data Science Group (DICE), Heinz Nixdorf Institute, Paderborn University, Germany

Abstract

Knowledge graphs are the backbone of an increasing number of data-driven software solutions, where performance, i.e., runtime and accuracy, depends critically on the data they process. Consequently, synthetic knowledge graph generators become essential tools for reliable evaluation and future performance prediction of knowledge graph-driven solutions. These generators address critical challenges, including the exponential growth of knowledge graphs, the difficulty in acquiring real-world domain knowledge graphs, and the need for realistic data to predict future behaviour under realistic workloads. Generators overcome these challenges by producing scalable and controllable knowledge graph structures. Knowledge graphs at various scales enable the testing of existing solutions on larger-

than-current graphs, supporting future-proof solutions. Controllable structures, in turn, support the generation of knowledge graphs in data-scarce domains and with realistic properties. This survey comprehensively analyzes the current landscape of synthetic knowledge graph generators, including evaluation tasks and metrics. We catalog 38 generators published between 2014 and 2025 based on their methodology, generation requirements, and the type of generated graph. We also survey the evaluation tasks and technical details reported for these generators, providing a reference for both practitioners looking for a generator and researchers designing new ones. We conclude with open challenges and promising research directions.

2012 ACM Subject Classification General and reference → Surveys and overviews; Theory of computation → Graph algorithms analysis; Information systems → Resource Description Framework (RDF)

Keywords and phrases Survey, Synthetic Knowledge Graph Generation, Knowledge Graph, Synthetic Data, Graph Generation

Digital Object Identifier 10.4230/TGDK.4.3.1

Category Survey

Funding This work has been supported by the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW) within the project SAIL under the grant no NW21-059D and by the German Federal Ministry of Research, Technology and Space (BMFTR) within the project KI-Akademie OWL under the grant no 16IS24057B.

Received 2026-01-09 **Accepted** 2026-08-13 **Published** 2026-09-30

Generative AI Declaration AI-based tools were used during the preparation of this manuscript to assist with selected passages, including language refinement, grammar checking, and spelling correction. All of the generated suggestions were critically reviewed and edited by the authors before being added to the manuscript.

¹ Corresponding author.



1 Introduction

Knowledge Graph (KG)-driven solutions depend critically on the availability of large high-quality KGs. However, using real-world KGs to benchmark such systems poses several challenges:

- (i) the continuous growth of data makes real data measurements obsolete within a short time frame [97], hence limiting resource planning (e.g., hardware, computing costs),
- (ii) the high cost and difficulty in obtaining real-world KGs in data-scarce domains limits KG-driven solutions in those domains [53],
- (iii) correlations present in large crowdsourced KGs limit the targeted evaluation of graph structures [90], and
- (iv) the open benchmarking with KGs from sensitive domains is limited as they contain private data [36, 53, 52].

The advantages of using synthetic data in addressing data scarcity, data denoising, and data privacy have already been thoroughly studied across domains [85, 127, 129, 106, 124]. While foundational generators such as the Erdős-Rényi [32], Watts-Strogatz [121], and the Barabási-Albert [7] models laid the groundwork for synthetic graph generation, they produce graphs that lack the structure and complexity of KGs. Synthetic Knowledge Graphs (SKGs) therefore require specialized generators, henceforth referred to as Synthetic Knowledge Graph Generators (SKGGs), to encode realistic constraints. SKGGs are typically designed to mirror characteristics of real-world KGs in the generated KGs to ensure that subsequent performance assessments are reliable. Note that the task of synthetic KG generation differs from KG construction in that the created graph relies on algorithmic generation [123] rather than being derived from unstructured or semi-structured data sources.

SKGGs are being used to fulfill the need for high-quality data of benchmarking systems since the 2000s [112, 45, 120, 88, 97, 29, 6, 126, 90, 41, 55, 73, 33, 91]. Despite the, at the time, modest number of large KGs [45], KGs already backed multiple systems, and an increase in the uptake of KGs was expected [120]. Evaluating existing systems was crucial to attract more attention to the field by supplying interested parties with a fair comparison. At the same time, it was also important to provide a platform for newly developed KG-driven solutions to establish themselves. SKGGs were introduced to generate large synthetic graphs, enabling system benchmarking under heavy workloads that reflect future demands. While then, suitable large datasets were far and few [45], nowadays KG datasets grow at a rapid pace [97], as are the examples of the DBpedia, Wikidata, UniProtKB, and the Google KGs. DBpedia started in 2007 with 103 million triples [4] and has grown to over 1 billion in its latest core release.² Similarly, Wikidata grew from 43 million triples [117] in 2014 to over 1.5 billion triples.³ UniProtKB [25] started in 2003 with over 800 thousand entries [14] and grew to ~ 200 million entries in the latest release. The latest release represents over 217 billion RDF triples.⁴ Google Knowledge Graph started in 2012 with 3.5 billion triples and grew to over 500 billion triples by 2020.⁵ Considering the exponential data growth rate, systems reliant on knowledge graphs may quickly become obsolete. SKGGs make it possible to design effective benchmarking systems that not only assess how systems currently perform against one another but also predict how these systems will perform over time, as datasets are likely to

² DBpedia statistics from <https://www.dbpedia.org/resources/latest-core/>

³ Wikidata statistics available at <https://grafana.wikimedia.org/d/000000175/wikidata-datamodel-statement>

⁴ UniProtKB statistics from <https://sparql.uniprot.org/.well-known/void>

⁵ Google Knowledge Graph statistics from <https://blog.google/products/search/introducing-knowledge-graph-things-not/> and <https://blog.google/products/search/about-knowledge-graph-and-knowledge-panels/>

grow. The dataset scaling ability of SKGGs also revealed a complementary role beyond scalability testing. Generating smaller graphs creates representative examples of larger graphs, enabling the reduction of correlations [90] and redundancy [54] present in real-world KGs. Generating smaller graphs allows the isolation of specific structures in the graph and a consequent targeted evaluation under controlled conditions. By generating SKGs of arbitrary size, SKGGs demonstrate their versatility as a benchmarking tool, balancing the need for scalability testing and the need for targeted evaluation. Another key advantage of SKGGs lies in their ability to address privacy constraints by generating graphs with synthetic entities that do not correspond to real-world data. Private data is common in sectors such as healthcare and finance, as data in medical and financial records is often highly sensitive, private, and sometimes even proprietary. SKGGs can ensure that data remains private, while preserving structural properties [52].

Given the versatility and growing importance of SKGGs, and the absence of a recent survey focused exclusively on the KG domain, a systematic review of the field is essential to consolidate findings. This survey focuses on SKGGs developed between 2014 and 2025 and presents the following contribution:

- We provide a systematic review of the current state of the art, focused on solutions tailored to KGs.
- We classify the SKGGs based on their data generation methodology, generation requirements, and temporal coverage.
- We discuss the advantages and disadvantages of approaches in each category.
- We provide an overview of all the evaluation tasks used by SKGGs in the survey timeframe and respective metrics.
- We summarize the technical details of each approach.
- We present current issues and possible research opportunities.

This paper is structured as follows: Section 2 introduces the required background knowledge and notation necessary to understand the rest of the paper. Section 3 presents the methodology behind this survey and the inclusion criteria for related publications. Section 4 proposes a categorization of the SKGG algorithms and defines these categories. Section 5 details the approaches in each category. Section 6 elaborates on the evaluation tasks and performance metrics of the approaches mentioned herein. Section 7 presents the practical implementation details of the approaches. We summarize our findings and identify open research questions in Section 8. Finally, Section 9 concludes this survey.

2 Preliminaries

This section introduces the notation and core concepts used in this survey. We present common background knowledge and terminology relevant to graph generation research.

► **Definition 1** (Knowledge Graph). *A Knowledge Graph is a data graph where the nodes represent entities, and the edges represent relations between these entities [49].*

Knowledge graphs are often modelled using the Resource Description Framework (RDF)⁶ standard.

► **Definition 2** (RDF Knowledge Graph). *Let $\mathbb{I}$ denote the set of all RDF Internationalized Resource Identifier (IRI) nodes, $\mathbb{B}$ the set of all anonymous resources, i.e., blank nodes, $\mathbb{L}$ the set of all literals, and $\mathbb{P}$ the set of all RDF predicates with $\mathbb{P} \subseteq \mathbb{I}$. $\mathbb{I}$, $\mathbb{B}$, and $\mathbb{L}$ are pairwise disjoint sets.*

⁶ <https://www.w3.org/TR/rdf11-primer/>

The set of all RDF resources is defined as $\mathbb{K} = \mathbb{I} \cup \mathbb{B}$. An RDF triple (s, p, o) represents a subject s connected to an object o via a predicate p . We define an RDF Knowledge Graph as a finite set of RDF triples [79]:

$$\mathcal{G} = \{(s, p, o) \mid s \in \mathbb{K}, p \in \mathbb{P}, o \in \mathbb{K} \cup \mathbb{L}\}. \quad (1)$$

Resources in an RDF KG are typed using the `rdf:type`⁷ predicate, which links each resource to a set of classes. The set of classes actually used in a graph forms its type system.

► **Definition 3** (Type System). *The type system $\mathcal{T}$ of a graph $\mathcal{G}$ is the set of classes defined by its instances in the corresponding graph [31]:*

$$\mathcal{T} = \{C \mid \text{rdf:type}(x, C) \in \mathcal{G}\} \quad (2)$$

Next to the general definition of RDF graphs presented in Equation 1, various ways to represent RDF data are used in practice [26, 75, 49]. For this survey, we introduce four of these representations – directed edge-labeled graphs [49], bipartite graphs [1], multiplex graphs [101], and binary tensors [29].

► **Definition 4** (Directed Edge-Labeled Graph). *A directed edge-labeled graph representing an RDF KG is defined as $\mathcal{G}^* = (\mathbb{V}_{src}, \mathbb{V}_{trg}, \mathbb{E}, \mathbb{Y}, \lambda)$ with a source vertex set $\mathbb{V}_{src} \subseteq \mathbb{I} \cup \mathbb{B}$, a target vertex set $\mathbb{V}_{trg} \subseteq \mathbb{I} \cup \mathbb{B} \cup \mathbb{L}$, an edge set $\mathbb{E} \subseteq \mathbb{V}_{src} \times \mathbb{V}_{trg}$ representing the directed edges, a finite set of labels $\mathbb{Y}$, and a labeling function $\lambda : \mathbb{E} \mapsto \mathbb{Y}$. The literal nodes $\mathbb{L}$ can only represent terminal nodes in the graph. The complete set of nodes in $\mathcal{G}^*$ is denoted by $\mathbb{V} = \mathbb{V}_{src} \cup \mathbb{V}_{trg}$.*

► **Definition 5** (Bipartite Graph). *A bipartite graph is a graph with a vertex set that can be divided into two disjoint subsets $\mathbb{S}$ and $\mathbb{O}$ with edges that start in one subset and end in the other [15].*

► **Definition 6** (Multiplex Graph). *A multiplex graph is a graph with multiple layers, where each layer represents different relations between the same vertices [11]. A multiplex graph $\mathcal{M}$ with k layers is defined as $\mathcal{M} = (\mathbb{V}, (\mathbb{E}_1, \dots, \mathbb{E}_k))$ with a shared vertex set $\mathbb{V}$ and k sets of edges [101].*

► **Definition 7** (Binary Tensor Representation of a Knowledge Graph). *A KG can be represented as a three-mode binary tensor $\mathcal{X} \in \{0, 1\}^{n \times n \times m}$, where n is the number of nodes and m the number of relations in the KG. An element $\mathcal{X}_{ijk}$ has the value 1 if the corresponding triple exists in the KG:*

$$\mathcal{X}_{ijk} = \begin{cases} 1, & \text{if } (s_i, p_k, o_j) \in \mathcal{G}, \\ 0, & \text{otherwise,} \end{cases} \quad (3)$$

where p_k denotes the relation at index k , and s_i and o_j denote the nodes at indices i and j , respectively [29].

KGs can also encode temporal information, capturing when facts hold. We follow the convention of Polleres et al. [89] and refer to KGs that do not encode any temporal information as *static* and those that do as *evolving* [89, 98]. Some examples of evolving KGs are versioned, temporal, and dynamic KGs [89]. Let $\mathcal{G}_\Delta$ denote an evolving KG.

⁷ We will henceforth use the `rdf` namespace prefix for the RDF namespace <http://www.w3.org/1999/02/22-rdf-syntax-ns#>, `rdfs` to refer to resources defined in the Resource Description Framework Schema (RDFS) namespace <http://www.w3.org/2000/01/rdf-schema#>, and the empty namespace prefix for example resources.

► **Definition 8** (Versioned Knowledge Graph). *A versioned KG is a set of N static KGs $\mathbb{G} = \{\mathcal{G}_0, \mathcal{G}_1, \dots, \mathcal{G}_N\}$, where each KG $\mathcal{G}_i$ represents a graph snapshot valid under the respective version i [56, 69].*

► **Definition 9** (Temporal KG). *A temporal KG extends an RDF KG by associating each triple with temporal information [66]. Let $\mathbb{T}$ denote the set of all valid timestamps. A temporal KG is defined as a finite set of quadruples:*

$$\mathcal{G}_\tau = \{(s, p, o, \tau) \mid s \in \mathbb{K}, p \in \mathbb{P}, o \in \mathbb{K} \cup \mathbb{L}, \tau \in \mathbb{T}\}, \quad (4)$$

where τ can represent either a time point $t \in \mathbb{T}$ or a time interval $[t_s, t_e]$ with $t_s, t_e \in \mathbb{T}$ and $t_s \leq t_e$.

► **Definition 10** (Dynamic KG). *A dynamic KG is represented as a sequence of timestamped atomic changes with $\tau_1 \leq \tau_2 \leq \dots$ and $\tau_i \in \mathbb{T}$, where each change is either an addition or a deletion of a triple.*

KGs can be characterized by various metrics, ranging from simple size statistics such as the number of nodes and edges, to more complex structural properties such as the diameter and the clustering coefficient [37]. Among many properties, the node degree is one of the most commonly leveraged by the KG generation community.

► **Definition 11** (Node Degree). *The degree of a vertex v is the number of triples incident to v [15], $\delta(v) = |\{(s, p, o) \in \mathcal{G} \mid s = v \vee o = v\}|$. The average degree of a directed graph is calculated as $\bar{\delta}(\mathcal{G}) = \frac{|\mathcal{G}|}{|\mathbb{V}|}$. The degree concept of directed graphs is also expanded to in- and out-degree. The in-degree of a vertex $\delta_{in}(v)$ denotes the number of incoming triples to v , $\delta_{in}(v) = |\{(s, p, v) \mid (s, p, v) \in \mathcal{G}\}|$ [37]. Similarly, the out-degree of a vertex $\delta_{out}(v)$ denotes the number of outgoing triples from v , $\delta_{out}(v) = |\{(v, p, o) \mid (v, p, o) \in \mathcal{G}\}|$ [37].*

KGs exhibit different in- and out-degree distributions, where the in-degree distribution is typically more skewed than the out-degree distribution. Nodes with high in-degrees are often dubbed *hubs*, and those with high out-degrees *stars* [37].

Beyond the instance level, KGs can be complemented by a schema that describes the underlying domain. It provides a conceptual framework against which instance data is interpreted and used [49].

► **Definition 12** (Schema). *A schema is formally defined as a set $\mathcal{S} = \{\mathcal{C}, \mathcal{P}, \mathcal{A}\}$ [53], where:*

- (i) $\mathcal{C}$ is the set of concepts or classes that define the types of entities in the data graph, e.g., `:Person`,
- (ii) $\mathcal{P}$ is the set of properties or predicates that relate the entities in the data graph, e.g., `:birthPlace`, and
- (iii) $\mathcal{A}$ is the set of axioms or logical statements that constrain concepts and roles, e.g., `rdfs:subClassOf(:City, :Place)`.

► **Definition 13** (Ontology). *An ontology is a framework that conceptualizes the domain [27, 49] by specifying rules and restrictions. Two commonly used ontology languages are the RDFS and the Web Ontology Language (OWL). We follow the definitions provided by Baader et al. [5] and denote interpretation as $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with a non-empty set $\Delta^{\mathcal{I}}$ and a mapping function $\cdot^{\mathcal{I}}$ to map a concept or role to its respective set. The mapping function maps a concept $C \in \mathcal{C}$ to a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and a role E to a binary relation $E^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.*

Below, we outline key axioms that models abiding by RDFS should fulfill as described in [114]:

- The predicate `rdfs:subClassOf` indicates class subsumption, i.e., all instances of a subclass A are also instances of a superclass B :

$$\text{rdfs:subClassOf}(A, B) \rightarrow A^{\mathcal{I}} \subseteq B^{\mathcal{I}}. \quad (5)$$

- The predicate `rdfs:domain` expresses a property's domain and dictates that all subjects of a given predicate E belong to a concept C . Let x and y denote individuals:

$$\text{rdfs:domain}(E, C) \rightarrow (\forall x, y : \langle x, y \rangle \in E^{\mathcal{I}} \rightarrow x \in C^{\mathcal{I}}). \quad (6)$$

- The predicate `rdfs:range` expresses a property's range and dictates that all objects of a given predicate E belong to a concept C :

$$\text{rdfs:range}(E, C) \rightarrow (\forall x, y : \langle x, y \rangle \in E^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}). \quad (7)$$

While RDFS can be used to define a lightweight ontology with limited capabilities and basic constructs, OWL expresses complex relations through additional axioms [47]. Ontologies enable knowledge inference through their axioms, but they do not enforce them. The process of extending a KG with facts deduced as logical consequences from the application of an ontology's axioms is referred to as *graph materialization* [47]. Commonly leveraged axioms from the OWL ontology express predicate transitivity, inverse properties, and predicate symmetry⁸:

- `owl:TransitiveProperty` is the class of transitive properties:

$$\text{owl:TransitiveProperty}(E) \rightarrow (\forall x, y, z : (\langle x, y \rangle \in E^{\mathcal{I}} \wedge \langle y, z \rangle \in E^{\mathcal{I}}) \rightarrow \langle x, z \rangle \in E^{\mathcal{I}}). \quad (8)$$

- `owl:SymmetricProperty` is the class of symmetric properties:

$$\text{owl:SymmetricProperty}(E) \rightarrow (\forall x, y : \langle x, y \rangle \in E^{\mathcal{I}} \rightarrow \langle y, x \rangle \in E^{\mathcal{I}}). \quad (9)$$

- `owl:inverseOf` indicates that two properties are inverse:

$$\text{owl:inverseOf}(E, F) \rightarrow (\forall x, y : \langle x, y \rangle \in E^{\mathcal{I}} \rightarrow \langle y, x \rangle \in F^{\mathcal{I}}). \quad (10)$$

OWL is based on Description Logics (DLs), a family of formal logics for representing concepts and their relationships.

► **Definition 14** (Description Logic). *DLs are a family of logic-based knowledge representation languages [5]. A DL models concepts, roles, and their relationships. It further allows the representation of richer concept descriptions using atomic concept constructors and role constructors. For example, the DL expression $\text{Male} \sqcap \exists \text{hasChild}. \text{Person}$ defines the concept of father by combining simpler concepts. This expression results in a new concept that represents individuals who are male and have at least one child.*

Given the rule-based nature of KGs and their underlying domains, a variety of logic-based formalisms are commonly adopted by the KG community. Horn rules are one such formalism, used to specify inference patterns.

⁸ We invite the interested reader to consult the full list of axioms at <https://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/>.

► **Definition 15** (Horn rule). *An atom is a fact of the form $(x \ p \ y)$, where the predicate p is always bound, i.e., not a variable. An atom is instantiated when one of the arguments x or y is bound, e.g., $(:VincentVanGogh :studiedIn \ ?y)$. If both arguments are bound, the atom is considered ground, e.g., $(:VincentVanGogh :studiedIn :RoyalAcademy)$. A Horn rule $\mathbf{B} \implies H$ is an implication between a body $\mathbf{B}$ and a head atom H [107]. The head is a single atom, while the body is a set of atoms represented by a conjunctive query of atoms $(B_1 \wedge \dots \wedge B_n)$. For example, $(?x :studiedIn ?y) \wedge (?z :studiedIn ?y) \implies (?x :colleague ?z)$ represents a Horn rule.*

3 Survey Methodology

In this section, we present our methodology by comparing our survey to existing ones and defining its scope. We also present the search methodology and inclusion criteria for finding and selecting the approaches discussed.

3.1 Related Surveys

The majority of surveys on graph generators focus on general graph generators [67], with much of the research in recent years being centered on deep learning solutions [34, 133, 44]. There are also other surveys on the generation of specific types of graphs, e.g., hypergraphs [63]. However, these surveys do not account for KGs. To the best of our knowledge, the only graph generation survey that includes KGs was presented by Bonifati et al. [17]. Bonifati et al. cover multiple types of graphs and thus provide an analysis that accommodates all graph types considered. Tangent related work includes surveys on KG construction from unstructured and semi-structured data [130, 131], as well as benchmarking surveys [16, 82] that describe several known benchmarks with data generators. However, KG generation is not the focus of any of these surveys. Our survey focuses on a more recent timeframe and overlaps minimally with Bonifati’s survey, sharing only 4 of the 38 generators we cover. Our survey distinguishes itself by focusing exclusively on KG generators, exposing challenges inherent to the knowledge graph domain. This survey also presents the current landscape of the performance evaluation of synthetic KGs and the implementation details of KGs generators.

3.2 Survey Scope

This survey focuses on SKGGs, i.e., graph generators that can generate synthetic KGs. This includes generators that explicitly incorporate KG semantics, as well as broader graph generators that have been shown to model KGs successfully, e.g., the authors directly include KG generation experiments in their publications.

3.2.1 Search databases

We searched for relevant publications in the following research databases:

- Google Scholar⁹
- OpenAlex¹⁰
- IEEE Xplore Digital Library¹¹
- Science Direct¹²

The results were exported directly from the databases or obtained using the *scholarly* [22] library.

⁹ <https://scholar.google.com/>

¹⁰ <https://openalex.org/>

¹¹ <https://ieeexplore.ieee.org/>

¹² <https://www.sciencedirect.com/>

■ **Table 1** Inclusion and exclusion criteria.

Criteria	Inclusion Criteria	Exclusion Criteria
C1 Publication Date	The publication is published between January 2014 and May 2025.	The publication is published before 2014 or after May 2025.
C2 Language	The publication is written in English.	The publication is written in a language other than English.
C3 Publication Type	The publication presents a new approach, algorithm, or method.	The publication is a review article, e.g., a survey.
C4 Publishing	The publication is peer-reviewed.	The publication is not peer-reviewed, e.g., theses, presentations, reports, and discussions.
C5 Accessibility	The publication can be accessed from a University without additional payment.	The publication cannot be accessed without additional payment.
C6 Graph Generation Scope	The publication presents graph generators that produce synthetic KGs.	The publication presents approaches in other topics, e.g., KG construction from text or semi-structured data.

3.2.2 Search keywords

The search query targeted graph generators by including the keywords ("graph generator" OR "dataset generator" OR "synthetic graphs" OR "graph generation" OR "data generator"). We complement the previous query with ("knowledge graph" OR "semantic web" OR "linked data" OR "RDF" OR "ABox" OR "A-Box") to target publications from the knowledge graph domain.

3.2.3 Selection Strategy

The inclusion and the exclusion criteria are presented in Table 1. These criteria ensure that only publications aligning with the survey's scope are considered. The survey includes publications that satisfy all of the inclusion criteria and none of the exclusion criteria.

3.2.4 Search Methodology

Our selection process consisted of five steps. First, the relevant publications were retrieved from the previously listed databases using the search query specified in Section 3.2.2. Second, we de-duplicated the search results using Zotero, a reference management tool that detects duplicate publications based on the bibliographic components.¹³ Third, we scanned the articles' metadata to enforce criteria C1–C5. Fourth, we reviewed the paper's titles and abstracts to exclude articles unrelated to our primary focus and enforce criterion C6. Lastly, we performed citation chaining on the remaining papers to ensure the thorough identification of relevant approaches. The citation chaining steps included backward and forward snowballing of the papers that survived Phase 4. The number of remaining publications after each phase is presented in Table 2.

¹³<https://www.zotero.org/>

■ **Table 2** Number of publications remaining after each phase.

	Methodology Phase	Publications
1	Use search query with the timeframe 2014-2025	11660
2	De-duplicate the publications	9725
3	Validate metadata (Criteria C1-C5)	7972
4	Validate content (Criterion C6)	36
5	Citation chaining (Criterion C6)	38

4 Categorization

This section presents a categorization of SKGGs and answers three questions: which methods SKGGs employ to generate a synthetic knowledge graph, which data the generation requires, and the temporal coverage of the output graph. Together, these three dimensions enable a structured comparison, covering the generation process, the input, and the output. The resulting categorization is presented in Figure 2, where the approaches are grouped by methodology, colour-coded by their input requirements, and shape-coded by the temporal coverage. We first establish a definition of an SKGG. Subsequently, we analyze the three categorization dimensions in the following order:

- (i) generator input,
- (ii) generation methodology, and
- (iii) temporal coverage of the output graph.

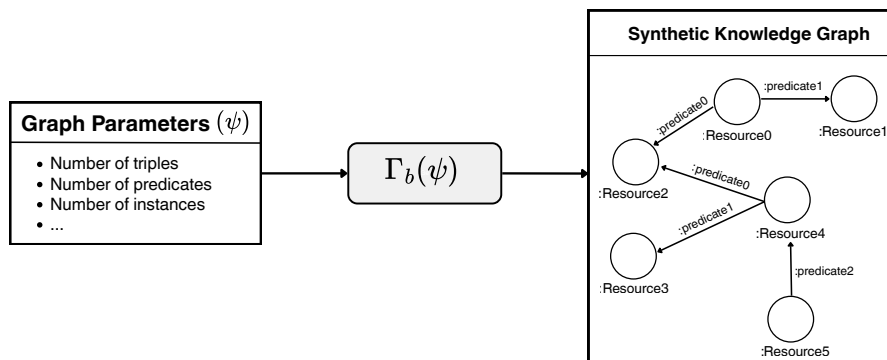
► **Definition 16** (Synthetic Knowledge Graph Generator). *A Synthetic Knowledge Graph Generator, denoted by Γ , is a generation process that takes an input $\mathcal{I}$ and applies a method $\mathcal{M}$ to produce a Synthetic Knowledge Graph $\bar{\mathcal{G}}$.*

4.1 Generator Input

This category characterizes the input $\mathcal{I}$ required by the generators.

► **Definition 17** (Basic Synthetic Knowledge Graph Generator). *An SKGG, denoted by Γ_b , generates a synthetic knowledge graph $\bar{\mathcal{G}}$ based on a set of input parameters ψ .*

Examples of parameters in ψ are the number of triples, nodes, predicates, and the desired vertex degree distribution of the synthetic graph. Figure 1 depicts such a generation process.



■ **Figure 1** Example of an SKGG taking only feature parameters as input.

While some generators only require feature parameterization [1, 126, 96, 84], others require additional input files. We identified two types of such input in our survey: schemas and reference KGs.

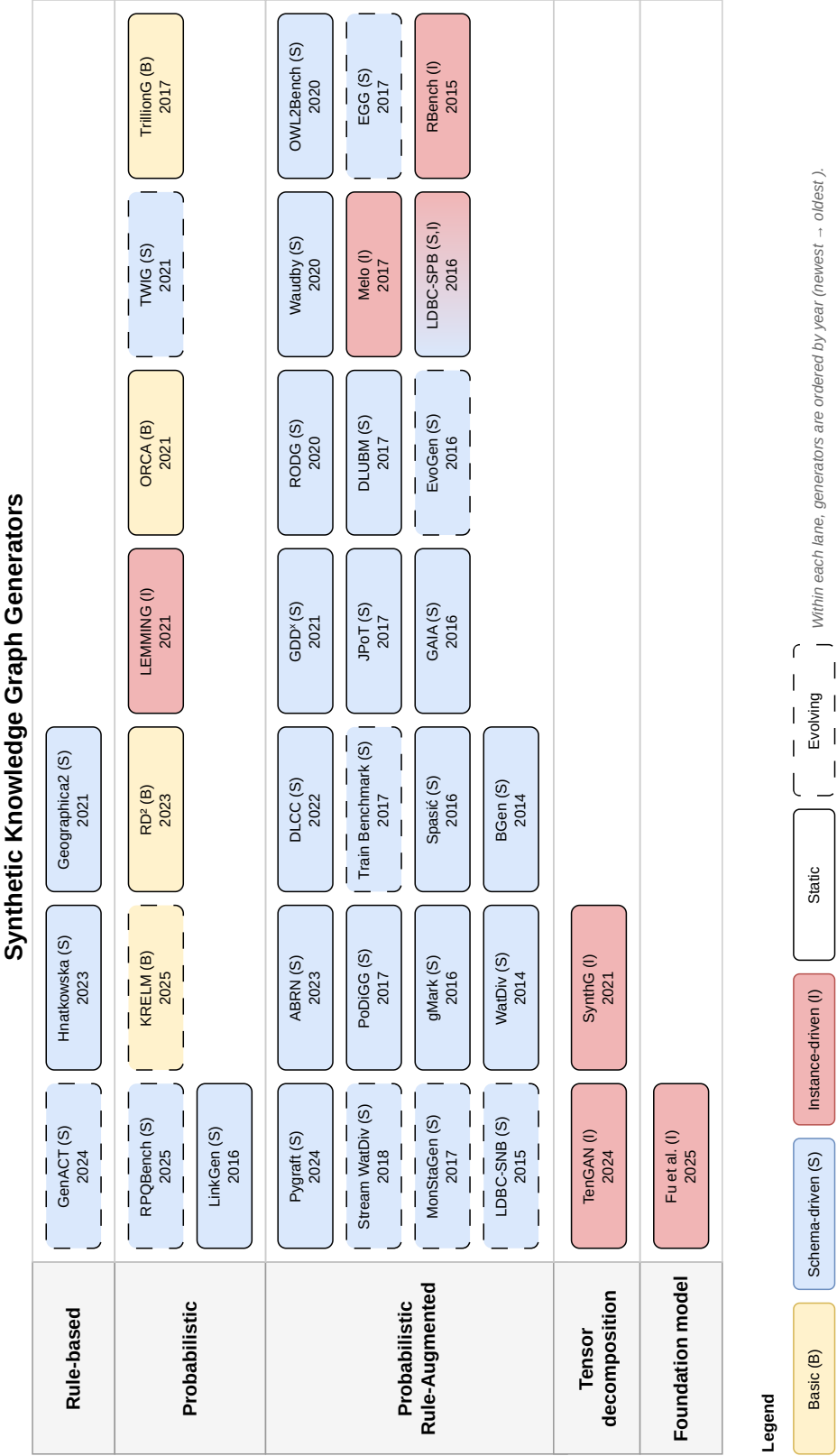
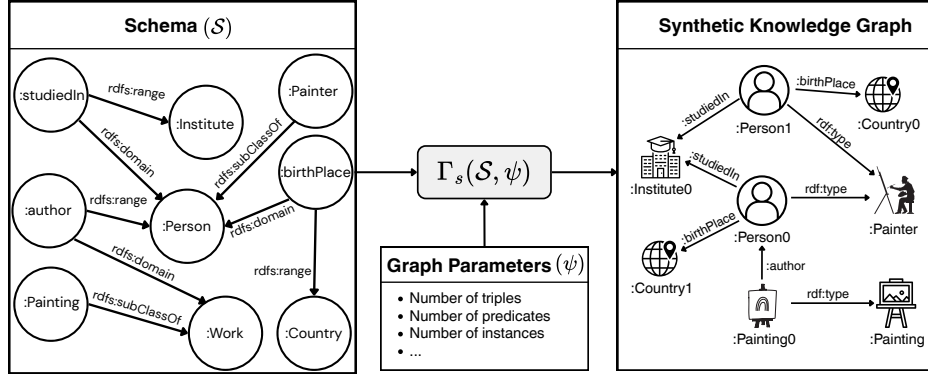


Figure 2 Classification of Synthetic Knowledge Graph Generators.

These two categories, schema-driven [53, 102, 118, 48, 132, 90, 36, 55, 105, 20, 103, 122, 109, 59, 2, 6, 116, 108, 18, 71, 93, 57, 104, 33, 61, 3, 40] and instance-driven [97, 29, 73, 91, 38, 101], are not disjoint, as a generator may require both reference KGs and schemas, as is the case of LDBC-SPB [60].

► **Definition 18** (Schema-driven Synthetic Knowledge Graph Generator). *We generalize the definition of [6] by defining a schema-driven generator Γ_s as one that produces an SKG $\bar{\mathcal{G}}$ based on a set of parameters ψ and an input schema $\mathcal{S}$.*

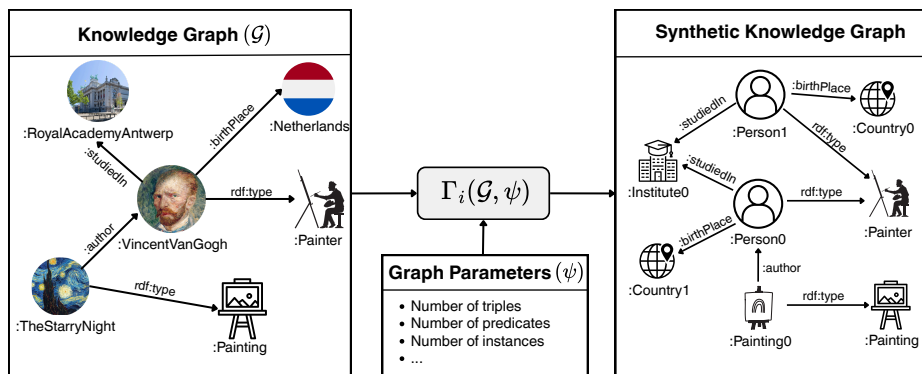


■ **Figure 3** Example of a schema-driven SKGG.

The input schema can be defined manually, through configuration files, or as an ontology. The latter is a solution more aligned with the KG domain. Figure 3 depicts the graph generation process using an ontology as an input schema, showing how the output graph obeys the constraints defined by the ontology.

► **Definition 19** (Instance-driven Synthetic Knowledge Graph Generator). *An instance-driven generator Γ_i takes a set of reference KGs $\mathbf{G}_{ref}$ with $|\mathbf{G}_{ref}| \geq 1$ and a set of parameters ψ as input, producing an SKG $\bar{\mathcal{G}}$.*

An instance-driven graph generator aims to produce a synthetic graph that mimics a selected set of features of the reference KGs. It does not replicate the input graph. The mimicked features can be structural characteristics of the graph, e.g., in- and out-degree distributions, clustering coefficients, degree centrality, or statistical properties, e.g., predicate and vertex type co-occurrences. Figure 4 depicts an example of an instance-driven SKGG.



■ **Figure 4** Example of an instance-driven SKGG.

4.2 Generation Methodology

The generation methodology refers to the methods $\mathcal{M}$ used by graph generators to create the synthetic graph. This survey identifies and defines five main methodology types: rule-based [55, 48, 102], probabilistic [118, 1, 126, 97, 96, 105, 84, 57], probabilistic rule-augmented [53, 132, 90, 36, 122, 20, 103, 40, 2, 116, 109, 108, 18, 59, 73, 71, 104, 6, 60, 93, 33, 91, 61, 3], tensor decomposition-based [29, 101], and foundation model-based generators [38].

► **Definition 20** (Rule-based Synthetic Knowledge Graph Generator). *Rule-based knowledge graph generation is a category of approaches that systematically leverage explicit rules to produce graphs. These rules define conditions that must be satisfied by the generation process. The rule-based generation process Γ_r iteratively applies a set of rules $\mathcal{R}$.*

Examples of such rules include Horn rules, DL reasoning, hard-coded patterns (e.g., templates and fixed cardinalities), and deterministic computations (e.g., geometric formulas for distance and size calculation).

► **Definition 21** (Probabilistic Synthetic Knowledge Graph Generator). *Probabilistic knowledge graph generation is a category of approaches that leverage probability distributions to model graph properties. The graph generation process Γ_p is achieved by iteratively sampling from a distribution P_θ .*

The probability distribution and its parameters θ can be fixed, defined by the user, or estimated from a reference KG. The latter is typical of instance-driven generators. The node degree is the property most commonly modelled by distributions.

► **Definition 22** (Probabilistic Rule-Augmented Synthetic Knowledge Graph Generator). *Probabilistic rule-augmented knowledge graph generation is a hybrid category of approaches that leverage probabilistic distribution modelling with explicit rules to guide the graph generation process. Approaches in this category combine techniques from Rule-based Graph Generation (20) and Probabilistic Graph Generation (21).*

There are multiple ways to combine probabilistic modelling with logical rules. Rules can be applied first to constrain the graph structure, with probabilistic sampling filling in the remaining elements. Alternatively, sampling from a distribution can proceed until the resulting graph satisfies the rules. The probability parameters θ may also be defined by the application of a rule, e.g., hard-coded cardinality ranges may determine the parameters of a probability distribution.

► **Definition 23** (Tensor Decomposition-based Synthetic Knowledge Graph Generator). *Tensor decomposition-based knowledge graph generation employs tensor factorization methods to model and generate graphs. The KG is represented as a binary tensor, which is decomposed into a set of factor matrices. The output graph is generated by leveraging the factorized representations of the binary tensor.*

Tensor decomposition models like RESCAL [81] are known to model multilinear relationships between entities and relations, while others, such as InfTucker [125], can capture non-linear interactions.

► **Definition 24** (Foundation Model-based Synthetic Knowledge Graph Generator). *Foundation model-based knowledge graph generation employs a pre-trained foundation model to generate synthetic KGs. The generation process prompts the model to produce graph elements, e.g., entities, relations, and triples, which are then parsed and assembled into the output SKG.*

Foundation models have been shown to encode broad world knowledge and structured information, making them attractive candidates for KG generation. Current approaches predominantly rely on Large Language Models (LLMs), which use natural language as an intermediate knowledge representation.

4.3 Temporal Coverage

Temporal coverage refers to the ability of the generators to produce SKGs with temporal information, e.g., evolving KGs. The output SKG $\tilde{\mathcal{G}}$ can either be a single static graph $\tilde{\mathcal{G}}_o$ or an evolving graph $\tilde{\mathcal{G}}_\Delta$. While most surveyed approaches focus on generating static KGs, a growing number of generators target evolving KGs [1, 102, 118, 105, 122, 2, 116, 108, 71, 33, 40]. We intentionally merge the different types of evolving KGs into a single category, as we consider the first-level distinction between static and evolving to be the most consequential for generator design.

► **Definition 25** (Static Synthetic Knowledge Graph Generator). *A static SKGG produces a single static SKG snapshot $\tilde{\mathcal{G}}_o$.*

► **Definition 26** (Evolving Synthetic Knowledge Graph Generator). *An evolving SKGG produces an evolving SKG $\tilde{\mathcal{G}}_\Delta$.*

5 Detailed System Comparison

This section first outlines common features across generators regarding generation parameters, highlighting the advantages and disadvantages of basic, schema, and instance-driven approaches. It then details the generation methodology, where each approach is explained in length. Approach-specific definitions are also provided here.

5.1 Common Features

This subsection addresses shared characteristics across SKGGs. We introduce common features to provide a consistent comparison framework to highlight the strengths and weaknesses of each approach. Across the generator approaches considered in this survey, standard features included the domain scope, the types of nodes the approaches can generate, and the presence of post-generation optimization processes.

Domain Scope. The domain scope refers to the domain on which an approach operates. It determines whether an approach is tailored to a fixed schema or designed to be general-purpose. Domain-specific generation enables specialization for a single area by leveraging experts' domain knowledge to address specific requirements, e.g., considering tailored constraints for each property in a schema. In contrast, domain-agnostic approaches offer flexibility across multiple domains.

Node Type Support. The node type support refers to the type of nodes the generator is capable of creating in the generated graph, i.e., IRI, blank, and literal nodes. While generating IRI nodes is required since they form the backbone of knowledge graphs, not all approaches explicitly consider literal or blank nodes. Literal node support is more common than blank node support among the generators.

Optimizer. The optimizer refers to the integration of an optimizing process after the graph generation workflow. An optimizer is an iterative process that refines the generated synthetic graph by minimizing a target function. The function measures the difference between the current and the desired graph state. Optimizers address post-generation discrepancies but introduce a computational overhead.

5.1.1 Basic Knowledge Graph Generation

In their base form, SKGGs require neither a schema nor a reference KG as outlined in Def. 17. This subsection examines their common features and discusses their respective strengths and limitations. The summary of purely feature-driven approaches is presented in Table 3. Given their feature-driven approach, all generators—KRELM [1], RD² [126], ORCA [96], and TrillionG [84]—are domain-agnostic. Of these, KRELM [1] is the only evolving SKGG as it simulates the growth of KGs over time. Regarding node support, KRELM [1], TrillionG [84] and ORCA [96] do not generate literal nodes, whereas RD² [126] can generate random literals based on a specific datatype. Additionally, RD² [126] treats blank nodes identically to IRI nodes during generation. Examples of parameters in ψ are size metrics – number of triples, number of entities, number of triples per entity, number of predicates – and distribution parameters like datatype creation probabilities.

These approaches offer several practical advantages. They require minimal configuration, which enables quick prototyping. Furthermore, their lightweight design and fast runtimes make them suitable for large-scale KG generation. However, these benefits come with structural limitations. They often fail to capture the complexity and diversity of real-world KGs, as they operate under fixed assumptions and lack the fine-grained control over multiple graph characteristics that other SKGG categories have. Because such generators require minimal parameterization, the remaining structural properties are determined implicitly by the generation process rather than specified by the user. For example, if the exposed parameters primarily govern the graph’s scale, properties such as the degree distribution emerge from the generative mechanism rather than being directly controllable, and other KG-specific aspects such as schema consistency are disregarded.

5.1.2 Schema-driven Knowledge Graph Generation

Building on the schema-driven SKGGs definition in Def. 18, this subsection compares schema-driven approaches along common features across generation paradigms and schema-specific characteristics. It concludes by discussing their respective advantages and disadvantages. An overview of the schema-driven approaches is presented in Table 4.

While most recent approaches are domain-agnostic, several earlier works target specific domains such as geospatial [55], public transportation [109, 108], academic data [59, 71, 103], semantic publishing [60], social networks [104, 33, 61, 102, 105, 122], product specification [3], and citation data [118]. Domain-specific generation leverages well-known constraints within a domain. For example, geometric equations for geospatial data generation. On the other hand, domain-agnostic approaches can be reused for different domains and can generate graphs without expert knowledge [53, 48, 132, 90, 36, 20, 2, 6, 116, 18, 93, 57]. The node support varies significantly among domain-agnostic and domain-specific approaches. Almost all approaches support the generation of literal nodes except for Pygraft [53], DLCC [90], and ABRN [132]. While literal node generation is widespread, the techniques employed by each approach differ. Common methods

■ **Table 3** General features of basic approaches sorted descendingly by year. Δ marks evolving SKGGs, ● represents domain-agnostic approaches, and ○ represents domain-specific approaches.

Approaches	Node Support		Optimizer	Domain Scope
	Blank	Literal		
Δ KRELM [1]	✗	✗	✗	●
RD ² [126]	✓	✓	✗	●
ORCA [96]	✗	✗	✗	●
TrillionG [84]	✗	✗	✗	●

include generating random literals based on the literal’s datatype [93, 57, 20, 103], sampling from defined distributions [36, 6, 116, 2, 3, 40, 118, 105, 103], using an incrementing index value [59, 71], and leveraging vocabularies [60, 33, 102, 48, 118, 103, 122]. Some domain-specific approaches also tie the literal value generation to the domain [55, 109, 108]. In contrast, blank node support is less comprehensive. BGen [61] is the only schema-driven approach considering a connectivity pattern for blank nodes separate from IRI nodes. Other approaches support blank node generation, but do not consider additional constraints compared to IRI nodes [118]. Furthermore, while most approaches generate a single static graph snapshot, some generators produce evolving graphs, as seen with GenACT [102], RPQBench [118], TWIG [105], Waudby [122], EGG [2], MonStaGen [116], TrainBenchmark [108], EvoGen [71], LDBC-SNB [33], and Stream WatDiv [40]. Notably, none of the schema-driven approaches incorporate optimization processes.

Schema-driven approaches can be further distinguished based on three criteria:

- (i) whether they accept user-provided ontologies as an input schema,
- (ii) whether they generate ontologies, and
- (iii) whether they include consistency checks to verify the alignment between the schema and the generated KG.

While a schema provides the structural information necessary for the graph generation, each approach has a different implementation for the concept. Some approaches require only user-provided parameters to define the schema, often in the form of program arguments and configuration files, while other approaches also accept OWL or RDFS ontologies. Hnatkowska [48], ABRN [132], RODG [20], JPoT [18], GAIA [93], and LinkGen [57] are approaches that require ontologies as input. However, despite the approaches accepting OWL ontologies as input, they vary in the supported OWL profiles. When schemas are specified by user-defined parameters, some methods first generate an ontology before generating the KG. For example, DLCC [90] and Pygraft [53] both generate an ontology as a preliminary step, but they differ in the supported constructs. DLCC supports a few core RDFS constructs, such as class hierarchy, domain, and range. Pygraft supports a broader range of RDFS and OWL constructs, including class and property hierarchy, domain, range, class disjointness, inverse properties, and properties like symmetry, reflexivity, transitivity, functionality, and their respective negations. As schema-driven SKGGs, the approaches assume the KG and the schema are consistent by construction. JPoT [18], TrainBenchmark [108], RODG [20], ABRN [132], Hnatkowska [48], and Pygraft [53] include additional consistency checks, often via reasoners. An exception is TrainBenchmark [108], which detects inconsistencies via graph pattern matching.

Schema-driven approaches have multiple advantages. Guided by a detailed schema, schema-driven approaches benefit from a fine-grained control over the desired graph structure, where graph properties can be explicitly specified. This level of control is valuable when the user wants to assess the influence of specific graph properties in downstream tasks. The workflow of schema-driven generation is also simpler than that of instance-driven generation, which relies on analyzing and mimicking patterns present in reference KGs. Schema-driven generation avoids the complexity of handling an existing graph, which can be computationally expensive. Moreover, even for domains where KG construction is avoided, a schema-driven approach can still generate a KG, as the schema carries all the structural information required for the process. As the entities in the resulting KG do not correspond to those in a real-world KG, the generated graph is considered an anonymous synthetic graph. The risk of entity alignment is therefore reduced, making synthetic graphs useful for sensitive domains. However, schema-driven approaches also have drawbacks. While real-world KGs have an underlying schema, they still contain inconsistencies as the graph

¹⁴ While the approach is theoretically applicable to any ontology, Pygraft, as of this survey, is limited to generating KGs from the ontologies it creates.

■ **Table 4** General features of schema-driven approaches sorted descendingly by year. Δ marks evolving SKGGs, ● represents domain-agnostic approaches, and ○ represents domain-specific approaches. = denotes that blank nodes are generated in the same way as IRI nodes. Approaches that extend or are based on previous works are greyed out in the table. Four earlier approaches inspired subsequent work in the timeframe of this survey: LUBM [45], gMark [6], LDBC-SNB [33], SP2Bench [99], and WatDiv [3]. For example, EGG [2] and MonStaGen [116] are extensions of gMark. EvoGen [71] and DLUBM [59] are based on LUBM. OWL2Bench [103] is based on UOBM [70], and by extension LUBM. Spasić [104] and Waudby [122] are based on LDBC-SNB. RPQBench [118] is inspired by SP2Bench. Stream WatDiv [40] is an extension of WatDiv [3].

Approaches	Ont. Input	Ont. Gen.	Checks	Node Support		Domain Scope
				Blank	Literal	
Pygraft [53]	$\times^{14}$	✓	✓	$\times$	$\times$	●
Δ GenACT [102]	-	-	$\times$	$\times$	✓	○ Academic Tweets
Δ RPQBench [118]	-	-	$\times$	✓	✓	○ DBLP
Hnatkowska [48]	✓	$\times$	✓	$\times$	✓	●
ABRN [132]	✓	$\times$	✓	$\times$	$\times$	●
DLCC [90]	$\times$	✓	$\times$	$\times$	$\times$	●
GDD ^x [36]	$\times$	$\times$	$\times$	$\times$	✓	●
Geographica2 [55]	-	-	$\times$	$\times$	✓	○ Geospatial
Δ TWIG [105]	-	-	$\times$	$\times$	✓	○ Twitter
RODG [20]	✓	$\times$	✓	$\times$	✓	●
OWL2Bench [103]	-	-	$\times$	$\times$	✓	○ University
Δ Waudby [122]	-	-	$\times$	$\times$	✓	○ Social Networks
PoDiGG [109]	-	-	$\times$	$\times$	✓	○ Transportation
DLUBM [59]	-	-	$\times$	$\times$	✓	○ University
Δ EGG [2]	$\times$	$\times$	$\times$	$\times$	✓	●
gMark [6]	$\times$	$\times$	$\times$	$\times$	✓	●
Δ MonStaGen [116]	$\times$	$\times$	$\times$	$\times$	✓	●
Δ TrainBenchmark [108]	-	-	✓	$\times$	✓	○ Transportation
Δ Stream WatDiv [40]	-	-	$\times$	$\times$	✓	○ Products
JPoT [18]	✓	$\times$	$\times$	$\times$	✓	●
Δ EvoGen [71]	-	-	$\times$	$\times$	✓	○ University
GAIA [93]	✓	$\times$	$\times$	$\times$	✓	●
LDBC-SPB [60]	-	-	$\times$	$\times$	✓	○ Publishing
LinkGen [57]	✓	$\times$	$\times$	$\times$	✓	●
Spasić [104]	-	-	$\times$	$\times$	✓	○ Social Networks
Δ LDBC-SNB [33]	-	-	$\times$	$\times$	✓	○ Social Networks
BGen [61]	-	-	$\times$	✓	✓	○ Social Networks
WatDiv [3]	-	-	$\times$	$\times$	✓	○ Products

deviates [128]. It poses a challenge to the realism of the synthetic graphs produced by schema-driven graph generators, as the graphs generated by approaches in this category are typically clean, i.e., devoid of noise, unless deliberately added. LinkGen [57] addresses this limitation by intentionally introducing noise to the generated synthetic graph. Additionally, schema-driven generators face a more practical challenge, where the schema often does not conform to a standard format. Multiple generators expect ontologies with minimal configuration, while others expect extensive parameterization in configuration files.

5.1.3 Instance-driven Knowledge Graph Generation

An overview of the features of instance-driven approaches is presented in Table 5. Most instance-driven approaches are inherently domain-agnostic. This is largely due to the mimicking goal of the generation process. Regarding the node support, Fu [38], LEMMING [97], and RBench [91] support the generation of literal nodes. LEMMING generates a literal value by sampling it from a learned distribution for each datatype, while RBench leverages pre-generated vocabularies for each predicate. It is unclear how Fu et al. [38] generate literal nodes. All of the instance-driven approaches are also capable of generating blank nodes, but they achieve this by treating them without any distinction from IRI nodes. LEMMING is the only approach that incorporates an optimization process post-generation. None of the instance-driven approaches currently generates an evolving graph as output. Instance-driven approaches can be further distinguished by the size of the set of reference KGs $\mathbf{G}_{ref}$, i.e., whether the approach requires one reference graph or multiple. Notably, LEMMING is the only approach that expects more than one reference KG.

The core benefits of instance-based generation lie in its ability to mimic the structure of existing KGs. As instance-based approaches rely on a reference KG, these generators also model the inconsistencies commonly present in real-world data. Aligning the synthetic graphs with observed KGs enhances the realism of the synthetically generated graph. Consequently, schema- and instance-driven generation methods complement each other since they largely address each other's limitations. However, a disadvantage lies in the increased time and resources required to preprocess the real-world examples. As some large-scale KGs such as UniProt [25] currently span billions of triples, preprocessing full real-world examples takes careful design and requires an optimized preprocessing step. As information extracted from the graph instance already expresses structural information, instance-driven generators require fewer user-defined parameters. Schema-driven generators require extensive specification of all schema elements, i.e., classes, properties, and axioms, while instance-driven approaches learn these from reference graphs. Their reliance on observed patterns significantly reduces the burden on the user. Instance-driven generators enhance the realism of synthetic graphs by mimicking selected characteristics from the reference graph. For example, an instance-driven generator can mimic the degree distribution of a reference KG, while generating a new graph that is not isomorphic to the input KG, and may differ in global properties such as diameter. However, the improved realism comes at the cost of fine-grained control over the generated graph, as they are constrained to the learned patterns. Consequently, while instance-driven graph generators are ideal for generating graphs with realistic features, they are less suitable for evaluating the influence of specific deviations of graph properties in downstream tasks. The only surveyed approach to both require a schema and reference KGs is

■ **Table 5** General features of purely instance-driven approaches sorted descendingly by year. ● represents domain-agnostic approaches, and ○ represents domain-specific approaches. = denotes that blank nodes are generated in the same way as IRI nodes.

Approaches	Node Support		Optimizer	$ \mathbf{G}_{ref} $	Domain Scope
	Blank	Literal			
Fu [38]	=	✓	✗	1	●
TenGAN [101]	=	✗	✗	1	●
LEMMING [97]	=	✓	✓	> 1	●
SynthG [29]	=	✗	✗	1	●
Melo [73]	=	✗	✗	1	●
RBench [91]	=	✓	✗	1	●

LDBC-SPB [60]. It is a domain-specific approach where the synthetic graph inherits the realistic co-occurrence patterns from the reference KGs, while keeping to the semantics expressed by the schema. This separation allows the generator to be used with less manual modelling effort while still producing structurally and semantically realistic data.

5.2 Generation Methodology

This subsection addresses the advantages and disadvantages of the different generation methodologies: rule-based, probabilistic, probabilistic rule-augmented, tensor decomposition-based, and foundation model-based knowledge graph generation. The selected approaches are also described in this subsection.

5.2.1 Rule-based Knowledge Graph Generation

Rule-based KG generators systematically enforce logical rules to generate graphs. Rule-based systems are particularly effective in domain-specific scenarios where certain features of the data are known *a priori* to follow a specific pattern. A key advantage of using logical rules is their interpretability, as rules are often transparent and intuitive. However, purely rule-based approaches lack flexibility and can lead to synthetic graphs that do not reflect the uncertainty and complexity of real-world data.

5.2.1.1 GenACT

GenACT [102] generates a synthetic evolving KG representing the activity surrounding conferences via tweets, modelling the events from pre-conference announcements until post-conference reflections. The authors introduce the Academic Conference Tweets ontology available in four OWL 2 profiles: EL, QL, RL, and DL. It also encodes temporal constraints, such as a strict chronological event order. GenACT defines multiple templates for each event type present in the ontology. Each template acts as a structure for a specific event type, e.g., an accepted paper requires a manuscript, an author, and a conference. For each template, entities are instantiated. Each template is filled with synthetic data and deterministically mapped to RDF triples through mappings. The input parameters ψ include the number of generated conferences and the number of iterations of each conference. For example, two conferences with two iterations each could correspond to ISWC2024, ISWC2025, ESWC2024, and ESWC2025. GenACT's support for multiple OWL constructs ensures consistency by construction and captures realistic temporal constraints. However, since it relies on hard templates, GenACT assumes that participants will only engage with one event at a time. In real social networks, tweets are unpredictable and have mixed messages, often condensing the information of multiple conference events into a single post.

■ **Table 6** Types of rules observed in Rule-based Knowledge Graph Generation.

Approaches	Rule type	Mechanism
GenACT [102]	Fixed	Tweet Templates
Hnatkowska [48]	Logical	DL constraints with HermiT [42] reasoning
Geographica2 [55]	Spatial	Geospatial constraints

5.2.1.2 Hnatkowska

Hnatkowska and Kimmel [48] propose a KG generator that extends the PDGF [92] generator with ontology support. The input is an OWL 2 ontology and a YAML configuration file. The configuration file specifies the number of instances per class, a scale factor, instantiation probabilities, and a dictionary. The generator iterates over each class in the ontology and creates the desired number of instances. For each instance, data property assertions are generated by selecting the most appropriate generator from three types: a synthetic generator that produces values from the ontology constraints and datatype ranges, dictionary generators that sample values from external sources, and hybrid generators that combine both. As the dictionary generator takes precedence over the synthetic one, the selection works by computing a semantic similarity score over the WordNet taxonomy, matching dictionary keywords against property and class names. Object property assertions are generated according to cardinality constraints, instantiation probability, and property features such as functionality or transitivity. Finally, Hermit [42] is applied to infer additional axioms.

5.2.1.3 Geographica2

Geographica2 [55] is a benchmark for RDF stores focused on evaluating their geospatial support, specifically their support for geospatial query languages such as GeoSPARQL and stSPARQL [10].¹⁵ It is a subsequent work of Geographica [41] in which the benchmarking effort is extended by increasing the number of tested triplestores and the evaluation effort. However, the details of the data generator remain the same from its predecessor. The synthetic data generator is bound to a geospatial ontology based on OpenStreetMap and GeoSPARQL ontologies.¹⁶ The graph generation process is therefore largely governed by geometrical constraints. The generator defines four entity types: land ownership, states in a country, roads, and points of interest. Land ownership defines the physical space for the subsequent features, organized as a hexagonal grid of n hexagons. States are represented by $\frac{n}{3}^2$ hexagons placed on a $\frac{n}{3} \times \frac{n}{3}$ grid. Roads are created by generating n straight lines. The points of interest are evenly placed on each road, with n^2 points of interest per road. For example, a synthetic workload of $n = 12$ would produce 144 instances of land ownership, 16 states, 12 roads, and 144 points of interest.

5.2.2 Probabilistic Knowledge Graph Generation

Probabilistic knowledge graph generators model graph properties using probability distributions. Several studies demonstrate that real-world KG metrics, such as the vertex degree distribution [21], often follow a specific distribution [30]. Consequently, many probabilistic graph generators model the degree distribution of the output graph using a fixed distribution, with a power-law distribution being a common choice. The most commonly modelled graph features were the degree of entity nodes [97, 57, 84] and the degree of subject and object entity nodes separately [1]. Other approaches model class distributions [97, 73]. Beyond vertex degrees, generating literal values using probability distributions is also a common practice [57, 105]. TWIG [105] also models the daytime distribution to generate realistic user activity. While probabilistic schema-driven approaches rely on established distribution types, probabilistic instance-driven approaches use non-standard distributions as they learn the observed distribution from the reference graph [97]. The summary of the probabilistic graph generators is presented in Table 7.

¹⁵ <https://www.ogc.org/standards/geosparql/>

¹⁶ <http://www.openstreetmap.org/>

■ **Table 7** Probabilistic Graph Generation. The postscripts *out* and *in* indicate that the condition only applies to out- and in-degrees, respectively. Separate marks approaches that model different characteristics of the same feature separately or not. For example, some approaches may model the average node degree distribution, while others model the in-degree and out-degree separately. NS stands for a non-standard distribution.

Approaches	Feature	Separate	Distribution type				
			Exp.	Power-law	Uniform	Gaussian	NS
RPQBench [118]	Path Length	✗	✗	✓	✗	✗	✗
KRELM [1]	Degree	✓	✓ <i>out</i>	✓ <i>in</i>	✗	✗	✗
RD ² [126]	Nodes, Edges	✗	✗	✗	✓	✗	✗
LEMMING [97]	Classes, Edges, Degree	✗	✗	✗	✓	✗	✓
ORCA [96]	Degree	✗	✗	✓	✗	✗	✗
TWIG [105]	Out-degree, Temporal	✗	✗	✗	✗	✗	✓
TrillionG [84]	Degree	✓	✗	✓	✓	✓	✗
LinkGen [57]	Degree	✗	✗	✓	✗	✓	✗

Probabilistic graph generation offers several advantages. They provide diversity by introducing stochasticity, enabling the generation of graphs that reflect real-world characteristics. Probabilistic instance-driven generators also present greater flexibility as they can model either observed (non-standard) or defined distributions. However, modelling with a fixed distribution also has disadvantages. While power-law distributions often occur in real-world KGs, they are not universally applicable [31]. This is the case for any distribution assumption. Furthermore, modelling the degree distributions, either jointly or separately, disregards predicate semantics. For example, modelling the in-degree and out-degree jointly does not account for inverse or symmetric properties, and modelling them separately still fails to account for property symmetry.

5.2.2.1 RPQBench

RPQBench [118] extends the data generator of SP2Bench [99] to ensure path lengths in synthetic KGs follow power-law distributions. RPQBench is the first KG generator to model a distribution over path lengths. The input parameters are the power-law exponent, the minimum and maximum path lengths, a year range, and a frequency of data generation over the years. The generator iterates over each year and generates path lengths following the desired distribution. For each path, the required instances are generated. The underlying triple generation is inherited from SP2Bench, which populates instances with properties drawn from distributions fitted to the original DBLP dataset.

5.2.2.2 KRELM

KRELM [1] generates KGs using two different attachment mechanisms depending on whether an existing subject or object is being reused, simulating the growth of crowdsourced KGs over time. For every new triple, the model creates or reuses a subject and an object according to subject and object growth probabilities. When an existing entity is reused, subjects are selected uniformly, whereas objects are selected using preferential attachment, with selection probability proportional to their current degree. This results in an exponential degree distribution for subjects and a power-law distribution for objects. KRELM operates on one relation at a time, so for each predicate, it requires as input parameters the number of facts n , the number of subjects n_s , and

the number of objects n_o . KRELM generates a graph by representing each relation p as a bipartite graph $(\mathbb{S}, \mathbb{O}, \mathbb{F})$ with the set of subject vertices $\mathbb{S} \subseteq \mathbb{I}$, the set of object vertices $\mathbb{O} \subseteq \mathbb{I}$, and the set of edges $\mathbb{F}_p = \{(s_i, o_j) : (s_i, p, o_j) \in \mathcal{G}\}$. For each predicate, the growth probability of new entities is set to $\frac{n_s}{n}$ for subjects and $\frac{n_o}{n}$ for objects. For each fact, the generator decides for both endpoints independently whether to create a new entity or reuse an existing one. If no new entity is created, a subject is selected at random with uniform probability, and an object is selected proportionally to the number of triples connected to the object. For example, considering the predicate `:nationality`, it is more likely that highly populated countries will gain another citizen. KRELM does not enforce cardinality constraints. However, since the growth probabilities for each predicate control how often a new entity is introduced versus reused, the user can approximate functional behaviour on either side by tuning n_s and n_o relative to n .

5.2.2.3 RD²

RD² [126] generates a KG at random. The only user-defined parameter affecting KG generation is the total number of desired KGs. The remaining parameters are hard-coded. Actual graph size is determined by sampling from these ranges, e.g., the number of IRI nodes, the number of triples per instance, and the proportion of blank nodes. A set of IRIs is generated first using the Faker library.¹⁷ The resulting KG is then created by iterating over each IRI as a subject, sampling a predicate and an object, where the object can be an IRI, a literal, or a blank node. Literal values are generated randomly according to a selected datatype. Since RD² is an approach that generates all elements at random, it fails to capture valid or consistent knowledge.

5.2.2.4 LEMMING

LEMMING [97] is an instance-based KG generator that analyzes the characteristics of a versioned KG and replicates these in a synthetic graph. The input parameter is the desired number of vertices in the output KG. It features three main stages: graph analysis, graph generation, and graph optimization. In the analysis stage, LEMMING computes the values of a configurable set of metrics and the average degree of each input graph $\mathcal{G}_i$ to determine the expected degree of the synthetic graph. Then it computes the type probability $P_{\mathcal{G}_i}(T)$, the probability of each predicate $P_{\mathcal{G}_i}(p)$, and the probability that the tail and head vertices are instances of the class sets T_t and T_h , $P_{\mathcal{G}_i}((T_t, T_h)|p)$. The out-degree of datatype properties is also stored. The results are averaged over all input graphs. During graph generation, the set of possible classes is determined for each edge, then the class sets for the tail and head entity are sampled from these sets, and then the instances are chosen. LEMMING allows for three different approaches to select the class sets: randomly drawing from a uniform distribution, sampling based on the sum of the conditional probabilities $P_{\mathcal{G}_i}((T_t, T_h)|p)$, and directly sampling both the tail and head entity simultaneously from $P_{\mathcal{G}_i}((T_t, T_h)|p)$. There are also two approaches to selecting instances: randomly drawing from a uniform distribution, and sampling from a distribution where each vertex is assigned a degree weight. Simultaneous to the analysis stage, LEMMING learns invariant arithmetic expressions of the configurable graph properties over the input graphs against graphs produced by a generator based on Barabási–Albert [7]. These expressions are used to optimize the synthetic graph. The optimization aims to minimize the error score between the values of the invariant expressions learned earlier for the input graphs and the synthetic graph. This optimization is achieved iteratively by adding and removing an edge, guided by the decreasing error. Finally, literal nodes are generated, and the `rdf:type` triples are restored in the graph. While the optimization phase can improve the structural fidelity of the output graphs, this step comes at a runtime cost.

¹⁷<https://github.com/joke2k/faker>

5.2.2.5 ORCA

ORCA [96] introduces a KG generator based on the Barabási-Albert [7] model. ORCA generates a scale-free network of RDF KGs that ensures that the generated dataset is crawlable, i.e., all resources are reachable from a seed resource. The input parameters are the average number of triples per KG and the average degree. Initially, it samples the average node degree from a uniform distribution, generates the connections via preferential attachment (the Barabási-Albert model), and uses a Bernoulli distribution to determine the direction of each triple. ORCA is a lightweight KG generator. However, it lacks semantic depth, as it is based on a plain graph generator. Furthermore, assigning edge directions via Bernoulli sampling assumes independent binary outcomes and therefore does not capture the semantically constrained directionality typically observed in KGs.

5.2.2.6 TWIG

TWIG [105] is a KG generator that mimics the structure of a Twitter social network, modelled with its own custom TWIG ontology. TWIG learns statistical patterns from a real Twitter crawl and then generates networks to match these patterns. It models three main distributions: the number of tweets per user, the daytime distribution of tweet posting times, and a first-order Markov chain over word predecessor-successor pairs. The number of tweets per user effectively models the out-degree distribution. Given a target user count, start date, and the mimicking duration in days, TWIG first generates the target number of users. For each user, it samples the tweet count and sending times from the estimated distributions, and produces each tweet's contents with the word predecessor-successor distribution. TWIG generates high-quality synthetic literal values due to the Markov chain walk and realistically models the quantity and the timing of posts. However, its schema is small and does not reflect most of the interaction types present in social networks.

5.2.2.7 TrillionG

TrillionG [84] is a synthetic graph generator that uses a stochastic approach inspired by the RMat [19] and Kronecker [65, 64] models. It employs a recursive vector model that enables distributed computing by computing the cumulative distribution values at specific intervals to determine the graph edges recursively. The input parameters are the scale factor that influences the number of nodes, a seed matrix that controls the graph structure, and the average degree of the graph. Despite the classification of TrillionG as a basic SKGG, TrillionG has an optional schema input parameter where the semantics can be injected into the probability matrix. TrillionG accomplishes this task by segmenting the probability matrix by node class using the vertex ranges. This partitioning allows specific in- and out-degree distributions for each segment. This extension is introduced as the extended recursive model. TrillionG is a very efficient generator. However, the type of constraints it can model is limited.

5.2.2.8 LinkGen

LinkGen [57] is a KG generator based on OWL or RDFS ontologies that replicates a desired degree distribution and perturbs the synthetic graph with noise. It supports two vertex degree distributions: gaussian and power-law. LinkGen first loads an ontology and extracts classes, properties, and statistics from the given ontology. It then generates the set of resources and assigns weights based on the desired distribution. Triples are then generated by sampling the weighted nodes and connecting them via properties associated with the node's class. Literal values are

■ **Table 8** Types of rules observed in Probabilistic Rule-Augmented Graph Generation.

Approaches	Rule type	Mechanism
Pygraft [53]	Logical	DL constraints with Hermit [42] reasoning
ABRN [132]	Logical	OWL and SWRL axioms
DLCC [90]	Logical	DL constructors
GDD ^x [36]	Structural	Extended Graph Differential Dependencies (GDD)
RODG [20]	Logical	DL reasoning
Waudby [122]	Fixed	Cardinality
OWL2Bench [103]	Logical	OWL 2 profile (EL, QL, RL, DL)
Stream WatDiv [40]	Dependency	Co-occurrences
PoDiGG [109]	Spatial	Geospatial constraints
TrainBenchmark [108]	Structural	Graph patterns
JPoT [18]	Logical	Via $\mathcal{ALCQ}^{(D)}$ axioms
DLUBM [59]	Fixed	Cardinality ranges
Melo [73]	Logical	Horn rules mined with AMIE [39]
EGG [2]	Structural	Constraint Vectors
MonStaGen [116]	Structural	Constraint Vectors
gMark [6]	Structural	Constraint Vectors
Spasić [104]	Dependency	Co-occurrences
GAIA [93]	Logical	OWL constraints
EvoGen [71]	Fixed	Cardinality ranges
LDBC-SPB [60]	Dependency	Co-occurrences
RBench [91]	Structural	Graph patterns
LDBC-SNB [33]	Dependency	Co-occurrences
WatDiv [3]	Dependency	Co-occurrences
BGen [61]	Structural	Morphology constraints

randomly generated according to their datatype. Since real-world graphs are frequently unclean, LinkGen aims to replicate this feature by perturbing the graph and adding inconsistent data to the final graph. This noise can be in the form of syntactic errors, invalid triples, or triples inconsistent with the given ontology. Furthermore, LinkGen supports linking with real-world entities through `owl:sameAs` links. Despite LinkGen accepting OWL as an input, it only supports RDFS.

5.2.3 Probabilistic Rule-Augmented Graph Generation

Probabilistic rule-augmented generators combine the strengths of rule-based generation with probabilistic modelling to synthesize graphs. Hybrid approaches recognize that graphs inherently exhibit patterns shaped by both logic and probability distributions and leverage this to achieve higher generalization and realism. These generators typically define logical rules that often encode domain knowledge. Simultaneously, they model probability distributions of graph properties. During graph generation, the rules ensure consistency, while the probabilistic part adds flexibility to produce more diverse graphs. The summary of the probabilistic rule-augmented graph generators is presented in Table 8 and 9. The type of rules varies from approach to approach. The approaches in this category mainly use DL [53, 90], Horn rules [73], graph patterns [36, 91], structural rules [61], spatial rules [109], or fixed constraints [59, 71]. The latter is more common in domain-specific approaches, as they remain fixed for all generated graphs and, in this case, stem from domain knowledge. Regarding the probability distributions, they follow a similar pattern to the probabilistic graph generators. Several approaches model the distribution of

the vertex degree [53, 90, 73, 91], the vertex degree conditioned on the class [36, 91], or the relations [53, 73]. Several approaches apply the rules and the probabilistic modelling to separate parts of the generation workflow, e.g., generate the graph in a probabilistic manner and then apply rules to validate the graph or triple [53, 73], or generate the graph based on the rules and apply probability distributions to decide on the outcome of a rule [90, 61]. However, some approaches combine the two more tightly by using rules or the result of rules to parameterize a probability distribution [59, 71], or by modelling a distribution on the application of a rule [91].

Probabilistic rule-augmented generators benefit from the advantages of both rule-based and probabilistic approaches. They inherit the interpretability and specificity of rules, while addressing the inflexibility of purely rule-based systems. By incorporating probability distributions, these generators introduce variance and diversity in an otherwise deterministic setting. Hybrid generators aim to generate graphs that are logically sound but representative. However, hybrid generators only mitigate the limitations of the single categories. As the combinations of rules and probabilities vary significantly with each approach, the degree of improvement also varies. That is, the extent of improvement depends heavily on how the rules and the distributions are combined. If probability distributions are used superficially, the benefits will also be limited.

■ **Table 9** Modelled features in Probabilistic Rule-Augmented Graph Generation. Separate marks approaches that model different characteristics of the same feature separately or not. For example, some approaches may model the average node degree distribution, while others model the in-degree and out-degree separately.

Approaches	Feature	Separate	Distribution type				
			Exp.	Power-law	Unif.	Gaussian	NS
Pygraft [53]	Nodes, Edges	✗	✗	✗	✓	✗	✓
ABRN [132]	Edges	✗	✗	✗	✓	✗	✗
DLCC [90]	Nodes	✗	✗	✗	✓	✗	✗
GDD ^x [36]	Degree	✓	✗	✗	✓	✓	✗
RODG [20]	Edges	✗	✗	✗	✗	✗	✓
Waudby [122]	Lifespans	✗	✗	✗	✓	✗	✗
OWL2Bench [103]	Cardinality	✗	✗	✗	✓	✗	✗
Stream WatDiv [40]	In-degree	✗	✗	✓	✓	✓	✗
PoDiGG [109]	Nodes	✗	✗	✗	✗	✗	✓
TrainBenchmark [108]	Cardinalities	✗	✗	✗	✓	✗	✗
JPoT [18]	Nodes, Classes	✗	✗	✗	✓	✗	✗
DLUBM [59]	Edges	✗	✗	✗	✓	✗	✗
Melo [73]	Nodes, Edges	✗	✓	✗	✓	✗	✓
EGG [2]	Nodes, Temporal	✓	✗	✓	✓	✓	✗
MonStaGen [116]	Nodes	✓	✗	✓	✓	✓	✗
gMark [6]	Nodes	✓	✗	✓	✓	✓	✗
Spasić [104]	Nodes	✗	✗	✓	✗	✗	✗
EvoGen [71]	Edges	✗	✗	✗	✓	✗	✗
LDBC-SPB [60]	Nodes	✗	✗	✓	✗	✗	✗
RBench [91]	Nodes	✗	✗	✗	✗	✗	✓
GAIA [93]	Nodes	✗	✗	✗	✓	✗	✗
LDBC-SNB [33]	Nodes	✗	✗	✓	✗	✗	✗
WatDiv [3]	In-degree	✗	✗	✓	✓	✓	✗
BGen [61]	Nodes	✗	✗	✓	✓	✗	✓

5.2.3.1 Pygraft

Pygraft [53] is a data generator that synthesizes both an ontology and a KG based on a JSON or YAML configuration file. The parameter list for the complete workflow is extensive. The input parameters of the KG generation stage include the number of entities, the number of triples, the relation distribution across triples, the proportion of untyped entities, the average depth of the most specific class of all entities, whether entities have multiple class memberships, and the average number of classes that typed entities have. The ontology is first built from user-specified parameters governing classes, relations, and a broad set of OWL and RDFS axioms. The KG is then generated on top of this ontology. Entities are first assigned a class from the class hierarchy, with configurable proportions of typed and untyped entities. Multiple classes can also be assigned to the same entity, while obeying class disjointness constraints. Triples are then generated sequentially up to a user-specified count. The relation is sampled according to this distribution, while the subject and the object are sampled according to the relation's domain and range, biased towards unobserved entities. A final consistency check is performed by HermiT [42], a DL reasoner.

5.2.3.2 ABRN

ABRN [132] is a KG generator designed to produce training data for a neural-symbolic ontology reasoner called TGR. The input parameters ψ include an OWL and Semantic Web Rule Language (SWRL) ontology and the number of instances per class that need to be generated. The generator first creates the class instances to meet the specified counts and applies the HermiT [42] reasoner to infer all `rdf:type` triples via the class hierarchy. For each property in the ontology, assertions are created by randomly selecting an instance from the property's domain and another from its range. HermiT is then used for consistency checking and to infer new triples, while inconsistent triples are identified and kept as negatives.

5.2.3.3 DLCC

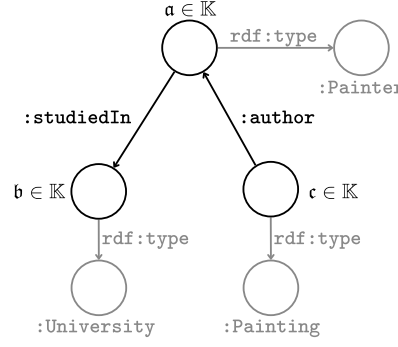
DLCC [90] introduces a KG generator based on DL constructors. It defines twelve DL expressions, and for each expression, a set of positive and negative examples is generated. The input parameters regarding the KG generation are the number of instances, the number of properties, the maximum number of triples per instance, and the number of positive examples per DL constructor. The data generator first creates the ontology before the KG. DLCC generates the entities and associates them with a type from the class hierarchy. Based on the set of entities previously created, the desired number of positive and negative examples of each expression is generated. For each entity in the positive and negative examples, a random number of triples with the entity as subject is generated, ensuring that the domain and range information are still adhered to. Finally, it checks if no additional positive examples are created.

5.2.3.4 GDD^x

GDD^x [36] is a KG generator based on Extended GDD, a concept introduced in the same work. We omit the formal definition of Extended GDD in favour of the practical example below.

Figure 5 illustrates an example graph pattern similar to [36]. For example, the Extended GDD σ_1 in Equation 11 dictates that at least 50% of the matches of the graph pattern Q_1 must have a `:studiedIn` relation between a painter `a` and a university `b`:

$$\sigma_1 = (Q_1[\varsigma, M], \emptyset \rightarrow \{\delta_{\equiv}(\mathbf{a}.\text{:studiedIn}, \mathbf{b}) = 0\}, 0.5). \quad (11)$$



■ **Figure 5** Graph pattern $Q_1[a, b, c]$.

The Extended GDD σ_2 in Equation 12 dictates that all matches of the graph pattern Q_1 must have an `:author` relation between a painting c and a painter a :

$$\sigma_2 = (Q_1[\varsigma, M], \emptyset \rightarrow \{\delta_{\equiv}(c : \text{author}, a) = 0\}, 1). \quad (12)$$

The graph generator GDD^x requires a set of Extended GDD as input to its generation process. First, it initializes the set of entities and assigns types to each entity based on user-defined class proportions and the input schema. Afterwards, the relationships are created by replicating the desired graph patterns, obeying the constraints imposed by the Extended GDD. The probability distribution of entities at a class-level and of the literal values can also be specified.

5.2.3.5 RODG

RODG [20] generates a KG as a collection of object descriptions, where each description is a KG centered on an entity of a user-specified class. The object descriptions are not connected with one another. Given an input OWL 2 ontology and a chosen root class, RODG first parses the ontology with OWL API [51] and applies a reasoner to derive the implicit triples. The generator starts by traversing the class hierarchy depth-first, recursively instantiating individuals. For each individual, it processes the axioms in a set order: equivalent classes, anonymous classes, and domain and range restrictions. Configurable probability thresholds govern whether each assertion is included in the description, introducing variability into the output.

5.2.3.6 WatDiv Extensions

WatDiv and its extension differ only in the type of KGs they produce.

WatDiv [3] introduces a KG generator based on the custom WatDiv schema representing products, e.g., books, movies, concerts. The parameters are a scale factor and a configuration file. The scale factor is applied to selected classes to calculate a scalable number of instances. The configuration file has the classes, the data properties associated with an instantiation probability, object properties each associated with the domain, range, subject/object cardinality, instantiation probability, and value distribution. The generator creates the KG by iterating over each entity class, instantiating the required number of instances per class. For each instance, data property assertions are generated based on the instantiation probability. Data properties can be grouped at the entity type level such that either all or none of the properties in a group are instantiated together for a given instance. The literal values are drawn from configurable ranges and distributions depending on the datatype. Finally, object property assertions are generated, objects are sampled from the specified class using the configured distribution (uniform, gaussian, or zipfian), and the cardinality range.

Stream WatDiv [40] extends WatDiv to RDF streams by partitioning the WatDiv schema into a static part (e.g., users, products, retailers) and a streaming part capturing user activities (purchases, reviews, offers, likes, follows, subscribes). The generator takes two scale factors, one for the static and one for the streaming dataset, instead of a single one. Data generation proceeds in two steps. First, static and stream triples are instantiated following the WatDiv procedure over the shared schema. Second, the stream is annotated with monotonically increasing millisecond timestamps according to a configurable rate, with triples with the same entity instance sharing a timestamp.

5.2.3.7 PoDiGG

PoDiGG [109] is a synthetic KG generator for the public transport domain. The dataset models regions, stops, edges, routes, and trips of public transport. Regions correspond to 2-dimensional matrices, where each cell has a population density. This can be based on real data or sampled from a distribution. The stops are placed based on a power-law distribution weighted by the population. The edges are randomly created based on the Euclidean distance, and are iteratively edited based on points with higher density in the vicinity.

5.2.3.8 TrainBenchmark

TrainBenchmark [108] is a generator that produces a synthetic railway KG with a fixed schema, covering elements like semaphores, routes, regions, switches, sensors, and segments. Counts and cardinalities are sampled at random from configurable ranges. The input parameters are a scale factor and a scenario (Batch, Inject, or Repair). After generating the KG, TrainBenchmark has three different use case scenarios, where noise is added to the KG at different rates. The batch validation scenario produces a clean KG with no faults. The fault injection scenario simulates user edits by injecting faults at low rates. While adding noise, the generator deliberately introduces known violations, e.g., removing triples, setting invalid literal node values. The automated model repair scenario injects faults at larger rates to simulate models that need repairing. Consistency checks are performed by matching inconsistencies described by hand-crafted graph patterns.

5.2.3.9 JPoT

JPoT [18] generates a KG for an input OWL2 ontology expressed in $\mathcal{ALCQ}^{(D)}$. The input parameters ψ are the number of individuals, the number of triples, the fraction of `rdf:type` triples relative to the total number of triples, and the fraction of role triples relative to the number of data triples. JPoT generates three types of triples in three stages: class assertions, object property assertions, and data property assertions. Class assertions use the `rdf:type` predicate, object property assertions link two resources, and data property assertions connect a resource with a literal node. The generation has three stages. First, entities are instantiated and assigned to classes with enforced disjointness. Second, object property assertions are generated by drawing a property and selecting subject and object entities such that the domain, range, universal restrictions, and cardinality constraints are respected. Third, the data property assertions are generated according to a datatype.

5.2.3.10 Melo

Melo et al. [73] propose an instance-driven knowledge graph generator. It collects several statistics from an input graph, such as the relation probability $P(p)$, the subject type set probability given a relation $P(T_s|p)$, and the object type probability given a relation and a subject type set $P(T_o|p, T_s)$.

Based on them, the joint probability $P(p, T_s, T_o)$ can be computed. The instance probabilities of an entity b , i.e., $P(b|p, T)$, are computed by fitting a uniform or an exponential distribution model on the occurrences of individuals given a relation and a type set. As probabilities cannot model path constraints, horn rules are mined from the KG to model transitive, symmetric, equivalent, and inverse relations.

The generator creates a triple sequentially, where a relation, a subject type set, and an object type set are first sampled from $P(p, T_s, T_o)$. Afterwards, the subject and object entities are selected based on the corresponding $P(b|p, T)$. Then, it verifies the triple against functionality and reflexivity constraints. If the triple is valid, the mined rules are matched recursively against the generated triple, and more triples are generated for the same rule with a probability equal to that of the rule's confidence. Finally, the joint probability is updated to account for the rule-generated triples.

5.2.3.11 GAIA

GAIA [93] loads an ontology from a file and generates a KG accordingly. Despite requiring an OWL ontology, it does not replicate all OWL constructs. GAIA first extracts the classes and properties from the ontology, and generates the KG. A random number of IRI nodes is generated for each class. Based on the predicate, it can either generate a literal value at random if the predicate is associated with a datatype, or a resource object otherwise. In the latter, it sets a 40% likelihood of nodes of a class being linked to nodes of another class.

5.2.3.12 gMark Extensions

Generators based on gMark target schema-driven KG generation with fine-grained control over degree distributions, and have been extended toward graph sequences and temporal KGs.

gMark [6] is a schema-driven KG generator. Users must specify the graph size as the number of nodes the target graph should have, the node types, the predicates, the occurrence constraints, and the expected node degree distributions. It supports different in- and out-degree distributions, e.g., uniform, gaussian, and power-law. An occurrence constraint $\nu(T_1, T_2, a)$ restricts how often a predicate a may connect nodes of source type T_1 to nodes of target type T_2 , thereby controlling the number of edges of that type in the generated graph. For each constraint, gMark creates a vector v_{src} for the source nodes and v_{trg} for the target nodes. v_{src} is populated with nodes of type T_1 drawn from the out-degree distribution. Similarly, v_{trg} is filled with nodes of type T_2 drawn from the in-degree distribution. The vectors are then shuffled, and triples are created by iterating simultaneously through both vectors and connecting the nodes at matching positions.

MonStaGen [116] is an extension of gMark that generates a sequence of graphs of increasing size. This is achieved by considering the monotonicity of edges, i.e., the edges present in the previous version will be present in the next version, and distribution stability, i.e., the node degree ranking remains stable across the sequence such that nodes with a high degree in one version retain a comparatively high degree in later versions. MonStaGen works on the bipartite graph representation and determines the number of possible connections for each edge type based on the expected degree.

EGG [2] is an extension of gMark that generates evolving KGs. It generates snapshots of a graph with temporal annotations that satisfy a set of evolving constraints defined by the user. The evolving constraints can indicate absolute ranges, distributions, change probability, and the magnitude of change.

5.2.3.13 LDBC Generators

The Linked Data Benchmark Council (LDBC) has produced a family of KG generators that have inspired several follow-up approaches. The original generators target two different domains: LDBC-SNB [33] simulates a social network, while LDBC-SPB [60] models semantic publishing. Subsequent works have extended LDBC-SNB to address shortcomings in the structuredness of the generated graphs [104] or to support dynamic graphs with temporal entity deletions [122].

The **LDBC-SNB** [33] data generator generates a synthetic KG by simulating a user's activity in a social network. The dataset consists of persons, tags, forums, messages, likes, organizations, and places. Entities are generated with random attributes, and relationships are established with a certain probability. The generated graph has highly correlated features, where, for example, a person's interests are correlated with the topic of a post that the same person shares in a forum. The literal strings in posts and comments are generated using text from DBpedia pages associated with the post's topic. A user's activity over time is connected with synthetic events where the volume of posts increases.

Spasić et al. [104] noticed that the graphs generated by the LDBC-SNB generator achieve high coherence values, while many real-world KGs exhibit low coherence [31]. Consequently, they propose an alteration to lower the coherence values by reducing the coverage of each resource type in the dataset. The modifications include:

- Removing post locations,
- Replacing 80% of the short comments with GIF images,
- Introducing a property to mention other users in 40% of the posts,
- Introducing a visibility property to 5% of the messages, and
- Introducing a link property to 43% of the posts and comments.

The exact percentage values were selected from existing research on these relationships on real-world social networks.

Waudby et al. [122] introduce an extension of the LDBC-SNB algorithm to support dynamic KGs with temporal entity deletions. It is based on the same social network schema, but it creates dynamic entities and edges, where it introduces a lifespan (creation and deletion date). The lifespan inheritance is dictated by domain-specific rules, e.g., walls share the same lifespan as the wall's owner. The input parameters include a network collapse date, a simulation start and end, and a minimum interval between consecutive operations on an entity. The generator selects dates from the lifespan of entities. For each node, the creation and deletion dates are drawn from the simulation period. For each edge, dates are bounded by the lifespan of the connected vertices.

The **LDBC-SPB** [60] data generator generates synthetic graphs based on ontologies and reference graphs. The reference graphs used are snapshots of real-world BBC, GeoNames, and DBpedia graphs. The synthetic data generator creates creative works based on entities from the reference datasets. The generation process models three characteristics:

- (i) a clustering effect by generating creative works about an entity decaying over time,
- (ii) a correlation between entities by generating creative works about two or three entities, and
- (iii) a random tagging of entities with a bias towards popular entities.

The authors randomly select 5% of all entities to form the set of popular entities. The popular entities are connected to 30% of the creative works, while the remaining 70% are connected to normal entities.

5.2.3.14 LUBM Generators

LUBM [45] is a widely used KG generator for the university domain that has inspired several extensions in the timeframe of this survey. These extensions adapt LUBM to address scenarios that the original generator does not cover, such as distributed environments, schema evolution over time, and improved OWL axiom coverage. We describe three such extensions below.

DLUBM [59] is an extension of LUBM [45] designed for distributed environments. DLUBM generates interconnected graphs across multiple hosts. Like its predecessor, it is built for the university domain, with features such as departments, professors, students, courses, and their relationships. It introduces three data granularity levels: global, university, and department. As an extension of LUBM, DLUBM also combines predefined rules with random sampling. The rules are defined by domain knowledge, such as determining the cardinality range of each class, e.g., each university must have between 15 and 25 departments. However, the exact number is randomly decided.

EvoGen [71] is a LUBM-based generator that accounts for schema evolution with the assumption that changes are evenly distributed across all versions within the time interval. It introduces 10 new classes to the LUBM ontology that better mimic temporal entities, such as visiting professors and research projects. EvoGen annotates the dataset with temporal information by introducing four parameters: dataset shift, dataset monotonicity, schema variation, and ontology evolution. Dataset shift determines if the dataset size increases or decreases in size. The monotonicity determines whether the change between versions is monotonic. The schema variation captures the schema variation that a dataset suffers through time. This is determined by quantifying the different characteristic sets of the dataset at different points in time. The ontology evolution is the ratio of classes between ontologies at different timestamps. Based on these parameters, EvoGen calculates the number of additions and deletions needed to satisfy the specified temporal constraints.

OWL2Bench [103] presents a KG generator that extends the University Ontology Benchmark (UOBM) [70] generator, and by extension LUBM, to cover OWL 2 axioms. It provides four different ontologies, one for each supported OWL 2 profile (EL, QL, RL, DL), and a KG generator that produces instance data depending on each profile. Starting from the desired number of universities, OWL2Bench generates instances of related classes recursively, e.g., colleges, departments, programs, students, and others. Then, object and data property assertions are created with random counts from user-defined ranges. Literal values for universities and persons are drawn from pre-arranged lists of names, while other instances receive counter-based naming. The main input parameters are the number of universities and the target OWL 2 profile. Other parameters, such as the counts of departments per college or the students per program, can be set in a configuration file.

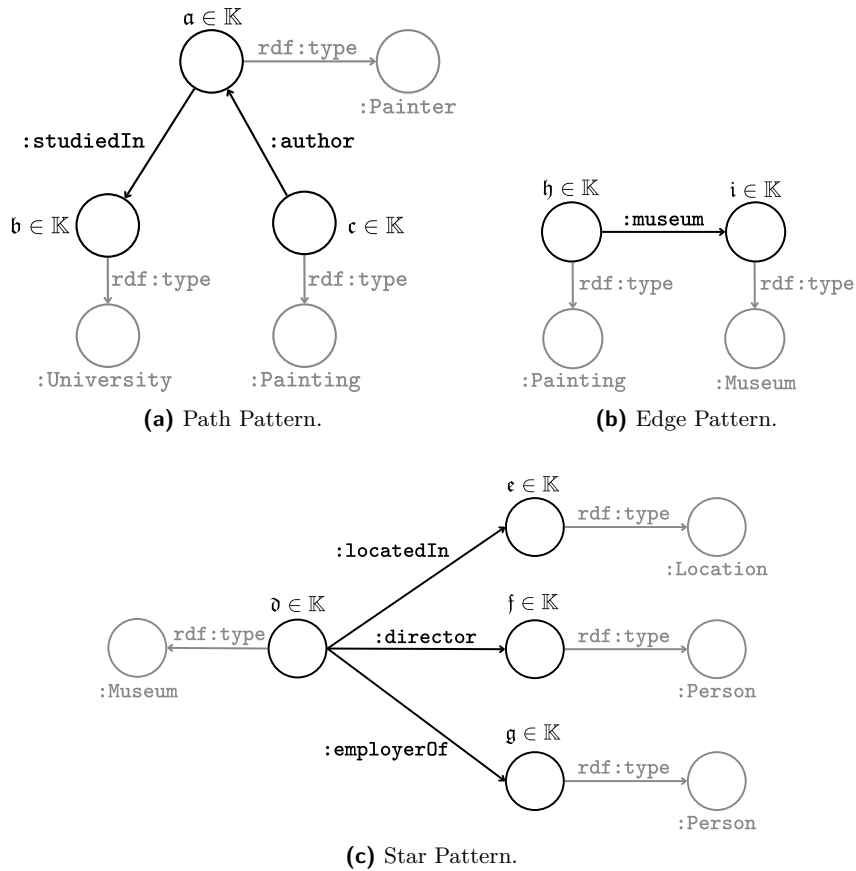
5.2.3.15 RBench

RBench [91] is an instance-driven generator that analyses the characteristics of an input KG and generates a synthetic version given a size scaling factor u and a degree scaling factor ζ . It starts by extracting a type system T from the input KG by finding triples with the predicate `rdf:type`. If less than 80% of the resources are typed, RBench complements the type system. As resources of the same type usually share a similar set of predicates, the authors match each predicate with each unknown-typed resource. The new types are chosen based on the coherence metric. Afterwards, the graph structure is analyzed. For triples involving two resources, resource identity and relationship patterns are collected. Resource identity is determined by the resource IRI, the

associated type, and the number of predicates associated with this resource. Relationship patterns are graph patterns connecting the predicates and resource types. The patterns considered in this approach were path, star, and edge, and are depicted in Figure 6. These are generated sequentially, and the number of occurrences in the graph is stored. For triples involving a literal, predicate dictionary, and attribute star patterns are analyzed. The predicate dictionary records the literal value and the occurrence count for each predicate associated with a literal node. The dictionary is only maintained for string literals. Similarly to the relationship patterns, the attribute pattern also connects the predicates and the resource types. For the literal node, this is only marked as literal. The graph generation process starts by generating the resources given the ratio $\frac{\mu}{\zeta}$. It assigns types to the new resources according to the type distribution from the resource identity previously computed. Literals are generated by sampling from a precomputed list of candidate words for each literal-associated predicate. The resources and literals are then matched against the patterns based on the type and generated degree. The triples are generated simultaneously with the pattern matching.

5.2.3.16 BGen

BGen [61] is a data generator for social networks focused on the different connectivity patterns of blank nodes. The schema features social networks with people, their activity, and messages. People are always represented as blank nodes, but the number of blank nodes connected to each



■ **Figure 6** Types of patterns in RBench.

person is a configurable parameter. However, certain classes are defined to always be instantiated as IRI resources. The model takes a graph morphology, clustering coefficient, diameter, density, and similarity parameters to control how different graph components interact with each other. A component is a subgraph containing only person instances, reflecting the network of connections between them. It also requires a probability distribution to control the distribution of components across graph morphologies and to regulate the distribution of people within these components.

5.2.4 Tensor Decomposition-based Graph Generation

Tensor decomposition-based graph generators leverage the fact that KGs can be represented as three-way tensors (**entity**, **relation**, **entity**). These generators use tensor decomposition techniques to obtain factorized representations, from which synthetic tensors are reconstructed. Tensor-based approaches capture complex interactions from the data and can replicate them during the reconstruction. However, tensor decomposition-based approaches do not guarantee interpretability, as it is dependent on the factorization technique used. Furthermore, the set of entities is fixed, which means that creating new entities is challenging for these approaches.

► **Definition 27** (Canonical Polyadic (CP) Decomposition). *The CP decomposition [9, 46] is a rank R factorization of an N -way tensor as a sum of R rank-1 tensors. Let $\circ$ denote the outer product, ι a scalar weight, and $\mathbf{a}$ the factor vectors. The CP decomposition is thus defined as:*

$$\mathcal{X} \approx \sum_{r=1}^R \iota_r \mathbf{a}_{1,r} \circ \mathbf{a}_{2,r} \circ \cdots \circ \mathbf{a}_{N,r}. \quad (13)$$

► **Definition 28** (RESICAL Factorization). *RESICAL [81] is a rank R -factorization, where each tensor slice $\mathcal{X}_k$ is factorized as $\mathcal{X}_k \approx \mathbf{A} \mathcal{R}_k \mathbf{A}^T$ for $k = 1, \dots, m$. The factor matrix $\mathbf{A} \in \mathbb{R}^{n \times R}$ is a latent component representation of the entities, and $\mathcal{R}_k \in \mathbb{R}^{R \times R}$ is an asymmetric matrix for the latent components of the k -th predicate. The factor matrices are typically computed by minimizing the reconstruction error across all tensor slices.*

5.2.4.1 TenGAN

TenGAN [101] is a multiplex graph generator based on a Graph Adversarial Network (GAN) [43], where the generator synthesizes the KG through factor matrices and is trained adversarially against a discriminator that distinguishes synthetic samples from real ones. The program inputs include a large multiplex graph and GAN hyperparameters. From the input graph, smaller multiplex subgraphs are sampled using ParCube [83] to form the training data. TenGAN has one generator and one discriminator network. The discriminator applies a Graph Convolutional Network per view, max-pools the outputs, and predicts real vs synthetic through a fully connected layer. Training uses a Wasserstein GAN loss function. Two variants are proposed based on different generator architectures: TenGAN-CP, based on the CP decomposition, and TenGAN-R, based on the RESICAL decomposition.

5.2.4.2 SynthG

SynthG [29] starts by materializing and converting the KG into a three-way tensor with literal nodes being disregarded. It then computes the RESICAL decomposition, applies a random rotation to the factor matrix $\mathbf{A}$, and regenerates the KG with this newly rotated matrix. The transformation combines the SVD of $\mathbf{A}$ with an orthogonal matrix obtained by applying QR decomposition to a randomly generated matrix. This random rotation also ensures that the approach can produce diverse graphs.

5.2.5 Foundation Model-based Knowledge Graph Generation

Foundation model-based graph generators leverage pre-trained foundation models to synthesize KGs. These generators rely on the knowledge and generative capabilities encoded in the foundation model during pre-training, optionally adapting it through fine-tuning or prompting. Foundation model-based approaches capture rich patterns from large pre-training corpora. They do not require training generative models from scratch, which makes them applicable even when the reference KG is small. However, relying on foundation models introduces several limitations. The fidelity of the resulting KG depends on the suitability of the chosen foundation model, and the generated content may inherit factual inconsistencies from it. Biases encoded during pre-training may propagate into the synthetic KG, which is particularly problematic when the KG is later used for downstream tasks. In addition, triples seen during pre-training may be replicated, which compromises the synthetic nature of the SKG and can leak sensitive information when the reference graph is private. The computational cost can also restrict the scale of the produced KGs.

This category is still emerging. While it currently consists of only a single approach in this survey, we expect it to grow as foundation models mature.

5.2.5.1 Fu

Fu et al. [38] introduce a KG generator that leverages large language models to operate on an intermediate text representation instead of a graph structure. The input is a reference KG. First, the reference KG is converted to text using a Graph-to-Text transformer [23]. Second, synthetic text is generated based on the reference text with GPT-3.5-Turbo in two ways: in-context prompting with sentence examples or fine-tuning on sentence sequences. In their experiments, the number of generated sentences is five times the number of input sentences. The synthetic text is filtered by computing the Mahalanobis distance of each synthetic-text embedding from the distribution of reference-text embeddings and discarding the 10% with the largest distances. Third, the remaining text is converted back to a KG with a Text-to-Graph transformer [72].

Beyond the general limitations of foundation model-based generation, this approach has a specific drawback: routing generation through an intermediate text representation incurs information loss. This could be mitigated by using LLMs capable of handling graph structures directly.

6 Performance Evaluation

This section expands on the evaluation scenarios and metrics found in the current literature, identifying common tasks and respective challenges. The summary of the evaluation tasks performed by each approach is presented in Table 11.

6.1 Task-agnostic Evaluation

Task-agnostic evaluation directly assesses the performance of the SKGGs. It focuses on two aspects:

- (i) generation efficiency, which reflects a generator’s scalability, and
- (ii) structural fidelity, which reflects a synthetic graph’s faithfulness.

6.1.1 Efficiency

Efficiency is measured by the generation runtime and memory consumption. Both low runtime and memory usage are crucial for generating large-scale graphs. Runtime is typically measured as either the total time required for an approach to generate a graph or as a data generation rate. Monitoring the memory consumption is essential to ensure generation scalability and optimal resource utilization. Runtime measurements are more common [53, 36, 97, 109, 84, 6, 73, 116, 93, 57, 91, 61, 118, 20, 18] than memory consumption monitoring [109, 84].

TrillionG [84] is the generator with the highest data generation rate among the surveyed approaches.

6.1.2 Structural Fidelity

Structural analysis is an evaluation scenario where graph properties are measured and compared. Both comparisons of graph characteristics between synthetic and target data, as well as direct evaluations of the synthetic graph's characteristics, fall into this category.

Generated Distribution Analysis. Evaluating the similarity of the synthetic graphs' properties against that of the target graph is an intuitive evaluation approach. A frequently modelled feature is the node degree distribution. KRELM [1] uses the Jensen-Shannon Divergence (JSD) to quantify the similarity between the two distributions. The JSD inspires two metrics: model precision and model coverage. Model precision is defined as the proportion of relations with at least 500 entities successfully regenerated, and model coverage is the proportion of relations successfully regenerated over all relations. Extended GDD (GDD^x) [36] uses the Chi-squared test to measure the statistical significance of the observed differences. In approaches that generate a sequence of graphs, measuring the stability of nodes [116] measures how a node maintains its degree across different graphs in a sequence. TenGAN [101] measures the Mean Maximum Mean Discrepancy (MMD) of clustering coefficient, graph orbit, and degree distributions. TrillionG [84] plots the degree distribution to confirm that the desired in- and out-degree distributions were achieved.

Graph structure. Another approach involves comparing the structural properties of the generated graph to those expected in real-world knowledge graphs, such as through isomorphic subgraph matching. For example, GDDx [36] measures for different graph sizes whether the generated graphs have the expected graph structure with Exact Subgraph Matching [68]. PoDiGG [109] computes a realism factor through the actual and the expected distance between elements, e.g., stops, routes. RBench [91] measures the relationship specialty and the literal diversity. BGen [61] uses blank node matching using a signature-based method by [115]. GenACT [102] monitors the synthetic activity volume over time for 5 synthetic conferences. TWIG [105] measures the quality of its synthetic tweets with experts trying to distinguish whether the tweets are real or synthetic.

Ontology coverage. Structural fidelity can also be assessed by measuring how the generated graph conforms to an ontology. Unlike the metrics above, these metrics do not require a target graph for comparison as the ontology itself defines the expected coverage. Hnatkowska [48] measures data completeness with four metrics from OntoQA [111]. Average population captures the ratio of generated individuals to concepts, class richness the fraction of concepts with at least one instance, class fullness the per-concept ratio of actual to expected instances, and relationship richness the share of schema-defined properties that are actually instantiated. RODG [20] reports

six coverage metrics over the ontology’s signature reachable from a chosen root class. Class, datatype property, and object property coverage measure the fraction of classes and properties that appear in at least one generated description, while three distribution metrics quantify how uniformly individuals and property assertions are spread across them.

Error rate. Comparing the quality of the generated graph against the input graph is possible for instance-driven approaches. While the output graph will differ from the input, some characteristics should be maintained in the synthetic graph. LEMMING [97] defines an error rate that quantifies deviations of graph properties in the synthetic graph from the input graphs. TenGAN [101] measures the normalized reconstruction error of the CP decomposition for different ranks. It samples tensors from the generated and the original graphs, calculates error vectors with the reconstruction error, and computes the sum of the Earth Mover’s Distance (EMD) between the error vectors to generate a custom scoring metric called TenScore.

Structuredness. Evaluating the quality of the generated graphs often requires a comparison to a target graph. However, some methods define alternative metrics that allow them to quantify the quality of the synthetic graph without a target graph. An example of such a metric is structuredness, also known as coherence. Structuredness is a concept introduced by [31] that expresses how closely the instances of a synthetic graph conform to a type system. Approaches that assess the structuredness of the generated graphs are [29, 109, 104, 91].

Regarding structural fidelity, a distinction has to be made between instance-driven and schema-driven approaches. Among instance-driven approaches, LEMMING [97], Melo [73], and RBench [91] explicitly model multiple structural patterns in the source data and therefore achieve high structural fidelity. Schema-driven approaches that output KGs with a high structural fidelity are approaches that allow a high degree of control over multiple graph characteristics. This is the case of Pygraft [53], gMark [6], and GDD^x [36]. In domain-specific settings, specialized generators may be more suitable than general-purpose approaches. For example, PoDiGG [109] and TrainBenchmark [108] for the transportation domain.

6.2 Downstream Tasks

A primary objective of synthetic data generation is the provision of a foundation to benchmark knowledge graph-based applications. Consequently, different approaches target different application scenarios. These lead to multiple downstream evaluation tasks, including triplestore benchmarking, reasoning engine evaluation, link prediction, type prediction, and classification evaluation scenarios.

6.2.1 Triplestore Benchmarking

Triplestore benchmarking focuses on the foundational SPARQL query execution by assessing the performance of triplestores in a controlled environment [24]. Examples of approaches that benchmark triplestores are [126, 55, 97, 29, 6, 60, 33, 91, 118, 3, 40, 103, 108]. The fundamental assumption behind using triplestore benchmarking as a proxy for the evaluation of synthetic knowledge graph generation is that a high-quality synthetic graph should lead to loads similar to those of a non-artificial KG when queried. Common metrics used in triplestore benchmarking are summarized in Table 10.

■ **Table 10** Triplestore benchmarking metrics.

Metric	Description
Query runtime	Duration a SPARQL query takes to run against a specific storage solution.
Queries per Second (QpS)	Number of queries processed per second by the triplestore.
Query Mixes per Hour (QMph)	Number of query mixes, i.e., collections of SPARQL queries, the triplestore processed in an hour.
Query completeness ¹⁸	Whether a SPARQL query returns the expected result.
Loading time	Duration required to load the KG into the triplestore’s memory.
Storage size	Memory space occupied by the triplestore after graph loading.
Exceptions	Number of errors encountered during query execution.

SKGGs that generate queries along with the data, such as gMark [6, 116, 2], WatDiv [3, 40], LDBC [33, 60, 104, 122], and the Geographica [55] generators, are by design suited for triplestore benchmarking. The full workload is shipped with the synthetic graphs, leaving no further configuration to the user. Recent developments have weakened this association. Tools such as Sparqlescope [8] apply query templates to arbitrary KGs to generate the workload. While this shifts more responsibility to the user, it also removes the requirement that an SKGG must provide query logs. Since benchmarking triplestores often involves stress-testing, the size of the generated graph remains an important selection criterion. Other approaches that produce large graphs include TrillionG [84], KRELM [1], and SynthG [29].

6.2.2 Reasoning Engines

Reasoning engines infer new knowledge from existing data [58, 78], often through logical rules. DLUBM [59] assesses the query runtime under different inference rules. GenACT [102] measures the latency and memory consumption of reasoning engines for temporal queries. ABRN [132] evaluates the reasoning quality, efficiency, and scalability. The latter are measured by runtime and memory consumption as the dataset grows. OWL2Bench [103] evaluated the scalability of six reasoners, measured by runtime and when each reasoner ran out of heap space. JPoT [18] measures the runtime and memory consumption of a reasoner during consistency checking, reporting the heap-space limits at which reasoning becomes infeasible as the number of generated assertions grows.

A generator’s ontology support directly determines its suitability for benchmarking reasoning engines. SKGGs that support expressive axioms, enable a more comprehensive evaluation of reasoning quality, scalability, and inference performance. Such expressiveness may include a broad range of OWL 2 profiles, including EL, QL, RL, and DL. OWL2Bench [103] and GenACT [102] explicitly consider these profiles. Several generators consider additional rule and axiom systems. ABRN [132] supports SWRL and OWL axioms. Pygraft [53] supports OWL 2 axioms. Melo et al. [73] mine Horn rules and apply the rules during the KG generation.

¹⁸Result correctness is rarely measured based on the assumption that triplestores implement SPARQL as specified and thus return correct results.

■ **Table 11** Summary of the evaluation setup. Approaches are marked in the respective column if they evaluated the generator’s efficiency (E) or the structure of the generated graph (S), or if they performed any of the following evaluation tasks: triplestore benchmarking (TB), reasoning engine benchmarking (RE), link prediction (LP), type prediction (TP), binary classification (BC), or crawling (CR).

Approach	E	S	TB	RE	LP	TP	BC	CR
RPQBench [118]	✓	✗	✓	✗	✗	✗	✗	✗
Fu [38]	✗	✗	✗	✗	✗	✗	✗	✗
Pygraft [53]	✓	✗	✗	✗	✗	✗	✗	✗
KRELM [1]	✗	✓	✗	✗	✗	✗	✗	✗
TenGAN [101]	✗	✓	✗	✗	✗	✗	✓	✗
GenACT [102]	✗	✓	✗	✓	✗	✗	✗	✗
RD ² [126]	✗	✗	✓	✗	✗	✗	✗	✗
Hnatkowska [48]	✗	✓	✗	✗	✗	✗	✗	✗
ABRN [132]	✗	✗	✗	✓	✗	✗	✗	✗
DLCC [90]	✗	✗	✗	✗	✗	✗	✓	✗
GDD ^x [36]	✓	✓	✗	✗	✗	✗	✗	✗
Geographica2 [55]	✗	✗	✓	✗	✗	✗	✗	✗
LEMMING [97]	✓	✓	✓	✗	✗	✗	✗	✗
SynthG [29]	✗	✓	✓	✗	✗	✗	✗	✗
TWIG [105]	✗	✗	✗	✗	✗	✗	✓	✗
ORCA [96]	✗	✗	✗	✗	✗	✗	✗	✓
RODG [20]	✓	✓	✗	✗	✗	✗	✗	✗
OWL2Bench [103]	✗	✗	✓	✓	✗	✗	✗	✗
Waudby [122]	✗	✗	✗	✗	✗	✗	✗	✗
PoDiGG [109]	✓	✓	✗	✗	✗	✗	✗	✗
Stream WatDiv [40]	✗	✗	✓	✗	✗	✗	✗	✗
TrillionG [84]	✓	✓	✗	✗	✗	✗	✗	✗
DLUBM [59]	✗	✗	✗	✓	✗	✗	✗	✗
EGG [2]	✗	✗	✗	✗	✗	✗	✗	✗
gMark [6]	✓	✗	✓	✗	✗	✗	✗	✗
Melo [73]	✓	✗	✗	✗	✓	✓	✗	✗
MonStaGen [116]	✓	✓	✗	✗	✗	✗	✗	✗
TrainBenchmark [108]	✗	✗	✓	✗	✗	✗	✗	✗
JPoT [18]	✓	✗	✗	✓	✗	✗	✗	✗
EvoGen [71]	✗	✗	✗	✗	✗	✗	✗	✗
GAIA [93]	✓	✗	✗	✗	✗	✗	✗	✗
LDBC-SPB [60]	✗	✓	✓	✗	✗	✗	✗	✗
LinkGen [57]	✓	✗	✗	✗	✗	✗	✗	✗
Spasić [104]	✗	✓	✗	✗	✗	✗	✗	✗
LDBC-SNB [33]	✗	✗	✓	✗	✗	✗	✗	✗
RBench [91]	✓	✓	✓	✗	✗	✗	✗	✗
WatDiv [3]	✗	✗	✓	✗	✗	✗	✗	✗
BGen [61]	✓	✓	✗	✗	✗	✗	✗	✗

6.2.3 Link Prediction

The link prediction task aims to determine whether a triple of the form (s, p, o) should exist in a KG [80]. A common approach is to learn a scoring function that yields higher scores for true triples and lower scores for false triples [28]. Melo et al. [73] tested link prediction techniques and measured the area under the Precision-Recall and the ROC curve. Link prediction poses a challenge to synthetic graph evaluation, as the generated graphs are often anonymized. This means that the set of entities and relations often changes, complicating the evaluation scenario as a dedicated test set is unavailable. This limitation is addressed by Melo et al. [73] through a 5-fold cross-validation evaluation scheme. However, a cross-validation strategy incurs a high computational cost, especially for large KGs.

Melo [73] is the only approach that explicitly evaluates generated data on a link prediction downstream task. Different Knowledge Graph Embedding (KGE) models support different properties. For example, some models cannot adequately represent transitive, antisymmetric, or irreflexive properties. Consequently, SKGGs with broader support for OWL axioms are better suited for evaluating the capabilities and limitations of KGE models across diverse logical patterns. This list would include approaches like Pygraft [53], OWL2Bench [103], and JPoT [18]. GenACT [102] could be considered for temporal KGE models.

6.2.4 Type Prediction

Similarly to the link prediction task, type prediction determines whether triples of the form $(s, \text{rdf:type}, o)$ should exist in a KG [73]. Common approaches include reasoning via inference rules, link-based approaches [86], and multi-label classification [13, 74]. For instance, Melo et al. [73] test different type prediction methods using accuracy and F1-score metrics.

Similarly to the previous section, Melo [73] is the only approach that explicitly evaluated on a type prediction task. For type prediction, the requirements are less strict. In particular, SKGGs that provide a class hierarchy and support RDFS axioms are typically sufficient to generate a meaningful evaluation. Beyond Melo [73], several generators would fit this setting, including Pygraft [53], OWL2Bench [103], JPoT [18], LEMMING [97], and GAIA [93].

6.2.5 Binary Classification

TenGAN [101] uses a binary classification model to predict whether the data is real or generated. They compute graph2vec [77] embeddings and feed the embeddings of each graph view to a Support Vector Machine (SVM). The final classification is then decided based on a majority vote of the ensemble of SVMs. The reported metrics were the F1-score and the accuracy. DLCC [90] measured the accuracy of binary classifiers to test the class separability of different KGE models. TWIG [105] measures the accuracy on a supervised text classification task trained to predict whether a tweet is real or fake.

The choice of SKGGs for binary classification depends strongly on the nature of the input data. For this reason, we do not provide a general recommendation.

6.2.6 Crawling

Data Web crawlers discover and retrieve KGs published across the Web by following links between resources. Evaluating them on the real Web is hard to reproduce, as the available datasets change over time and the ground truth of retrievable triples is unknown. Synthetic KGs address both

issues by providing a controlled environment with a known set of triples to be retrieved. ORCA [96] evaluates Data Web crawlers on a synthetic Data Web of interconnected KGs. The reported metrics are recall, runtime, and memory consumption.

ORCA [96] is the only generator in our survey that generates interconnected graphs that are crawlable, i.e., given a set of starting points, a crawler is able to access all data of all graphs by following the directed edges. ORCA is therefore uniquely suited for crawling-based scenarios. RODG [20] also generates multiple graphs. However, these graphs are independent and not interconnected.

7 Technical Details

This section addresses the practical requirements a user often encounters when selecting and deploying an SKGG. Beyond evaluating algorithmic sophistication, practitioners face many constraints: availability of source code, maintenance status, computational scalability, programming language, and dependency ecosystems. As benchmarking is one of the main drivers of SKG generation, such information is essential to select the appropriate SKGG. We present in Table 12 the technical details of the surveyed SKGGs.

Programming Languages and Dependencies. JVM-based languages (Java and Scala) dominate the surveyed generators, accounting for 20 out of 38 implementations. Followed by C++ with 5, and Python with 4 generators. JavaScript and R appear with a single generator each. Seven generators lack publicly available source code and are marked as unknown. Java’s prevalence reflects its established role in KGs tools, particularly through frameworks such as Apache Jena and OWL API. However, reliance on these frameworks often introduces scalability constraints, e.g., reasoning with OWL API over large KGs is computationally expensive. Python-based generators offer ease of configuration but may similarly face scalability challenges, especially when built with libraries like RDFLib, which lack optimization. C++-based generators tend to be more lightweight and therefore more suitable for large-scale generation.

Repository Availability and Maintenance. Of the 38 surveyed generators, 31 have available source code. However, the absence of source code for seven generators limits their practical utility and reproducibility, highlighting the importance of open-source availability for the approach’s long-term adoption. A second concern is the maintenance status of the repositories. Only six generators showed activity in 2026, pointing to the lack of maintenance across existing generators. While some generators are still widely used, it becomes increasingly difficult to include non-maintained repositories in projects built on modern technology stacks.

Parallelization and Acceleration Strategies. Only a minority of generators explicitly mention parallelization or acceleration techniques. We note that a generator marked without acceleration may still support some form of parallel execution in practice, but absent an explicit mention, we do not consider it as such. Among the surveyed tools, ten support some form of acceleration: multithreading is the most common strategy. GPU acceleration appears in TenGAN and ABRN. The remaining generators rely on sequential generation, often to resolve schema constraints in a strict priority order and thereby avoid conflicts during triple creation. While this design choice simplifies semantic consistency, it introduces a critical bottleneck as benchmarking scales to billion-scale triples.

■ **Table 12** Implementation details: source code link, last update (as of May 2026), order of magnitude of the largest reported graph size (in triples), acceleration techniques, programming language, and key dependencies. * denotes estimates from reported node counts, ** denotes estimates from the reported number of generated universities, and [†] denotes a size reported for plain graphs rather than KGs.

Approach	Link	Last Updated	Order of Magnitude	Acceleration	Language	Key Dependencies
RPQBench [118]	🔗	2024	10^7	-	C++	-
Fu [38]	✗	-	10^4	-	-	-
Pygraft [53]	🔗	2026	10^6	-	Python	RDFLib, Owlready2
KRELM [1]	🔗	2024	10^9	-	Java	-
TenGAN [101]	🔗	2023	$< 10^4$	Multithreading, GPU	Python	PyTorch, TensorLy, NetworkX
GenACT [102]	🔗	2024	10^7	-	Java	OWL API
RD ² [126]	🔗	2023	10^1	-	Java	Apache Jena, RDF4J
Hnatkowska [48]	🔗	2023	-	-	Java	PDGF
ABRN [132]	🔗	2022	10^4	GPU	Java	OWL API, Pellet
DLCC [90]	🔗	2022	10^5	-	Java	Apache Jena
GDD ^x [36]	✗	-	10^4	-	-	-
Geographica2 [55]	🔗	2024	10^8	-	Java	-
LEMMING [97]	🔗	2026	10^6	Multithreading	Java	Apache Jena, grph
SynthG [29]	🔗	2021	10^8	Multithreading	R	RDFTensor
TWIG [105]	🔗	2020	10^6	Multithreading	Java	Apache Jena
ORCA [96]	🔗	2023	10^5	-	Java	-
RODG [20]	✗	-	-	-	-	-
OWL2Bench [103]	🔗	2022	10^7	-	Java	OWL API
Waudby [122]	🔗	2026	10^{10}	Apache Spark	Java, Scala	LDBC-SNB
PoDiGG [109]	🔗	2018	10^6	-	JavaScript	-
Stream WatDiv [40]	🔗	2018	10^6	-	C++	-
TrillionG [84]	🔗	2019	$10^{12\dagger}$	Apache Spark	Scala	Apache Spark
DLUBM [59]	🔗	2018	10^{5**}	Docker Swarm	Java	-
EGG [2]	🔗	2017	10^{7*}	-	Python	RDFLib, gMark
gMark [6]	🔗	2023	10^{8*}	-	C++	-
Melo [73]	🔗	2017	10^5	-	Python	RDFLib
MonStaGen [116]	🔗	2023	10^{8*}	-	C++	-
TrainBenchmark [108]	🔗	2022	10^7	-	Java	Apache Jena
JPoT [18]	✗	-	10^6	-	-	-
EvoGen [71]	🔗	2017	10^{6**}	-	Java	Apache Jena
GAIA [93]	✗	-	10^{9*}	-	-	-
LDBC-SPB [60]	🔗	2026	10^9	-	Java	-
LinkGen [57]	🔗	2018	10^8	-	Java	Apache Jena
Spasić [104]	🔗	2026	10^9	Multithreading	Java	LDBC-SNB
LDBC-SNB [33]	🔗	2026	10^{10}	Apache Spark	Java, Scala	Apache Spark
RBench [91]	✗	-	10^8	-	-	-
WatDiv [3]	🔗	2023	10^8	-	C++	-
BGen [61]	✗	-	10^7	-	-	-

Scalability. We report the maximum graph size that each approach’s authors have tested on. It is important to note that these values reflect the largest configurations evaluated in the respective publications, not necessarily the upper bound of what each generator is technically capable of producing. We report the graph size as the number of triples when possible. The scale ranges from dozens to trillions of edges. Out of the 38 surveyed generators, only TrillionG has been evaluated at the trillion scale. However, TrillionG only reports the classical graph size and not the KG. Six generators reach the billion scale: KRELM, GAIA, LDBC-SPB, Spasić, Waudby, and LDBC-SNB. The last four are all part of, or built on, the LDBC family of benchmarks. The majority, 20 generators, have been tested in the million-to-hundred-million range, and nine remain below the million-triple mark. It is unclear what the generated graph size is for two generators. A direct comparison across all generators remains difficult, as reported sizes are not always expressed in the same unit. The gMark family generators, gMark, EGG, and MonStaGen, report node counts rather than triple counts. However, assuming an average degree lower than ten (which is often the case [35]), the resulting number of triples lies in the same order of magnitude as the number of generated nodes. For LUBM-based generators, we convert the reported number of synthetic universities into triple counts using LUBM [45] statistics.

Both source code availability and demonstrated scalability have a direct impact on the adoption rate of SKGGs for benchmarking scenarios. A generator that is not openly available cannot be reproduced, extended, or integrated into a benchmarking pipeline regardless of its merits. Likewise, some benchmarking applications, such as triplestores, rely in part on stress-testing the underlying engines. Stress-testing a triplestore is only possible if the SKGG can generate large-scale KGs [12]. Applying these two criteria leaves 12 out of the 38 surveyed generators: WatDiv, LDBC generator family, gMark, MonStaGen, Geographica2, TrillionG, LinkGen, SynthG, and KRELM. However, when we also take into account the respective repository health, only the LDBC generator family has had recent updates.

8 Challenges and Future Work

While the preceding sections have shown that synthetic KG generation has matured along several axes, the survey also surfaces a set of recurring gaps. We synthesise these in Section 8.1 as open issues that follow directly from the limitations observed across the surveyed generators, and then step back in Section 8.2 to discuss broader research directions we believe the field could pursue beyond incremental improvements to existing pipelines.

8.1 Remaining Challenges

Despite significant progress in knowledge graph generation, several challenges remain.

Limited support for literal nodes. A common limitation across many generators is their limited support for literal nodes. These nodes are prevalent in real-world KGs, with approximately 30% of triples in DBpedia involving literal values.¹⁹ Despite their widespread use, many techniques generate literals through straightforward methods, reflecting the challenge of producing truly realistic literal nodes.

Literal nodes quality assessment. The reduced support for literal node generation is exacerbated by the lack of evaluation methods designed to assess the quality of the generated literal values. This is noticeable in all of the studied evaluation scenarios. Structural metrics focus on the

¹⁹ Based on DBpedia live endpoint.

existence of a literal node, rather than the potential influence of its value over the generated graph. Triplestore benchmarking metrics rely on the query runtime and throughput, where the influence of a literal node's value is limited. As most of the KGE models also disregard literal nodes while learning embeddings, the insufficient support is also propagated to the link prediction task.

Need for more comprehensive evaluation tasks. The current evaluation of synthetic graph generators focuses primarily on generation time and structural integrity. While this evaluation focus offers valuable insights, it fails to predict the performance of solutions of real-world KGs of larger size. Furthermore, the generators generally include only one application-specific evaluation task. This provides no guarantees that the generated synthetic graphs are realistically generated and that the performance would transfer across tasks. To address these gaps, a comprehensive evaluation framework should integrate structural analysis with multiple task-based assessments, thereby offering a more holistic view of the generated graphs' potential.

Privacy assessment. One of the motivations to use synthetic data is to protect private information, as the entities present in the generated graph are anonymized. However, the anonymization degree of the generated graphs is not assessed by current generators. Designing new privacy metrics tailored to KGs or porting existing metrics from the general graph generation research to KGs could prove insightful.

Evolving graph analysis. While multiple approaches generate an evolving graph, only one accepts an evolving graph as input to the generation. However, it does not analyze the KG from a temporal perspective, e.g., via genetic algorithms. Using evolutionary metrics to analyze the graph's changes over time, or employing evolutionary algorithms to generate the temporal changes, remains an open question. Generating subsequent iterations would aid in the study of how information evolves across different domains and in the evaluation of its impact.

Generating multi-modal Knowledge Graphs. The SKGGs analyzed currently model either static or evolving KGs. In recent times, KGs containing multi-modal data – such as visual data – have emerged [119, 100, 87]. Integrating diverse data types in synthetic KGs remains an open research opportunity.

Trade-off between scalability and control. We observe a shift in the design priorities of SKGGs. Earlier works emphasized scalability with relatively simple generation mechanics, whereas newer approaches accept higher algorithmic complexity (and the scalability cost that comes with it) in exchange for finer-grained control over the output. This mirrors a shift in the benchmarking applications that motivate SKGG: from stress-testing applications, where raw size mattered, towards interpretability, explainability, and realistic downstream evaluation, where controllability is rewarded. Acknowledging this trade-off explicitly would help future work position itself more clearly.

8.2 Future Directions

The challenges above point to several directions in which SKGG research could move beyond incremental refinement of existing pipelines. We highlight four that we consider particularly promising, ranging from community infrastructure to a rethinking of how generators are architected.

A benchmarking platform for SKGGs. Evaluation in the field is fragmented: each approach tends to select their own set of metrics, as there is no consensus in the literature on a common set of metrics. This leads to multiple approaches evaluating the structure of their output graphs in just as many ways, making comparisons difficult. Methodological variety is healthy, but a shared platform on which generators can be deployed and compared under a common set of metrics and evaluation tasks would make claims about realism, scalability, and downstream utility directly verifiable. Developing a platform or integrating onto existing ones, such as HOBbit [95], would also improve the adoption rates of more recent SKGGs.

Plan first, build later. A broader observation underlying several of the challenges above is that most existing SKGGs couple structural decisions and triple-level realization into a single pipeline. Neighbouring fields such as natural language generation have moved in the opposite direction, explicitly separating content planning from realization [94], with gains in coherence. Adopting a similar two-stage view for KG generation could help avoid locally plausible but globally incoherent outputs.

Foundation models. Several recent advances in foundation models have transformed neighboring fields. Natural language processing has been reshaped by LLMs [113], and tabular data by TabPFN [50]. Applying the same paradigm to KG generation appears equally promising. However, graph foundation models have so far been studied almost exclusively on generating plain graphs rather than KGs. While the usage of diffusion models has been proposed for graphs [110], other architectures like Prior-Data Fitted Networks (PFNs) [76] or Joint Embedding Predictive Architectures (JEPAs) [62] might be promising, too. The introduction of foundation models for KG generation marks a shift away from the traditional architecture of SKGGs. The emergence of the foundation model-based SKGG category, so far represented by a single approach [38], indicates that this development is already underway and is likely to gain momentum.

9 Concluding Remarks

This survey provides a comprehensive overview of the current state of synthetic knowledge graph generation research. We identify and categorize thirty-eight approaches developed between 2014 and 2025. These approaches are categorized via a three-way classification scheme: First, according to the generation parameters, whether the generation process is basic, schema-driven, or instance-driven. Second, based on its methodology, whether the approach is rule-based, probabilistic, probabilistic-rule-augmented, tensor-decomposition-based, or foundation model-based. Probabilistic and hybrid approaches that combine probabilistic modelling with rule-based systems are the most prevalent. And third, by temporal coverage of the output graph. We provide an overview of the current evaluation scenarios of synthetic KGs. The evaluation focuses on the efficiency of the generators, the structural analysis of the generated graphs, and downstream task benchmarking. Here, triplestore benchmarking was the primary application scenario. We complement this with a dedicated discussion of the scalability and practical implementation details of the surveyed approaches.

Overall, we observed significant progress over the years. Despite these advancements, there are improvement opportunities. Key improvement areas include the limited support for literal nodes, the lack of metrics for literal node quality assessment, the need for more comprehensive evaluation environments, the need for privacy assessment of synthetic KGs, the ability to model versioned datasets, and the ability to generate multi-modal KGs. Addressing these gaps and rethinking the architectural choices that underlie current generators defines the agenda for the next generation of SKGGs.

References

- 1 Hassan Abdallah, Béatrice Markhoff, and Arnaud Soulet. A complex network model for knowledge graphs' relationships. *Semantic Web*, 2025. doi:10.1177/22104968251361342.
- 2 Karim Alami, Radu Ciucanu, and Engelbert Mephu Nguifo. Egg: A framework for generating evolving rdf graphs. In *ISWC (Posters, Demos & Industry Tracks)*, 2017. URL: <http://ceur-ws.org/Vol-1963/#paper460>.
- 3 Güneş Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. Diversified stress testing of rdf data management systems. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble, editors, *The Semantic Web – ISWC 2014*, pages 197–212, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-11964-9_13.
- 4 Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-76298-0_52.
- 5 Franz Baader, Ian Horrocks, and Ulrike Sattler. *Description Logics*, pages 21–43. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-540-92673-3_1.
- 6 Guillaume Bagan, Angela Bonifati, Radu Ciucanu, George H. L. Fletcher, Aurélien Lemay, and Nicky Advokaat. gmark: Schema-driven generation of graphs and queries. *IEEE Transactions on Knowledge and Data Engineering*, 29(4):856–869, 2017. doi:10.1109/TKDE.2016.2633993.
- 7 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. doi:10.1126/science.286.5439.509.
- 8 Hannah Bast, Johannes Kalmbach, Robin Textor-Falconi, and Christoph Ullinger. Sparqlscope: A generic benchmark for the comprehensive and concise performance evaluation of sparql engines. In *The Semantic Web – ISWC 2025: 24th International Semantic Web Conference, Nara, Japan, November 2–6, 2025, Proceedings, Part II*, pages 22–40, Berlin, Heidelberg, 2025. Springer-Verlag. doi:10.1007/978-3-032-09530-5_2.
- 9 Casey Battaglini, Grey Ballard, and Tamara G. Kolda. A practical randomized cp tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 39(2):876–901, 2018. doi:10.1137/17M112303.
- 10 Konstantina Bereta, Panayiotis Smeros, and Manolis Koubarakis. Representation and querying of valid time of triples in linked geospatial data. In Philipp Cimiano, Oscar Corcho, Valentina Pre-sutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data*, pages 259–274, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38288-8_18.
- 11 Piotr Bielak and Tomasz Kajdanowicz. Representation learning in multiplex graphs: Where and how to fuse information? In *Computational Science – ICCS 2024: 24th International Conference, Malaga, Spain, July 2–4, 2024, Proceedings, Part VI*, pages 3–18, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-63778-0_1.
- 12 Alexander Biggerl, Felix Conrads, Charlotte Behning, Mohamed Ahmed Sherif, Muhammad Saleem, and Axel-Cyrille Ngonga Ngomo. Tentriss – a tensor-based triple store. In Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web – ISWC 2020*, pages 56–73, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-62419-4_4.
- 13 Russa Biswas, Radina Sofronova, Mehwish Alam, and Harald Sack. Entity type prediction in knowledge graphs using embeddings. *CoRR*, abs/2004.13702, 2020. arXiv:2004.13702.
- 14 Brigitte Boeckmann, Amos Bairoch, Rolf Apweiler, Marie-Claude Blatter, Anne Estreicher, Elisabeth Gasteiger, Maria J Martin, Karine Michoud, Claire O'Donovan, Isabelle Phan, et al. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic acids research*, 31(1):365–370, 2003. doi:10.1093/NAR/GKG095.
- 15 John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory*. Springer Publishing Company, Incorporated, 2008.
- 16 Angela Bonifati, George Fletcher, Jan Hidders, and Alexandru Iosup. *A Survey of Benchmarks for Graph-Processing Systems*, pages 163–186. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-96193-4_6.
- 17 Angela Bonifati, Irena Holubová, Arnau Prat-Pérez, and Sherif Sakr. Graph generators: State of the art and open challenges. *ACM Comput. Surv.*, 53(2), April 2020. doi:10.1145/3379445.
- 18 Jean-Rémi Bourguet. Jpot: Just another populator of tboxes. In Juan Antonio Lossio-Ventura and Hugo Alatrística-Salas, editors, *Proceedings of the 4th Annual International Symposium on Information Management and Big Data, SIMBig 2017, Lima, Peru, September 4-6, 2017*, CEUR Workshop Proceedings, pages 109–115. CEUR-WS.org, 2017. URL: <https://ceur-ws.org/Vol-2029/paper9.pdf>.
- 19 Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn, editors, *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*, pages 442–446. SIAM, 2004. doi:10.1137/1.9781611972740.43.

- 20 Yanji Chen, Mieczyslaw M. Kokar, and Jakub J. Moskal. Rdf object description generator. *Int. J. Web Eng. Technol.*, 15(2):140–169, January 2020. doi:10.1504/ijwet.2020.109729.
- 21 Wenliang Cheng, Chengyu Wang, Bing Xiao, Weinong Qian, and Aoying Zhou. On statistical characteristics of real-life knowledge graphs. In Jianfeng Zhan, Rui Han, and Roberto V. Zicari, editors, *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, pages 37–49, Cham, 2016. Springer International Publishing.
- 22 Steven A. Cholewiak, Panos Ipeirotis, Victor Silva, and Arun Kannawadi. SCHOLARLY: Simple access to Google Scholar authors and citation using Python, 2021. doi:10.5281/zenodo.5764801.
- 23 Anthony Colas, Mehrdad Alvandipour, and Daisy Zhe Wang. GAP: A graph-aware language model framework for knowledge graph-to-text generation. In Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na, editors, *Proceedings of the 29th International Conference on Computational Linguistics*, pages 5755–5769, Gyeongju, Republic of Korea, October 2022. International Committee on Computational Linguistics. URL: <https://aclanthology.org/2022.coling-1.506/>.
- 24 Felix Conrads, Jens Lehmann, Muhammad Saleem, Mohamed Morsey, and Axel-Cyrille Ngonga Ngomo. Iguana: A generic framework for benchmarking the read-write performance of triple stores. In *The Semantic Web – ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part II*, pages 48–65, Berlin, Heidelberg, 2017. Springer-Verlag. doi:10.1007/978-3-319-68204-4_5.
- 25 The UniProt Consortium. Uniprot: the universal protein knowledgebase in 2025. *Nucleic Acids Research*, 53(D1):D609–D617, November 2024. doi:10.1093/nar/gkaf1010.
- 26 Frithjof Dau. Rdf as graph-based, diagrammatic logic. In Floriana Esposito, Zbigniew W. Raś, Donato Malerba, and Giovanni Semeraro, editors, *Foundations of Intelligent Systems*, pages 332–337, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11875604_38.
- 27 S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: the roles of xml and rdf. *IEEE Internet Computing*, 4(5):63–73, 2000. doi:10.1109/4236.877487.
- 28 Caglar Demir and Axel-Cyrille Ngonga Ngomo. Convolutional complex knowledge graph embeddings. In Ruben Verborgh, Katja Hose, Heiko Paulheim, Pierre-Antoine Champin, Maria Maleshkova, Oscar Corcho, Petar Ristoski, and Mehwish Alam, editors, *The Semantic Web*, pages 409–424, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-77385-4_24.
- 29 Abdelmoneim Amer Desouki, Felix Conrads, Michael Röder, and Axel-Cyrille Ngonga Ngomo. Synthg: Mimicking rdf graphs using tensor factorization. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pages 76–79, 2021. doi:10.1109/ICSC50631.2021.00017.
- 30 Li Ding and Tim Finin. Characterizing the semantic web on the web. In Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora M. Aroyo, editors, *The Semantic Web - ISWC 2006*, pages 242–257, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11926078_18.
- 31 Songyun Duan, Anastasios Kementsietsidis, Kavitha Srinivas, and Octavian Udrea. Apples and oranges: a comparison of rdf benchmarks and real rdf datasets. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 145–156, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1989323.1989340.
- 32 Paul Erdos and Alfred Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 1960.
- 33 Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. The ldsc social network benchmark: Interactive workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 619–630, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2723372.2742786.
- 34 Faezeh Faez, Yassaman Ommi, Mahdiah Soleymani Baghshah, and Hamid R. Rabiee. Deep graph generators: A survey. *IEEE Access*, 9:106675–106702, 2021. doi:10.1109/ACCESS.2021.3098417.
- 35 Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and YAGO. *Semantic Web*, 9(1):77–129, 2018. doi:10.3233/SW-170275.
- 36 Zaiwen Feng, Wolfgang Mayer, Keqing He, Selasi Kwashie, Markus Stumptner, Georg Grossmann, Rong Peng, and Wangyu Huang. A schema-driven synthetic knowledge graph generation approach with extended graph differential dependencies (gddxs). *IEEE Access*, 9:5609–5639, 2021. doi:10.1109/ACCESS.2020.3048186.
- 37 Javier D Fernández, Miguel A Martínez-Prieto, Pablo de la Fuente Redondo, and Claudio Gutiérrez. Characterising rdf data sets. *Journal of Information Science*, 44(2):203–229, 2018. doi:10.1177/0165551516677945.
- 38 Shuran Fu, Peihua Mai, Zhang Jingqi, and Yan Pang. Utilizing language models for synthetic knowledge graph generation. In *ICLR 2025 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2025. URL: <https://openreview.net/forum?id=IutH9tRtMI>.
- 39 Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 413–422, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488388.2488425.

- 40 Libo Gao, Lukasz Golab, M. Tamer Özsu, and Güneş Aluç. Stream watdiv: A streaming rdf benchmark. In *Proceedings of the International Workshop on Semantic Big Data*, SBD'18, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3208352.3208355.
- 41 George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A benchmark for geospatial rdf stores (long version). In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, pages 343–359, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-41338-4_22.
- 42 Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: an owl 2 reasoner. *Journal of automated reasoning*, 53:245–269, 2014. doi:10.1007/S10817-014-9305-1.
- 43 Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, October 2020. doi:10.1145/3422622.
- 44 Xiaojie Guo and Liang Zhao. A systematic survey on deep generative models for graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5370–5390, 2023. doi:10.1109/TPAMI.2022.3214832.
- 45 Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005. Selected Papers from the International Semantic Web Conference, 2004. doi:10.1016/j.websem.2005.06.005.
- 46 Richard A Harshman et al. Foundations of the parafac procedure: Models and conditions for an “explanatory” multi-modal factor analysis. *UCLA working papers in phonetics*, 16(1):84, 1970.
- 47 Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. *Foundations of semantic web technologies*. Chapman and Hall/CRC, 2009.
- 48 Bogumila Hnatkowska and Mateusz Kimmel. Data generation based on domain ontology. In Alberto Rodrigues da Silva, Miguel Mira da Silva, Jacinto Estima, Chris Barry, Michael Lang, Henry Linger, and Christoph Schneider, editors, *Information Systems Development: Organizational Aspects and Societal Trends -Proceedings of the 31st International Conference on Information Systems Development, ISD 2023, Lisbon, Portugal, 30 August - 1 September 2023*. Instituto Superior Técnico / Association for Information Systems, 2023. URL: <https://aisel.aisnet.org/isd2024/proceedings2023/ontologies/2>.
- 49 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D’amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4), July 2021. doi:10.1145/3447772.
- 50 Noah Hollmann, Samuel Müller, Katharina Eggen-sperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations 2023*, 2023.
- 51 Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semant. Web*, 2(1):11–21, January 2011. doi:10.3233/SW-2011-0025.
- 52 Zhisheng Huang, Frank van Harmelen, Annette ten Teije, and Kathrin Dentler. Knowledge-based patient data generation. In David Riaño, Richard Lenz, Silvia Miksch, Mor Peleg, Manfred Reichert, and Annette ten Teije, editors, *Process Support and Knowledge Representation in Health Care*, pages 83–96, Cham, 2013. Springer International Publishing. doi:10.1007/978-3-319-03916-9_7.
- 53 Nicolas Hubert, Pierre Monnin, Mathieu d’Aquino, Davy Monticolo, and Armelle Brun. Pygraft: Configurable generation of synthetic schemas and knowledge graphs at your fingertips. In Albert Meroño Peñuela, Anastasia Dimou, Raphaël Troncy, Olaf Hartig, Maribel Acosta, Mehwish Alam, Heiko Paulheim, and Pasquale Lisena, editors, *The Semantic Web*, pages 3–20, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-60635-9_1.
- 54 EunJeong Hwang, Veronika Thost, Vered Shwartz, and Tengfei Ma. Knowledge graph compression enhances diverse commonsense generation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 558–572, Singapore, December 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.emnlp-main.37.
- 55 Theofilos Ioannidis, George Garbis, Kostis Kyzirakos, Konstantina Bereta, and Manolis Koubarakis. Evaluating geospatial rdf stores using the benchmark geographica 2. *Journal on Data Semantics*, 10(3):189–228, 2021. doi:10.1007/s13740-021-00118-x.
- 56 Xiongnan Jin, Zhilin Wang, Manni Duan, Yan Shao, Xingyun Hong, Yongheng Wang, and Byungkook Oh. A survey on knowledge graph evolution: proliferation, dynamic embedding, and versioning. *International Journal of Web and Grid Services*, 21(1):88–111, 2025. doi:10.1504/IJWGS.2025.144974.
- 57 Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong. Linkgen: Multipurpose linked data generator. In Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil, editors, *The Semantic Web – ISWC 2016*, pages 113–121, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-46547-0_12.
- 58 Tobias Käfer and Andreas Harth. Specifying, monitoring, and executing workflows in linked data environments. In Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web – ISWC 2018*, pages 424–440, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-030-00671-6_25.

- 59 Felix Leif Keppmann, Maria Maleshkova, and Andreas Harth. Dlubm: A benchmark for distributed linked data knowledge base systems. In Hervé Panetto, Christophe Debruyne, Walid Gaaloul, Mike Papazoglou, Adrian Paschke, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, pages 427–444, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-69459-7_29.
- 60 Venelin Kotsev, Nikos Minadakis, Vassilis Papakonstantinou, Orri Erling, Irini Fundulaki, and Atanas Kiryakov. Benchmarking rdf query engines: The case of semantic publishing benchmark, 2016. URL: <http://ceur-ws.org/Vol-1700/#paper-01>.
- 61 Christina Lantzaki, Thanos Yannakis, Yannis Tzitzikas, and Anastasia Analyti. Generating synthetic rdf data with connected blank nodes for benchmarking. In Valentina Presutti, Claudia d’Amato, Fabien Gandon, Mathieu d’Aquin, Stefan Staab, and Anna Tordai, editors, *The Semantic Web: Trends and Challenges*, pages 192–207, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-07443-6_14.
- 62 Yann LeCun. A path towards autonomous machine intelligence. *Open Review*, pages 1–62, 2022.
- 63 Geon Lee, Fanchen Bu, Tina Eliassi-Rad, and Kijung Shin. A survey on hypergraph mining: Patterns, tools, and generators. *ACM Comput. Surv.*, 57(8), March 2025. doi:10.1145/3719002.
- 64 Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(33):985–1042, 2010. doi:10.5555/1756006.1756039.
- 65 Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In Alípio Mário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, pages 133–145, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 66 Zixuan Li, Xiaolong Jin, Wei Li, Saiping Guan, Jiafeng Guo, Huawei Shen, Yuanzhuo Wang, and Xueqi Cheng. Temporal knowledge graph reasoning based on evolutionary representation learning. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 408–417, 2021. doi:10.1145/3404835.3462963.
- 67 Seung-Hwan Lim, Sangkeun Lee, Sarah S Powers, Mallikarjun Shankar, and Neena Imam. Survey of approaches to generate realistic synthetic graphs. Technical report, Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States), October 2016. doi:10.2172/1339361.
- 68 Haibin Liu, Vlado Kešelj, and Christian Blouin. Exploring a subgraph matching approach for extracting biological events from literature. *Computational Intelligence*, 30(3):600–635, 2014. doi:10.1111/COIN.12009.
- 69 Jiaqi Liu, Qin Zhang, Luoyi Fu, Xinbing Wang, and Songwu Lu. Evolving knowledge graphs. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2260–2268, 2019. doi:10.1109/INFOCOM.2019.8737547.
- 70 Li Ma, Yang Yang, Zhaoming Qiu, Guotong Xie, Yue Pan, and Shengping Liu. Towards a complete owl ontology benchmark. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications*, pages 125–139, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- 71 Marios Meimaris and George Papastefanatos. The evogen benchmark suite for evolving rdf data. In *MEPDaW/LDQ@ ESWC*, pages 20–35, 2016. URL: https://ceur-ws.org/Vol-1585/mepdaw2016_paper_03.pdf.
- 72 Igor Melnyk, Pierre Dognin, and Payel Das. Knowledge graph generation from text. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1610–1622, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.findings-emnlp.116.
- 73 André Melo and Heiko Paulheim. Synthesizing knowledge graphs for link and type prediction benchmarking. In Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig, editors, *The Semantic Web*, pages 136–151, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-58068-5_9.
- 74 André Melo, Heiko Paulheim, and Johanna Völker. Type prediction in rdf knowledge bases using hierarchical multilabel classification. In *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics, WIMS ’16*, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2912845.2912861.
- 75 Amadis Antonio Martinez Morales and Maria Esther Vidal Serodio. A directed hypergraph model for rdf. In Elena Simperl, Joerg Diederich, and Guus Schreiber, editors, *Proceedings of the KWEPSY 2007 Knowledge Web PhD Symposium 2007*, 2007.
- 76 Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022. URL: <https://openreview.net/forum?id=KSugKcbNf9>.
- 77 Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. In *Proceedings of the 13th International Workshop on Mining and Learning with Graphs (MLG)*, 2017.
- 78 Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. Rdfbox: A highly-scalable rdf store. In Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d’Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab, editors, *The Semantic Web - ISWC 2015*, pages 3–20, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-25010-6_1.
- 79 Axel-Cyrille Ngonga Ngomo, Sören Auer, Jens Lehmann, and Amrapali Zaveri. *Introduction to*

- Linked Data and Its Lifecycle on the Web*, pages 1–99. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-10587-1_1.
- 80 Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016. doi: 10.1109/JPROC.2015.2483592.
 - 81 Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML’11*, pages 809–816, Madison, WI, USA, 2011. Omnipress. URL: https://icml.cc/2011/papers/438_icmlpaper.pdf.
 - 82 Zhengyu Pan, Tao Zhu, Hong Liu, and Huansheng Ning. A survey of rdf management technologies and benchmark datasets. *Journal of Ambient Intelligence and Humanized Computing*, 9(5):1693–1704, 2018. doi:10.1007/S12652-018-0876-2.
 - 83 Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D. Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 521–536, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-33460-3_39.
 - 84 Himchan Park and Min-Soo Kim. Trilliong: A trillion-scale synthetic graph generator using a recursive vector model. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD ’17*, pages 913–928, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3035918.3064014.
 - 85 Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410, 2016. doi:10.1109/DSAA.2016.49.
 - 86 Heiko Paulheim and Christian Bizer. Type inference on noisy rdf data. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, pages 510–525, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-41335-3_32.
 - 87 Jinghui Peng, Xinyu Hu, Wenbo Huang, and Jian Yang. What is a multi-modal knowledge graph: A survey. *Big Data Research*, 32:100380, 2023. doi:10.1016/j.bdr.2023.100380.
 - 88 Matthew Perry. Tontogen: A synthetic data set generator for semantic web applications. *AIS SIGSEMIS Bulletin*, 2005. URL: <https://corescholar.libraries.wright.edu/knoesis/977>.
 - 89 Axel Polleres, Romana Pernisch, Angela Bonifati, Daniele Dell’Aglio, Daniil Dobriy, Stefania Dumbrava, Lorena Etcheverry, Nicolas Ferranti, Katja Hose, Ernesto Jiménez-Ruiz, Matteo Lisandrini, Ansgar Scherp, Riccardo Tommasini, and Johannes Wachs. How Does Knowledge Evolve in Open Knowledge Graphs? *Transactions on Graph Data and Knowledge*, 1(1):11:1–11:59, 2023. doi:10.4230/TGDK.1.1.11.
 - 90 Jan Portisch and Heiko Paulheim. The dlcc node classification benchmark for analyzing knowledge graph embeddings. In Ulrike Sattler, Aidan Hogan, Maria Keet, Valentina Presutti, João Paulo A. Almeida, Hideaki Takeda, Pierre Monnin, Giuseppe Pirrò, and Claudia d’Amato, editors, *The Semantic Web – ISWC 2022*, pages 592–609, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-19433-7_34.
 - 91 Shi Qiao and Z. Meral Özsoyoglu. Rbench: Application-specific rdf benchmarking. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD ’15*, pages 1825–1838, New York, NY, USA, 2015. Association for Computing Machinery. doi: 10.1145/2723372.2746479.
 - 92 Tilmann Rabl, Michael Frank, Hatem Mousselly Sergieh, and Harald Kosch. A data generator for cloud-scale benchmarking. In Raghunath Nambiar and Meikel Poess, editors, *Performance Evaluation, Measurement and Characterization of Complex Systems*, pages 41–56, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
 - 93 Tanguy Raynaud, Samir Amir, and Rafiqul Haque. A generic and high-performance rdf instance generator. *International Journal of Web Engineering and Technology*, 11(2):133–152, 2016. doi: 10.1504/IJWET.2016.077342.
 - 94 Ehud Reiter. Building natural-language generation systems. *CoRR*, cmp-lg/9605002, 1996. arXiv:cmp-lg/9605002.
 - 95 Michael Röder, Denis Kuchelev, and Axel-Cyrille Ngonga Ngomo. HOBbit: A platform for benchmarking big linked data. *Data Sci.*, 3(1):15–35, 2020. doi:10.3233/DS-190021.
 - 96 Michael Röder, Geraldo de Souza, Denis Kuchelev, Abdelmoneim Amer Desouki, and Axel-Cyrille Ngonga Ngomo. Orca - a benchmark for data web crawlers. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pages 272–279, 2021. doi:10.1109/ICSC50631.2021.00054.
 - 97 Michael Röder, Pham Thuy Sy Nguyen, Felix Conrads, Ana Alexandra Morim da Silva, and Axel-Cyrille Ngonga Ngomo. Lemming – example-based mimicking of knowledge graphs. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pages 62–69, 2021. doi:10.1109/ICSC50631.2021.00015.
 - 98 Lukas Schmelzeisen, Corina Dima, and Stefan Staab. Wikidated 1.0: An evolving knowledge graph dataset of wikidata’s revision history. In Lucie-Aimée Kaffee, Simon Razniewski, and Aidan Hogan, editors, *Proceedings of the 2nd Wikidata Workshop (Wikidata 2021) co-located with the 20th International Semantic Web Conference (ISWC 2021), Virtual Conference, October 24, 2021*, CEUR Workshop Proceedings. CEUR-WS.org, 2021. URL: <https://ceur-ws.org/Vol-2982/paper-11.pdf>.
 - 99 Michael Schmidt, Thomas Hornung, Michael Meier, Christoph Pinkel, and Georg Lausen. *SP2Bench: A SPARQL Performance Benchmark*,

- pages 371–393. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-04329-1_16.
- 100 Mohamed Ahmed Sherif, Ana Alexandra Morim da Silva, Svetlana Pestryakova, Abdullah Fathi Ahmed, Sven Niemann, and Axel-Cyrille Ngonga Ngomo. IngridKG: A FAIR Knowledge Graph of Graffiti. *Scientific Data*, 2023. URL: https://papers.dice-research.org/2023/NSDJ_IngridKG/public.pdf.
 - 101 William Shiao, Benjamin A. Miller, Kevin Chan, Paul Yu, Tina Eliassi-Rad, and Evangelos E. Papalexakis. Tengan: adversarially generating multiplex tensor graphs. *Data Mining and Knowledge Discovery*, 38(1):1–21, 2024. doi:10.1007/s10618-023-00947-3.
 - 102 Gunjan Singh, Udit Arora, Shashikant Kumar, Riccardo Tommasini, Pieter Bonte, Sumit Bhatia, and Raghava Mutharaju. Genact: An ontology-based temporal web data generator. In Wolfgang Maass, Hyoil Han, Hasan Yasar, and Nick Multari, editors, *Conceptual Modeling*, pages 317–336, Cham, 2025. Springer Nature Switzerland.
 - 103 Gunjan Singh, Sumit Bhatia, and Raghava Mutharaju. Owl2bench: A benchmark for owl 2 reasoners. In Jeff Z. Pan, Valentina Tamma, Claudia d’Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web – ISWC 2020*, pages 81–96, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-62466-8_6.
 - 104 Mirko Spasic, Milos Jovanovic, and Arnau Prat-Pérez. An rdf dataset generator for the social network benchmark with real-world coherence. In *BLINK@ ISWC*, pages 18–25, 2016. URL: <http://ceur-ws.org/Vol-1700/#paper-02>.
 - 105 René Speck, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo. Twitter network mimicking for data storage benchmarking. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pages 298–305, 2021. doi:10.1109/ICSC50631.2021.00057.
 - 106 Fahim Sufi. Addressing data scarcity in the medical domain: A gpt-based approach for synthetic data generation and feature extraction. *Information*, 15(5), 2024. doi:10.3390/info15050264.
 - 107 Tangina Sultana and Young-Koo Lee. Expressive rule pattern based compression with ranking in horn rules on rdf style kb. In *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 13–19, 2021. doi:10.1109/BigComp51126.2021.00012.
 - 108 Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró. The train benchmark: cross-technology performance evaluation of continuous model queries. *Softw. Syst. Model.*, 17(4):1365–1393, 2018. doi:10.1007/S10270-016-0571-8.
 - 109 Ruben Taelman, Pieter Colpaert, Erik Mannens, and Ruben Verborgh. Generating public transport data based on population distributions for rdf benchmarking. *Semantic Web*, 10(2):305–328, 2019. doi:10.3233/SW-180319.
 - 110 Wenzhuo Tang, Haitao Mao, Danial Dervovic, Ivan Brugere, Saumitra Mishra, Yuying Xie, and Jiliang Tang. Cross-domain graph data scaling: A showcase with diffusion models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, pages 123619–123648. Curran Associates, Inc., 2025. URL: <https://openreview.net/pdf?id=0UGbZOQAXi>.
 - 111 Samir Tartir, I Budak Arpinar, Michael Moore, Amit P Sheth, and Boanerges Aleman-Meza. Ontoqa: Metric-based ontology quality analysis. In *IEEE ICDM Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 2005. URL: <https://corescholar.libraries.wright.edu/knoesis/660>.
 - 112 Christoph Tempich and Raphael Volz. Towards a benchmark for semantic web reasoners - an analysis of the daml ontology library. In *Workshop on Evaluation of Ontology-based Tools*, 2003.
 - 113 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. doi:10.48550/arXiv.2302.13971.
 - 114 Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, volume 104, pages 41–50, 2004. URL: <https://ceur-ws.org/Vol-104/05Tsarkov-final.pdf>.
 - 115 Yannis Tzitzikas, Christina Lantzaki, and Dimitris Zeginis. Blank node matching and rdf/s comparison functions. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web – ISWC 2012*, pages 591–607, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-35176-1_37.
 - 116 Wilco Van Leeuwen, George H.L. Fletcher, Nikolay Yakovets, and Angela Bonifati. Stability notions in synthetic graph generation: A preliminary study. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, volume 2017-March, pages 486–489, 2017. doi:10.5441/002/edbt.2017.51.
 - 117 Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014. doi:10.1145/2629489.
 - 118 Hui Wang, Xin Wang, Menglu Ma, and Yiheng You. Rpqbench: A benchmark for regular path queries on graph data. In Mahmoud Barhamgi, Hua Wang, and Xin Wang, editors, *Web Information Systems Engineering – WISE 2024*, pages 351–367, Singapore, 2025. Springer Nature Singapore.
 - 119 Meng Wang, Haofen Wang, Guilin Qi, and Qishuo Zheng. Richpedia: A large-scale, comprehensive multi-modal knowledge graph. *Big Data Research*, 22:100159, 2020. doi:10.1016/j.bdr.2020.100159.

- 120 Sui-Yu Wang, Yuanbo Guo, Abir Qasem, and Jeff Hefflin. Rapid benchmarking for semantic web knowledge base systems. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *The Semantic Web – ISWC 2005*, pages 758–772, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11574620_54.
- 121 Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’ networks. *nature*, 393(6684):440–442, 1998.
- 122 Jack Waudby, Benjamin A. Steer, Arnau Prat-Pérez, and Gábor Szárnyas. Supporting dynamic graphs and temporal entity deletions in the ldbc social network benchmark’s data generator. In *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, GRADES-NDA’20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3398682.3399165.
- 123 Tanja Wiehn. Synthetic data: From data scarcity to data pollution. *Surveillance & Society*, 22(4):472–476, 2024.
- 124 Jinhong Wu, Konstantinos Plataniotis, Lucy Liu, Ehsan Amjadi, and Yuri Lawryshyn. Interpretation for variational autoencoder used to generate financial synthetic tabular data. *Algorithms*, 16(2), 2023. doi:10.3390/a16020121.
- 125 Zenglin Xu, Feng Yan, and Yuan Qi. Bayesian nonparametric models for multiway data analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):475–487, 2015. doi:10.1109/TPAMI.2013.201.
- 126 Rui Yang, Yingying Zheng, Lei Tang, Wensheng Dou, Wei Wang, and Jun Wei. Randomized differential testing of rdf stores. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 136–140, 2023. doi:10.1109/ICSE-Companion58688.2023.00041.
- 127 Jinsung Yoon, James Jordon, and Mihaela van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2019. URL: <https://openreview.net/forum?id=S1zk9iRqF7>.
- 128 Amrapali Zaveri, Dimitris Kontokostas, Mohamed A. Sherif, Lorenz Bühmann, Mohamed Morsey, Sören Auer, and Jens Lehmann. User-driven quality evaluation of dbpedia. In *Proceedings of the 9th International Conference on Semantic Systems, I-SEMANTICS ’13*, pages 97–104, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2506182.2506195.
- 129 Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. *ACM Trans. Database Syst.*, 42(4), October 2017. doi:10.1145/3134428.
- 130 Zhigang Zhao, Xiong Luo, Maojian Chen, and Ling Ma. A survey of knowledge graph construction using machine learning. *CMES - Computer Modeling in Engineering and Sciences*, 139(1):225–257, 2023. doi:10.32604/cmes.2023.031513.
- 131 Lingfeng Zhong, Jia Wu, Qian Li, Hao Peng, and Xindong Wu. A comprehensive survey on automatic knowledge graph construction. *ACM Comput. Surv.*, 56(4), November 2023. doi:10.1145/3618295.
- 132 Xixi Zhu, Bin Liu, Li Yao, Zhaoyun Ding, and Cheng Zhu. TGR: neural-symbolic ontological reasoner for domain-specific knowledge graphs. *Appl. Intell.*, 53(20):23946–23965, 2023. doi:10.1007/S10489-023-04834-8.
- 133 Yanqiao Zhu, Yuanqi Du, Yinkai Wang, Yichen Xu, Jieyu Zhang, Qiang Liu, and Shu Wu. A survey on deep graph generation: Methods and applications. In Bastian Rieck and Razvan Pascanu, editors, *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pages 47:1–47:21. PMLR, 09–12 December 2022. URL: <https://proceedings.mlr.press/v198/zhu22a.html>.