A Simplex Approach to Synthetic Knowledge Graph Generation

Ana Alexandra Morim da Silva
Atul Bhopalsing Pundir
ana.silva@uni-paderborn.de
Data Science Group (DICE)
Heinz Nixdorf Institute
Paderborn University
Paderborn, Germany

Michael Röder
Axel-Cyrille Ngonga Ngomo
axel.ngonga@upb.de
Data Science Group (DICE)
Heinz Nixdorf Institute
Paderborn University
Paderborn, Germany

Abstract

The growing scale of knowledge graphs demands scalable systems for their subsequent processing. However, accurate benchmarking requires large knowledge graphs. While data-driven synthetic generators based on versioned datasets are promising to generate large realistic graphs, current approaches generate the graph at a triple level without considering higher-order structures. This work introduces SimplexKG, a simplex-based synthetic knowledge graph generator. Our approach analyzes d-dimensional simplices within input knowledge graphs and leverages the identified simplicial networks to generate a synthetic graph of arbitrary size. We explore whether leveraging higher-dimensional structures enhances the realism of synthetic graphs by evaluating the structure and the utility of the generated graphs. Our approach consistently outperforms 2 baseline generators and 6 variants of the state-of-the-art generator LEMMING in structural fidelity and triple store benchmarking scenarios across 3 datasets. Specifically, compared to the second-best approach, our graphs achieve a structuredness value up to 26.62% closer to the target graph, while reducing the query throughput error by up to 6.59% across storage solutions.

CCS Concepts

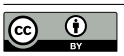
• **Information systems** → **Resource Description Framework (RDF)**; • **Networks** → *Topology analysis and generation.*

Keywords

Knowledge Graphs; Synthetic Knowledge Graphs; Synthetic Data Generation; Triple Store Benchmarking

ACM Reference Format:
Ana Alexandra Morim da Silva, Atul Bhopalsing Pundir, Michael Röder, and Axel-Cyrille Ngonga Ngomo. 2026. A Simplex Approach to Synthetic Knowledge Graph Generation. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3774904.3792566

Resource Availability:
The source code of this paper has been made publicly available at https://doi.org/10.5281/zenodo.18302828 and https://github.com/dice-group/Lemming.



WWW '26, Dubai, United Arab Emirates

ACM ISBN 979-8-4007-2307-0/2026/04
https://doi.org/10.1145/3774904.3792566

1 Introduction

The increasing variety, complexity, and capacity of Knowledge Graph (KG)-processing systems lead to an increased demand for scalability benchmarking. However, data grows too rapidly for benchmarks to remain valid for an extended period. While synthetic data generation addresses this limitation, it often produces clean graphs that do not approximate real-world workloads [30] with fixed graph properties (e.g., fixed degree distributions). Data-driven KG generation addresses these concerns [20, 32] by mirroring the structure of real-world graphs. A variant within this paradigm is version-aware generation [40], which leverages successive releases of the same dataset as structural blueprints, e.g., Linked Geo Data's periodic releases [43]. By generating synthetic graphs from previous versions of a dataset, the generated graph inherits a stable structure, as random fluctuations in a single version are smoothed out. Consequently, the generated graph preserves the average structure of input graphs without retaining any real-world information. These approaches focus on generating the graph at an assertion level, rather than considering higher-order structures. Simplices [15, 21]—sets of interconnected n vertices—can model multi-entity interactions and can capture higher-order relationships present in graphs [38]. To the best of our knowledge, simplicial modeling remains unexplored for KG generation.

We propose SimplexKG, a version-aware data-driven KG generator that leverages observed simplices and their connectivity pattern in different versions of a real-world KG. Furthermore, SimplexKG generates a KGs of arbitrary size, while retaining the structure of the original KGs. We hold out the latest version of each dataset as the ground-truth graph, and compare the generated graphs against it. We evaluate the generated graphs based on three dimensions: generation runtime, graph structure, and downstream task utility. For the downstream study, we benchmark five triple stores on both synthetic graphs and real target graphs through the SPARQL benchmarking platform IGUANA [13], following query workloads generated by Sparqloscope [4]. Within our evaluation, SimplexKG achieved a higher structural similarity and lower error scores in query throughput than two baseline generators and six variants of the state-of-the-art generator LEMMING [40]. It also achieved a lower generation runtime than LEMMING in all datasets. We summarize our contributions as follows:

- We propose SimplexKG, the first KG generator to leverage simplices.
- We perform a comprehensive evaluation based on generation runtime, graph structure, and downstream task utility.

- We present empirical evidence that SimplexKG yields graphs closer to real releases than existing generators.
- Our experiments corroborate that simplex-modeling in KG generation improves the realism of synthetic KGs.
- We release the full source code and generated datasets.

2 Related work

Synthetic data has been used across multiple fields for its proxy learning and benchmarking potential. In machine learning, synthetic data is used to train models that require access to sensitive information, e.g., medical datasets, that later predict based on real-world data [6]. In graph neural networks, synthetic graphs are used to benchmark different architectures of graph neural networks, e.g., GraphWorld [34]. As such, generating graphs with real-world properties is a vital research direction.

Watts-Strogatz [47] introduced a model that generates graphs with small diameters and a high clustering coefficient. Barabási-Albert [2] introduced a generator based on preferential attachment, where new vertices are attached to already well-connected vertices. Darwini [23] mimics the degree distribution and the local clustering coefficient from an example graph. Kronecker [31] generates real-world graphs with small-world properties, power-law degree distribution, and community structure. More recently, there have been approaches that leverage simplex structures to generate general graphs [5, 8, 14, 15, 46, 48, 49]. Wang et al. [46] propose a parametric model based on the dimension of a simplex. The graph is generated by combining a d-dimensional simplex with a complete graph, where a simplex is added to each edge of the complete graph and connected to all the vertices in the added simplex. Bianconi and Rahmede [8] introduce a model that relies on simplex-based distributions and takes two parameters: the simplex dimension d and a network geometry flavor f. The flavor variable describes the simplex space and dictates the evolution over time of the algorithm. Courtney and Bianconi [15] introduce a model that generates both directed and undirected simplicial complexes. The graph starts with a 2-dimensional simplex. Afterward, the model selects a new source node through the Pitman-Yor process [3, 27, 35] and selects the target node randomly from the graph. If the simplex to which the new edge forms already exists, this is reflected in the weights of the existing simplex. Despite simplices having been leveraged extensively for general graphs, they have not been applied to generate knowledge graphs.

General graph generators generate graphs with specific properties that real-world graphs have been shown to follow, i.e., specific degree distributions, clustering coefficients, community structures, and small diameters [11, 31]. However, they do not consider KG-specific properties [26, 45]. Several synthetic KG generators have been proposed that mimic KG-specific properties, but are specific to a schema [1, 10, 24, 28, 30, 42, 44]. On the other hand, domain-agnostic approaches have been introduced and are not bound to a single schema [29, 34, 36]. Thus reflecting the current state and needs of the KG community. However, schema-driven approaches still require the manual configuration of properties. KG-mimickers mine the properties from real-world input graphs, and replicate these in a synthetic KG. RBench [37] generates a graph of arbitrary size, similar to an input graph, requiring a size scaling factor and a degree scaling factor. However, the degree scaling factor cannot be inferred from a single input graph. LEMMING [40] produces synthetic KGs of arbitrary size given several versions of an input dataset. The first version of the graph is generated from statistics collected from the set of input graphs. LEMMING then optimizes this initial version based on learned graph-specific invariant arithmetic expressions. LEMMING is the only KG-generator that can process multiple versions of the same dataset and mimic their characteristics. However, LEMMING operates only at an assertion level. In this work, we explore whether modeling simplices in version-aware KG generation improves the realism of synthetic KGs.

3 Preliminaries

Knowledge Graph. Let $\mathbb{R}$, $\mathbb{B}$, and $\mathbb{L}$ be mutually disjoint sets of all RDF resources, blank nodes, and RDF literals, respectively. Furthermore, let $\mathbb{A}$ be the set of all RDF predicates with $\mathbb{A} \subset \mathbb{R}$. An RDF KG G is a set of triples of the form $(s, p, o) \in (\mathbb{R} \cup \mathbb{B}) \times \mathbb{A} \times (\mathbb{R} \cup \mathbb{B} \cup \mathbb{L})$ [40]. A set of KGs is denoted by $\mathcal{G} = \{G_1, ..., G_N\}$ with N elements.

Directed edge-labeled graphs. A directed edge-labeled graph representing an RDF KG is defined as $G^* = (V, E)$ with a vertex set $V \subseteq \mathbb{I} \cup \mathbb{B} \cup \mathbb{L}$ and an edge set $E \subseteq V \times \mathbb{A} \times V$ representing the directed edges [39]. Let V_{G^*} and E_{G^*} denote the sets of all vertices and edges of a corresponding graph G^*, respectively. A vertex $v \in V$ is isolated if $\forall (k, r, t) \in E : k \neq v \vee t \neq v$, and connected otherwise.

Simplex. A d-simplex, $\phi_{d,G} = \{ n_i \mid n_i \in V_G, i = 0, 1, ..., d \}$ for $0 \leq d < |V_G|$, is a set of $d + 1$ nodes interconnected with one another [7]. Its *1-skeleton* representation is a complete graph of d nodes. Given α as the function that maps a node to its respective set of classes, $\alpha : V \mapsto 2^C$, and C as the set of classes existing in G, the respective class d-simplex is defined as $\kappa(\phi_{d,G}) = \{ \alpha(n_i) \mid n_i \in \phi_{d,G}, i = 0, ..., d \}$. The faces of a d-simplex are all the lower-order simplices of order $0 \leq \delta < d$ formed from the proper subset of nodes of the d-simplex, $F_\delta(\phi_{d,G}) = \{\phi_{\delta,G} \mid 0 \leq \delta < d \wedge \phi_{\delta,G} \subset \phi_{d,G}\}$. In the case of a 2-simplex, the faces would be the set of 1 and 0-simplices that form it, i.e., the 3 links for $\delta = 1$ and the 3 vertices for $\delta = 0$. A d-dimensional simplicial network is a set of connected d-simplices– that is, simplices that share one of its faces. A d-simplex is isolated otherwise.

4 Approach

Our approach first identifies the simplices of dimensions up to d and their interactions within a set of input graphs. It then analyzes the simplices and their interactions to estimate the underlying probability distributions, which are conditioned on the classes of the involved nodes and the properties of the associated edges. SimplexKG generates a new output graph by leveraging the estimated distributions. Finally, the graph is converted to an RDF KG that mirrors the structure of the input graphs.

4.1 Locating simplices

We preprocess each graph by extracting all edges with the property `rdf:type` from the input graphs $\mathcal{G}$. We retain the class information by annotating each node in the graph with its corresponding set of classes. After this step, the graph contains only relationships

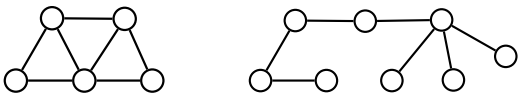
Simplex Types	Set	Example Fig.
Networks of d-simplices	S_{d,G_i}	1
Isolated d-simplices	I_{d,G_i}	2
Equal-dimension link	X_{d,G_i}	3a
Cross-dimension link	U_{d,G_i}	3b
Dangling simplices	D_{d,G_i}	4

Table 1: Types of identified simplices.

between instances, ensuring the subsequent analyses focus on instance assertions rather than class definitions. We then identify two types of simplices: connected simplices and isolated simplices. The main objective of this step is to locate sub-graphs representing d-dimensional simplicial networks and simplices that interact with these networks. For each graph $G_i \in \mathcal{G}$, we classify the simplex interactions in five categories:

In this work, we consider simplices of dimensions $d \leq 2$.

4.1.1 Find simplices of $d \leq 2$. We first identify the higher-dimensional simplices in all input graphs. For each graph $G_i \in \mathcal{G}$, we distinguish between connected d-simplices, stored in Y_{d,G_i} where $1 \leq d$ and isolated d-simplices I_{d,G_i} for $0 \leq d$. The set of connected d-simplices includes d-dimensional networks and their interactions: networks of d-simplices S_{d,G_i}, simplex links U_{d,G_i}, network links X_{d,G_i}, and dangling simplices D_{d,G_i}. The isolated simplices are identified from their non-existent neighbourhood. To find the 2-simplices in an input graph, we adapt the node triangle iterator algorithm from [41] as presented in the appendix A.2. Figure 1 illustrates networks of d-simplices.



(a) $d = 2$ **(b)** $d = 1$

Figure 1: Networks of d-simplices.

Once the 2-simplices and their connections are identified, S_{1,G_i} and I_{1,G_i} are populated with the remaining 1-simplices. The set of 0-simplices, I_{0,G_i}, contains all isolated nodes for each input graph $G_i \in \mathcal{G}$. Figure 1 illustrates isolated d-simplices.

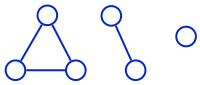
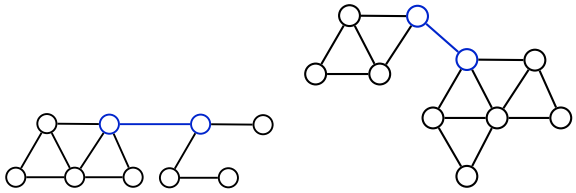


Figure 2: Isolated d-simplices for $d = \{0, 1, 2\}$.

4.1.2 Simplex links. We consider two types of simplex links: one between simplices of equal dimensions and a cross-dimensional link. As the maximum dimension is $d_{max} = 2$, linking entails the creation of 1-simplices between an existing node of a simplex and another existing node of another simplex. Links between simplices of equal dimensions are stored in U_{d,G_i} between simplices in S_{d,G_i} for a given d. While links between simplices of equal dimensions



(a) Cross-dimension link. **(b) Equal-dimension link.**

Figure 3: Simplex interactions.

are between simplices in S_{d,G_i} for a given d, cross-dimensional links are links between a simplex in S_{1,G_i} and another simplex in S_{2,G_i}. Equal-dimension links are stored in U_{d,G_i} and cross-dimension links in X_{d,G_i}. Figure 3 exemplifies interactions between these simplices.

4.1.3 Dangling simplices. Dangling simplices are represented by a-simplices connected to d-simplices where $a < d$. For each graph $G_i \in \mathcal{G}$, the dangling simplices are stored in D_{d,G_i}. Figure 4 illustrates three dangling 1-simplices connected to a 2-simplex.

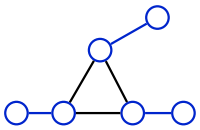


Figure 4: Three dangling 1-simplices connected to a 2-simplex.

4.2 Analyzing the simplices

After simplex identification, we analyze the structural and semantic characteristics of the identified simplices. We estimate the output graph's density and compute probability distributions for classes and properties. This analysis is done for all applicable simplex sets from Step 4.1.

4.2.1 Node and edge density. Let $\mathcal{N}(\mathcal{Z})$ and $\mathcal{L}(\mathcal{Z})$ be the functions that retrieve the number of unique nodes and the number of unique edges in a simplex set $\mathcal{Z}_{d,G} \in \{S_{d,G}, U_{d,G}, X_{d,G}, D_{d,G}, I_{d,G}\}$. We estimate the number of nodes $\vartheta_{\mathcal{Z}}$ and the number of edges $\xi_{\mathcal{Z}}$ that each previously populated simplex set should have in the output graph by considering the desired number of nodes in the output graph $|V_{G_{N+1}}|$ and the average node and edge density over all input graphs. The node, $\vartheta_{\mathcal{Z}}$, and edge, $\xi_{\mathcal{Z}}$, estimates are defined as

$$\vartheta_{\mathcal{Z}_{d,G_i}} = |V_{G_{N+1}}| \times \frac{1}{N} \sum_{i=1}^{N} \frac{\mathcal{L}(\mathcal{Z}_{d,G_i})}{|V_{G_i}|}, \tag{1}$$

$$\xi_{\mathcal{Z}_{d,G_i}} = |V_{G_{N+1}}| \times \frac{1}{N} \sum_{i=1}^{N} \frac{\mathcal{N}(\mathcal{Z}_{d,G_i})}{|V_{G_i}|}. \tag{2}$$

4.2.2 Class distribution. We compute class distributions for $d = \{0, 1, 2\}$, separately. For 2-simplices, this operation involves modeling the distribution of the classes of all 3 nodes, $\mathbb{P}(C_1, C_2, C_3)$, for the

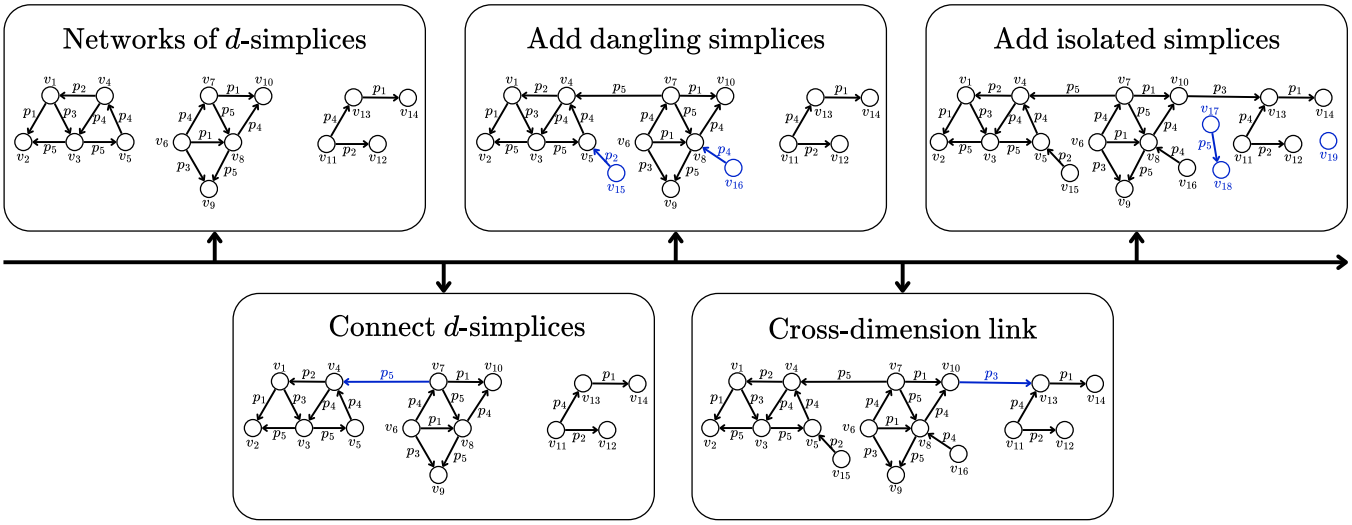


Figure 5: Example of the graph generation process.

sets of 2-simplices S_{2,G_i} and I_{2,G_i} on each input graph $G_i \in \mathcal{G}$.[1] Let $\mathcal{E}(\phi)$ be a function that retrieves the number of edges that connect 3 nodes of classes $\kappa(\phi)$. $\mathcal{Y}_{\mathcal{Z}_{d,G_i}}(\kappa(\phi_{d,G_i}))$ is a function that returns the edge density of $\kappa(\phi)$ and is defined as

$$\mathcal{Y}_{\mathcal{Z}_{d,G_i}}(\kappa(\phi_{d,G_i})) = \frac{|\{\mathcal{E}(\phi') \mid \phi' \in \mathcal{Z}_{d,G_i} \wedge \kappa(\phi') = \kappa(\phi_{d,G_i})\}|}{|\{\phi'' \mid \phi'' \in \mathcal{Z}_{d,G_i} \wedge \kappa(\phi'') = \kappa(\phi_{d,G_i})\}|}. \tag{3}$$

We compute the class probability for a set of classes (C_1, C_2, C_3) as defined by

$$\mathbb{P}_{\mathcal{Z}_{2,G_i}}(C_1, C_2, C_3) = \frac{1}{N}\sum_{i=1}^{N} \frac{\mathcal{Y}_{\mathcal{Z}_{2,G_i}}(\{C_1, C_2, C_3\})}{\sum_{\forall \phi' \in \mathcal{Z}_{2,G_i}} \mathcal{Y}_{\mathcal{Z}_{2,G_i}}(\kappa(\phi'))}. \tag{4}$$

For 1-simplices, we compute the probability of the classes of the 2 nodes $\mathbb{P}_{\mathcal{Z}}(C_1, C_2)$ defined as

$$\mathbb{P}_{\mathcal{Z}}(C_1, C_2) = \frac{1}{N}\sum_{i=1}^{N} \frac{|\{\phi \mid \phi \in \mathcal{Z}_{1,G_i} \wedge \kappa(\phi) = \{C_1, C_2\}\}|}{|\mathcal{Z}_{1,G_i}|}. \tag{5}$$

Similarly, this probability is also computed for all possible class combinations found in 1-simplex sets, S_{1,G_i}, X_{1,G_i}, and D_{1,G_i}.

For 0-simplices, we compute the probability of a class $\mathbb{P}(C)$ as shown in Equation 6. As 0-simplices represent nodes, $\mathbb{P}(C)$ is only computed for the set of isolated 0-simplices I_{0,G_i}, defined by

$$\mathbb{P}(C) = \frac{1}{N}\sum_{i=1}^{N} \frac{|\{\phi \mid \phi \in I_{0,G_i} \wedge \kappa(\phi) = \{C\}\}|}{|I_{0,G_i}|}. \tag{6}$$

4.2.3 Class distribution for simplex faces. We compute the class distributions for $(d-1)$ faces of d-simplices, required for connected 2-simplices and 1-simplices. For 2-simplices, it means we compute the probabilities of classes C_1 and C_2 for a 1-simplex present in a 2-simplex for every possible pair of distinct classes, as defined in

$$\mathbb{P}(C_1, C_2) = \frac{1}{N}\sum_{i=1}^{N} \frac{\left|\{\phi_1 | \kappa(\phi_1) = \{C_1, C_2\} \wedge \exists \phi_2 \in S_{2,G_i} : \phi_1 \in F_1(\phi_2)\}\right|}{3|S_{2,G_i}|}. \tag{7}$$

For 1-simplices, we compute the probability for every possible head class $\mathbb{P}(C)$ defined by

$$\mathbb{P}_{\mathcal{Z}_{1,G_i}}(C) = \frac{1}{N}\sum_{i=1}^{N} \frac{|\{\phi \mid \phi \in \mathcal{Z}_{1,G_i} \wedge \kappa_h(\phi) = C\}|}{|\mathcal{Z}_{1,G_i}|}, \tag{8}$$

with κ_h representing the classes of the head node of a 1-simplex.

4.2.4 Conditional distribution. We compute the conditional distribution given classes of $(d-1)$ faces. This is required to sample classes for a node, given classes of a face, which is essential for network growth. For example, if we are growing a 2-simplex network, it involves adding a new node and connecting it to 2 existing nodes. Let Γ_d denote the set of all class permutations of d-simplices. The conditional distribution allows us to sample the class of the new node, given the classes of the 2 other existing nodes in the network, as defined by

$$\mathbb{P}(C_3|(C_1, C_2)) = \frac{\mathbb{P}_S(C_1, C_2, C_3)}{\sum_{(C_1,C_2,C')\in\Gamma_2} \mathbb{P}_S(C_1, C_2, C')}. \tag{9}$$

For 1-simplices, this entails computing the probability of a tail class C_t given a class C_h defined as

$$\mathbb{P}_{\mathcal{Z}}(C_t|C_h) = \frac{\mathbb{P}_{\mathcal{Z}}(C_t, C_h)}{\sum_{(C',C_h)\in\Gamma_1} \mathbb{P}_{\mathcal{Z}}(C', C_h)}. \tag{10}$$

4.2.5 Property distribution. Finally, we analyze property distributions conditioned on the classes of the tail and head nodes, as shown in Equation 11. For this reason, we need to collect the triples representing d-simplices in the form of (C_t, p, C_h). Let γ be a function that retrieves all triples from the KG that represent a simplex set $\mathcal{Z}$. We denote the set of retrieved triples as $\gamma(\mathcal{Z})$, where $\gamma(\mathcal{Z}) \subseteq G_i$.

[1] Within our implementation, we use a canonical form to reduce the number of possible combinations of C_1, C_2, and C_3. This reduces the memory requirement and improves the generation runtime. However, we do not include this canonical form in the description of our approach since this optimization is not in the focus of our work.

The probability of a property r connecting a node of class C_t with a node of class C_h is defined as

$$\mathbb{P}_{\mathcal{Z}}(r|(C_t, C_h)) = \frac{1}{N}\sum_{i=1}^{N} \frac{|\{(s,p,o) \in \mathcal{T}_{d,i}(C_h, C_t) : p = r\}|}{|\mathcal{T}_{d,i}(C_h, C_t)|}, \quad (11)$$

where

$$\mathcal{T}_{d,i}(C_h, C_t) = \{(s,p,o) \in \gamma(\mathcal{Z}_{d,i}) : \alpha(s) = C_h \wedge \alpha(o) = C_t\}. \quad (12)$$

The property distributions are computed for all of the 1 and 2-simplex sets previously identified in Step 4.1.

4.3 Generating the graph

We generate the KG incrementally using the distributions computed in the analysis phase. The graph generation process is exemplified in Figure 5. The approach prioritizes the generation of higher-dimensional simplices and integrates the simplex interactions analyzed in Step 4.1:

(1) Initialize local networks for $d = \{1, 2\}$,
(2) Connect 2-simplices via 1-simplices,
(3) Add dangling 1-simplices to 2-simplices,
(4) Connect 2-simplices with 1-simplices via 1-simplices, and
(5) Add isolated simplices for $d = \{0, 1, 2\}$.

4.3.1 Local networks of d-simplices. We adapt the algorithm from [15] to generate directed edge-labeled graphs that consider edge properties and node classes, as presented in Algorithm 1. When creating a new 2-simplex, we first sample the classes for each node. Then, for each edge in the simplex, we assign a direction based on semantic constraints by checking valid tail and head class pairs. This means that we check which tail and head class pairs from the classes of the three nodes are valid based on the input graphs $\mathcal{G}$. The edge directions are assigned accordingly. This process addresses the limitation of fixed-direction assignments, especially in directed d-simplices, as edge directions in knowledge graphs must reflect semantic relationships between nodes. Another distinction from [15] is that we base our algorithm on distributions from the input graphs, instead of fixed distributions. The function `evaluateDistribution` evaluates whether the computed distribution would validate a newly proposed class. We generate networks iteratively by considering 2 cases. We either expand the network by proposing a new node and attaching it to an existing simplex's face, or if that is no longer possible, we create a new simplex. Consequently, this algorithm forms local networks, instead of one network. This process stops when the estimated edge counts for each simplex type are reached. We also consider a maximum number of retries in the simplex expansion step when the newly proposed simplex is not a valid one. In this case, we retry the operation by sampling a different simplex.

Similarly, the network of connected 1-simplices is also generated by Algorithm 1. However, the number of nodes in 1-simplices is determined by:

$$\vartheta_{1,S} = |V_{G_{N+1}}| - \left(\sum_{d=0}^{2} \vartheta_{d,I} + \vartheta_{2,S} + \vartheta_{1,D}\right). \quad (13)$$

This ensures consistency in the output graph, as it ensures that the output graph always has $|V_{G_{N+1}}|$ vertices.

Algorithm 1 Generating networks of d-simplices based on [15].

Input: Estimated node count $\vartheta_{d,S}$ and edge count $\xi_{d,S}$
  Sample classes for the nodes of a new d-simplex (ϕ_d)
  Choose properties for every pair of classes in $\kappa(\phi_d)$
  Create nodes for each $C \in \kappa(\phi^d)$
  Add edges with determined properties to form a simplex ϕ_d
  Add ϕ_d to $S_{d,N+1}$
  Initialize edge g_e and node count g_n
  while $g_e < \xi_{d,S}$ **do**
    if $g_n < \vartheta_{d,S}$ **then**
      Select an existing face of a simplex $\phi_{d-1} : \exists \phi_d \in S_{d,n+1} \wedge \phi_{d-1} \in F_{d-1}(\phi_d)$
      Find set of classes assigned to face nodes $\kappa(\phi^{d-1})$
      Propose new classes C_n for new node
      if `evaluateDistribution`$(\kappa(\phi_{d-1}), C_n)$ **then**
        Create a new node q with classes C_n
        Create a new simplex ϕ'_d by connecting ϕ_{d-1} to q
        Assign properties to new edges
        Add ϕ'_d to $S_{d,N+1}$
        Update g_e and g_n
      else
        Create a new d-simplex ϕ_d
        Sample classes for the nodes to nodes of ϕ_d
        Assign properties to edges of ϕ_d
        Add ϕ_d to $S_{d,N+1}$
        Update g_e and g_n
      end if
    else
      Select an existing simplex $\phi_d \in S_{d,N+1}$
      Find set of classes $\kappa(\phi_d)$ assigned to nodes of ϕ_d
      Determine property r for any pair in $\kappa(\phi_d)$
      Add an edge with property r
      Add ϕ_d to $S_{d,N+1}$
      Update g_e
    end if
  end while

4.3.2 Connect d-simplices. In this step, we link 2-simplices via 1-simplices. To do this, we first sample a pair of head and tail classes. We then randomly select 2 simplices such that one contains at least one node of the head class and the other contains at least one node of the tail class. From each simplex, we choose one node that matches the criteria. Finally, we connect these nodes with a 1-simplex and assign a property to the edge according to the head and tail classes. This process repeats until $\xi_{1,U}$ edges are added, prioritizing connections between isolated simplices. For 1-simplices between connected 2-simplices, we verify that no additional 2-simplices are formed during this operation. This means that the two selected nodes should have no common neighbour before the connection. Creating additional 2-simplices would otherwise invalidate the estimated counts established in the analysis step.

4.3.3 Add dangling simplices. We create lower-order simplices and connect them to higher-dimensional simplices. First, we propose head and tail classes for the new 1-simplex, then we search for an appropriate candidate 2-simplex from $S_{2,G_{N+1}}$ to connect with,

Table 2: Proposed types of simplex generators.

Generator	Classes	Properties
BPBC	$\mathcal{B}$	$\mathcal{B}$
BPUC	$\mathcal{U}$	$\mathcal{B}$
UPBC	$\mathcal{B}$	$\mathcal{U}$
UPUC	$\mathcal{U}$	$\mathcal{U}$

which has at least a node that matches either the proposed tail or head class. The property is then sampled for the given head and tail classes and used to create the 1-simplex. This step creates $\vartheta_{1,D}$ nodes.

4.3.4 Connect local networks. In this step, $\xi_{1,X}$ edges are created to connect local networks in $S_{1,G_{N+1}}$ and $S_{2,G_{N+1}}$. First, the head and tail classes are proposed. The 2 candidate simplices are selected, one from each set. The nodes are then selected, and the edge is formed with the sampled property.

4.3.5 Add isolated simplices. We create isolated d-simplices of $d = \{0, 1, 2\}$. For 1 and 2-simplices, new simplices are created until the number of nodes $\vartheta_{d,G_{N+1}}$ and the number of edges $\xi_{d,G_{N+1}}$ have been reached. Once the number of nodes is reached, but the number of edges is still not satisfied, edges are added between already connected nodes.

4.4 Finalize graph

The graph is finalized as in LEMMING [40]. This involves materializing the `rdf:type` triples, generating literal nodes, and generating URI representations for the resources.

4.5 Generator types

We propose 4 types of generators based on different sampling strategies of classes and properties as presented in Table 2. $\mathcal{U}$ represents that a feature was sampled from a uniform distribution, and $\mathcal{B}$ from the computed distribution of the input graphs $\mathcal{G}$ as estimated in Step 4.2.

5 Evaluation

We evaluate the runtime of our generators, the structural fidelity of the created graphs, and the performance in triple store benchmarking as a downstream task. In our experiments, we always compare the performance of the synthetic graphs against that of the real-world graph.

5.1 Evaluation Setup

All the graphs were generated and benchmarked on a machine with 4x Intel Xeon Platinum 8462Y+ and 32 GB RAM. As all the approaches rely on sampling, the results reported below are the average of three runs for each respective configuration.

5.1.1 Other Approaches. We compare our performance against 6 generators from LEMMING [40], as it is the only existing data-driven graph generator that accepts multiple versions of a KG as input. All other data-driven generators are designed for a single static graph and thus address a different use case. The 6 variants from LEMMING are UCS-UIS, UCS-BIS, CCS-UIS, CCS-BIS, BCS-UIS, and BCS-BIS. We also compare our system's performance against 2 baseline generators based on the Barabási-Albert (BL-BA) [2] and Watts-Strogatz (BL-WS) [47] models. We use LEMMING's implementation of the Barabási-Albert model for RDF KGs. We also adapted the Watts-Strogatz model to generate a KG based on the average node degree of $\mathcal{G}$ and with the desired number of vertices $|V_{G_{N+1}}|$. For a fair comparison, the same graph preprocessing and finalization stages were used on the baseline graphs as in SimplexKG and LEMMING. In both baselines, the `rdf:triples` were assigned such that the average class distribution from $\mathcal{G}$ was preserved in the output graph.

Table 3: Dataset statistics of the target graph.

Feature	Datasets		
	SWDF	LGD	ICC
Resources	45,221	591,237	1,243
Edges (with `rdf:type`)	325,896	4,224,003	8,757
Edges	181,967	583,257	5,499
Node triangles	19,260	0	4,279
Edge triangles	66,594	0	47,680
Max. in-degree	816	6,195	197
Max. out-degree	621	1,850	170
Std. dev. in-degree	12.15	9.48	11.38
Std. dev. out-degree	10.07	6.55	11.43

5.1.2 Datasets. We re-use the datasets of Röder et al. [40]: a subset of Linked Geo Data (LGD) [43] from 2013 to 2015, Semantic Web Dog Food (SWDF) [33] from 2001 to 2015, and the International Chronostratigraphic Chart (ICC) [16–19] datasets from 2004 to 2018. The latest version of each graph was held out from the graph generation process and used as a target performance in the experiments. The dataset statistics of the target graphs without class assertions are presented in Table 3.

5.2 Runtime

The summary of the graph generation runtimes is presented in Table 4. BL-BA is the fastest approach across all 3 datasets, which is expected since it operates as a general graph generator and does not have the added complexity of handling semantics of KG. On average, the simplex approaches are 43.15% faster than the six variants of LEMMING for LGD (the largest dataset). For ICC, the smallest dataset, they are 26.68% faster, but 18.94% slower for the SWDF dataset. Among the simplex variants, BPBC is faster for the SWDF dataset, BPUC for the LGD dataset, and UPBC for the ICC dataset.

5.3 Graph Structure

The structural fidelity is measured by comparing the structuredness and the overall graph characteristics of the synthetic graphs against those of the target graph.

5.3.1 Structuredness. We compute the Mean Absolute Error (MAE) between the structuredness [22] of the synthetic and the target

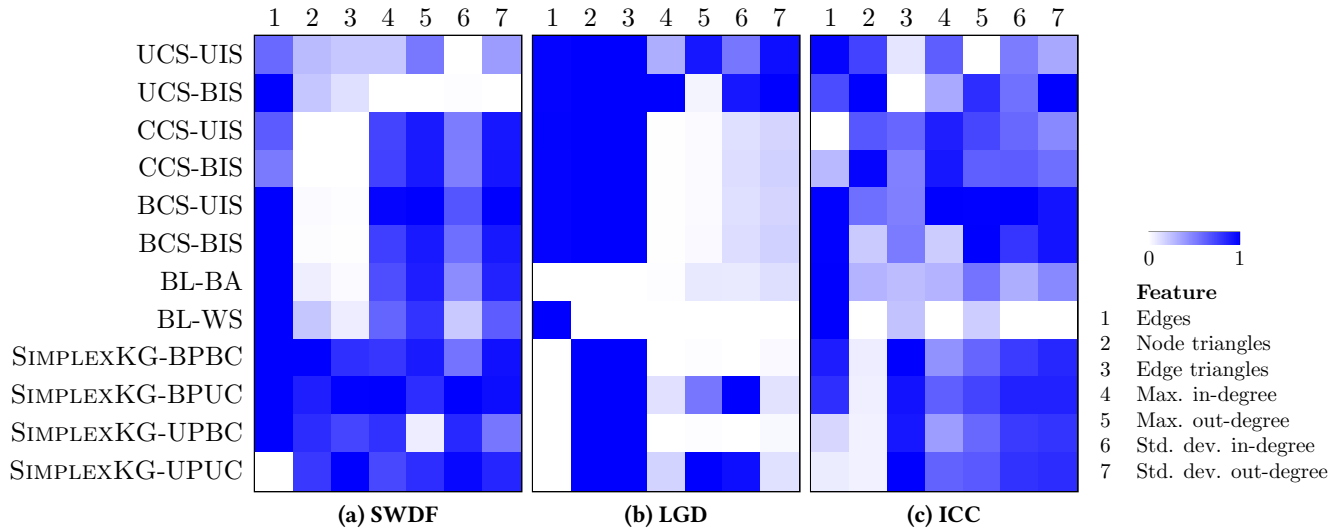


Figure 6: Comparison of graph characteristics for different datasets. Darker colours represent a higher similarity.

graph. We present the results in Table 5, where lower values indicate higher structural similarity. As expected, the baseline graphs (BL-BA and BL-WS) show the highest difference to the target's structuredness across all datasets. Our simplex-based approaches consistently show a smaller divergence than all other approaches. Among our approaches, the UPBC and UPUC show a higher standard deviation specifically for the LGD dataset. We hypothesize that the increased standard deviation produced by UPBC and UPUC on the LGD dataset stems from the lack of 2-simplices in the LGD input graphs, leading to degeneration when the properties are selected at random.

5.3.2 Graph Properties. We compare seven different graph properties: 1. number of edges, 2. number of node triangles, 3. number of edge triangles, 4. maximum in-degree, 5. maximum out-degree, and 6. standard deviation of the in- and 7. out-degree. The graph properties were evaluated on graphs without class assertions. We disregard the number of vertices, since all the approaches ensure the generated graphs have the same number of vertices as the target graph. We establish the target graph similarity by first computing the absolute difference between each of the feature counts and the target graph, and normalizing this score. The summary of the similarity scores is illustrated by the heatmaps in Figure 6. The rows in the heatmaps represent the different approaches, and each column represents a feature. The darker the cell colour, the higher the similarity in the corresponding feature for a given approach to the target graph. Thus, the darker a row is, the more similar that approach is to the target graph. The baseline generators performed the poorest on the LGD and the ICC datasets, while LEMMING's

Table 4: Runtime in seconds reported by the approaches. The values in bold represent the lowest runtimes for the respective dataset.

Approach		SWDF	LGD	ICC
LEMMING [40]	UCS-UIS	195	1,645	35
	UCS-BIS	236	10,527	32
	CCS-UIS	141	1,902	76
	CCS-BIS	175	15,277	100
	BCS-UIS	83	1,844	10
	BCS-BIS	110	15,048	21
	BL-BA	**9**	**130**	**0**
	BL-WS	37	1,322	**0**
SimplexKG	BPBC	160	1,422	12
	BPUC	193	514	29
	UPBC	216	1,804	10
	UPUC	236	720	24

Table 5: Structuredness difference between generated graphs and target graph. Bold values indicate the best performance (mean ± standard deviation) in each dataset column.

Approach		SWDF	LGD	ICC
LEMMING [40]	UCS-UIS	0.108 ± 0.006	0.111 ± 0.012	0.244 ± 0.007
	UCS-BIS	0.107 ± 0.004	0.141 ± 0.029	0.261 ± 0.003
	CCS-UIS	0.088 ± 0.004	0.105 ± 0.001	0.246 ± 0.028
	CCS-BIS	0.094 ± 0.003	0.105 ± 0.002	0.236 ± 0.016
	BCS-UIS	0.088 ± 0.004	0.115 ± 0.010	0.237 ± 0.023
	BCS-BIS	0.090 ± 0.001	0.104 ± 0.004	0.246 ± 0.009
	BL-BA	0.122 ± 0.001	0.212 ± 0.002	0.358 ± 0.004
	BL-WS	0.120 ± 0.000	0.209 ± 0.001	0.354 ± 0.005
SimplexKG	BPBC	0.080 ± 0.001	0.083 ± 0.002	**0.184 ± 0.002**
	BPUC	0.086 ± 0.003	**0.047 ± 0.002**	0.279 ± 0.007
	UPBC	**0.067 ± 0.006**	0.033 ± 0.239	0.236 ± 0.016
	UPUC	0.079 ± 0.006	0.143 ± 0.201	0.315 ± 0.010

Table 6: QPS Root Mean Squared Error (RMSE) of the synthetic graphs for the datasets SWDF, LGD, and ICC. Values in bold represent the best performance, and underlined values represent the second-best performance.

Approach		SWDF	LGD	ICC
LEMMING [40]	UCS-UIS	48.057	47.577	258.936
	UCS-BIS	51.494	42.077	272.191
	CCS-UIS	46.420	46.500	280.580
	CCS-BIS	<u>44.670</u>	47.808	264.217
	BCS-UIS	47.816	41.586	<u>251.107</u>
	BCS-BIS	46.931	<u>37.415</u>	263.641
	BL-AB	67.839	39.042	285.227
	BL-WS	56.197	39.925	302.377
SimplexKG	BPBC	**40.197**	**32.065**	**244.759**
	BPUC	46.046	208.129	328.510
	UPBC	51.551	388.246	263.716
	UPUC	54.743	403.705	335.592

UCS-BIS performed the poorest on the SWDF dataset. For the SWDF dataset, SimplexKG demonstrated the highest similarity across all features. Especially the BPBC and BPUC generators successfully approximate the node and edge triangles—something that the other approaches failed to do. For the LGD dataset, LEMMING's UCS generators are the most similar, likely because SimplexKG, lacking 2-simplices in the dataset, were restricted to modeling 1-simplices networks and lost their leverage of modeling higher-order structures. The LGD dataset's very low node degree may further impact approaches that operate beyond the assertion level. For the ICC dataset, the SimplexKG generators are consistent and competitive across all features, though slightly outperformed by BCS-UIS. While other approaches' performance varied significantly in each dataset, SimplexKG demonstrated the most stable performance overall.

5.4 Triple Store Benchmarking

In our third experiment, we test whether benchmarking a system, e.g., a triple store, with a generated graph leads to similar results as with the real-world target graph. We use Sparqloscope [4] to generate a set of queries for each dataset. We use the same query templates to generate the set of queries for all datasets. Sparqloscope generates 67 queries for ICC, and 69 queries for LGD and SWDF datasets. The benchmarking experiments were conducted using IGUANA [13] version 4.1.1 with 100 query mixes. The following triple stores were benchmarked: Tentris 0.19.5+beta [9], GraphDB 11.0.0, Virtuoso Open Source Edition 7.2.15 [25], Apache Jena Fuseki 5.5.0 [12], and Blazegraph 2.1.6. [2] The performance is measured through the Queries per Second (QPS) metric, which measures the query throughput of each triple store. We compute the RMSE between the achieved QPS by the synthetic graph and the QPS of the target graph on all triple stores. The RMSE scores are present in Table 6. The average QPS values per triple store are presented in the Appendix Tables 7, 8, and 9. For the interested reader, we also defer the Normalized RMSE (NRMSE) scores to the Appendix Table 10. SimplexKG-BPBC consistently outperforms the other approaches, achieving the lowest RMSE in all three datasets. [3] While the other SimplexKG generators have a comparable performance on the SWDF and the ICC datasets, they experience a significant performance degradation on the LGD dataset. We attribute this to the graph's low degree and absence of higher-order structures, which amplify errors when the properties and/or the classes are uniformly sampled. LEMMING's optimization of synthetic graphs, based on invariant arithmetic expressions, can also be applied to graphs generated by SimplexKG. The experiments on the optimized graphs are presented in Appendix C.

6 Findings

The experimental results show that SimplexKG consistently outperforms the baselines and the previous state-of-the-art generators of LEMMING on the majority of the evaluated metrics. Although the baseline methods are faster, they ignore KG-specific constraints. SimplexKG is on average 17% faster than the LEMMING generators. Across all datasets, SimplexKG achieves the lowest structuredness error, producing graphs that are on average 18.71% closer to the target graph compared to the second-best approach. When measuring graph characteristics, SimplexKG achieves high similarity for the SWDF and ICC datasets. The advantage in the LGD dataset is reduced as it does not have node triangles. SimplexKG-BPBC also exhibits the lowest RMSE on QPS, reducing the error on average by 3.27% compared to the next best approach. These findings corroborate that explicitly modeling simplices in KGs leads to more realistic synthetic structures and better downstream performance overall. However, the experiments also highlight that the choice of generator should be dependent on the structure of the graph at hand. Using simplicial modeling provides limited benefits when the input graphs do not contain higher-order simplices.

7 Conclusion

We introduce SimplexKG, a simplex-based KG generator with four variants: BPBC, BPUC, UPBC, and UPUC. Our experiments demonstrate that our approaches achieve a higher structure fidelity than all other approaches, as observed in the structuredness and graph features. Furthermore, SimplexKG-BPBC also achieves the lowest RMSE in our triple store benchmarking experiments, confirming our graphs' utility for downstream tasks. While baseline methods have a lower runtime, our approaches still achieve a lower runtime than state-of-the-art approaches. Future work will focus on increasing the dimension of the modelled simplices and generating larger graphs.

Acknowledgments

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.

[2] The triple stores are available at their respective webpages: https://tentris.io/, https://graphdb.ontotext.com/, https://virtuoso.openlinksw.com/, https://jena.apache.org/, and https://blazegraph.com/.

[3] Statistical significance ($p < 0.05$) was achieved on the LGD and ICC datasets, while the SWDF dataset showed borderline significance ($p = 0.0501$) with respect to the second-best approach, as determined by paired t-tests.

References

[1] Guillaume Bagan, Angela Bonifati, Radu Ciucanu, George H. L. Fletcher, Aurélien Lemay, and Nicky Advokaat. 2017. gMark: Schema-Driven Generation of Graphs and Queries. *IEEE Transactions on Knowledge and Data Engineering* 29, 4 (2017), 856–869. doi:10.1109/TKDE.2016.2633993

[2] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[3] Bruno Bassetti, Mina Zarei, Marco Cosentino Lagomarsino, and Ginestra Bianconi. 2009. Statistical mechanics of the "Chinese restaurant process": Lack of self-averaging, anomalous finite-size effects, and condensation. *Phys. Rev. E* 80 (Dec 2009), 066118. Issue 6. doi:10.1103/PhysRevE.80.066118

[4] Hannah Bast, Johannes Kalmbach, Robin Textor-Falconi, and Christoph Ullinger. 2025. Sparqloscope: A generic benchmark for the comprehensive and concise performance evaluation of SPARQL engines. https://ad-publications.cs.uni-freiburg.de/ISWC_sparqloscope_BKTU_2025.pdf

[5] Federico Battiston, Giulia Cencetti, Iacopo Iacopini, Vito Latora, Maxime Lucas, Alice Patania, Jean-Gabriel Young, and Giovanni Petri. 2020. Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports* 874 (2020), 1–92. doi:10.1016/j.physrep.2020.05.004 Networks beyond pairwise interactions: Structure and dynamics.

[6] Brett K. Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, Ran Lee, Sanjeev P. Bhavnani, James Brian Byrd, and Casey S. Greene. 2019. Privacy-Preserving Generative Deep Neural Networks Support Clinical Data Sharing. *Circulation: Cardiovascular Quality and Outcomes* 12, 7 (2019), e005122. arXiv:https://www.ahajournals.org/doi/pdf/10.1161/CIRCOUTCOMES.118.005122 doi:10.1161/CIRCOUTCOMES.118.005122

[7] Ginestra Bianconi. 2021. *Higher-Order Networks.* Cambridge University Press.

[8] Ginestra Bianconi and Christoph Rahmede. 2016. Network geometry with flavor: From complexity to quantum geometry. *Phys. Rev. E* 93 (Mar 2016), 032315. Issue 3. doi:10.1103/PhysRevE.93.032315

[9] Alexander Bigerl, Felix Conrads, Charlotte Behning, Mohamed Ahmed Sherif, Muhammad Saleem, and Axel-Cyrille Ngonga Ngomo. 2020. Tentris – A Tensor-Based Triple Store. In *The Semantic Web – ISWC 2020*, Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal (Eds.). Springer International Publishing, Cham, 56–73.

[10] Daniel Blum and Sara Cohen. 2010. Generating RDF for Application Testing. In *Proceedings of the 2010 International Conference on Posters & Demonstrations Track - Volume 658* (Shanghai, China) *(ISWC-PD'10)*. CEUR-WS.org, Aachen, DEU, 105–108.

[11] Angela Bonifati, Irena Holubová, Arnau Prat-Pérez, and Sherif Sakr. 2020. Graph Generators: State of the Art and Open Challenges. *ACM Comput. Surv.* 53, 2, Article 36 (apr 2020), 30 pages. doi:10.1145/3379445

[12] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. 2004. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters* (New York, NY, USA) *(WWW Alt. '04)*. Association for Computing Machinery, New York, NY, USA, 74–83. doi:10.1145/1013367.1013381

[13] Felix Conrads, Jens Lehmann, Muhammad Saleem, Mohamed Morsey, and Axel-Cyrille Ngonga Ngomo. 2017. 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* (Vienna, Austria). Springer-Verlag, Berlin, Heidelberg, 48–65. doi:10.1007/978-3-319-68204-4_5

[14] Owen T. Courtney and Ginestra Bianconi. 2017. Weighted growing simplicial complexes. *Phys. Rev. E* 95 (Jun 2017), 062301. Issue 6. doi:10.1103/PhysRevE.95.062301

[15] Owen T. Courtney and Ginestra Bianconi. 2018. Dense power-law networks and simplicial complexes. *Phys. Rev. E* 97 (May 2018), 052303. Issue 5. doi:10.1103/PhysRevE.97.052303

[16] Simon Cox. 2017. *RDF representation of 2016 edition of International Chronostratigraphic Chart (Geologic Timescale) v1.* Data Collection. CSIRO.

[17] Simon Cox. 2018. *RDF representation of 2017 edition of International Chronostratigraphic Chart (Geologic Timescale) v3.* Data Collection. CSIRO.

[18] Simon Cox and Stephen Richard. 2014. *RDF representation of International Chronostratigraphic Chart (Geologic Timescale) v2.* Data Collection. CSIRO.

[19] Simon Cox and Stephen Richard. 2019. *RDF representation of 2018 edition of International Chronostratigraphic Chart (Geologic Timescale) v1.* Data Collection. CSIRO.

[20] Abdelmoneim Amer Desouki, Felix Conrads, Michael Röder, and Axel-Cyrille Ngonga Ngomo. 2021. SYNTHG: Mimicking RDF Graphs Using Tensor Factorization. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*. 76–79. doi:10.1109/ICSC50631.2021.00017

[21] Karel Devriendt and Piet Van Mieghem. 2019. The simplex geometry of graphs. *Journal of Complex Networks* 7, 4 (01 2019), 469–490. arXiv:https://academic.oup.com/comnet/article-pdf/7/4/469/29161067/cny036.pdf doi:10.1093/comnet/cny036

[22] Songyun Duan, Anastasios Kementsietsidis, Kavitha Srinivas, and Octavian Udrea. 2011. 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* (Athens, Greece) *(SIGMOD '11)*. Association for Computing Machinery, New York, NY, USA, 145–156. doi:10.1145/1989323.1989340

[23] Sergey Edunov, Dionysios Logothetis, Cheng Wang, Avery Ching, and Maja Kabiljo. 2018. Generating synthetic social graphs with darwini. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 567–577.

[24] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 619–630. doi:10.1145/2723372.2742786

[25] Orri Erling and Ivan Mikhailov. 2009. *RDF Support in the Virtuoso DBMS.* Springer Berlin Heidelberg, Berlin, Heidelberg, 7–24. doi:10.1007/978-3-642-02184-8_2

[26] Javier D Fernández, Miguel A Martínez-Prieto, Pablo de la Fuente Redondo, and Claudio Gutiérrez. 2016. Characterizing RDF datasets. *Journal of Information Science* 1 (2016), 1–27.

[27] Alexander Gnedin, Ben Hansen, and Jim Pitman. 2007. Notes on the occupancy problem with infinitely many boxes: general asymptotics and power laws. *Probability Surveys* 4, none (2007), 146 – 171. doi:10.1214/07-PS092

[28] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics* 3, 2 (2005), 158–182. doi:10.1016/j.websem.2005.06.005 Selected Papers from the International Semantic Web Conference, 2004.

[29] Nicolas Hubert, Pierre Monnin, Mathieu d'Aquin, Davy Monticolo, and Armelle Brun. 2024. PyGraft: Configurable Generation of Synthetic Schemas and Knowledge Graphs at Your Fingertips. In *The Semantic Web*, Albert Meroño Peñuela, Anastasia Dimou, Raphaël Troncy, Olaf Hartig, Maribel Acosta, Mehwish Alam, Heiko Paulheim, and Pasquale Lisena (Eds.). Springer Nature Switzerland, Cham, 3–20.

[30] Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong. 2016. LinkGen: Multipurpose Linked Data Generator. In *The Semantic Web – ISWC 2016*, Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil (Eds.). Springer International Publishing, Cham, 113–121. doi:10.1007/978-3-319-46547-0_12

[31] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker Graphs: An Approach to Modeling Networks. *Journal of Machine Learning Research* 11, 33 (2010), 985–1042. http://jmlr.org/papers/v11/leskovec10a.html

[32] André Melo and Heiko Paulheim. 2017. Synthesizing Knowledge Graphs for Link and Type Prediction Benchmarking. In *The Semantic Web*, Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig (Eds.). Springer International Publishing, Cham, 136–151.

[33] Knud Möller, Tom Heath, Siegfried Handschuh, and John Domingue. 2007. Recipes for Semantic Web Dog Food — The ESWC and ISWC Metadata Projects. In *The Semantic Web*, 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 (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 802–815.

[34] John Palowitch, Anton Tsitsulin, Bryan Perozzi, and Brandon A. Mayer. 2022. Synthetic Graph Generation to Benchmark Graph Learning. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*. https://openreview.net/forum?id=jdJtWFSC3-S

[35] Jim Pitman. 2006. *Combinatorial stochastic processes* (1 ed.). Springer Berlin, Heidelberg. doi:10.1007/b11601500

[36] Jan Portisch and Heiko Paulheim. 2022. The DLCC Node Classification Benchmark for Analyzing Knowledge Graph Embeddings. In *The Semantic Web – ISWC 2022*, Ulrike Sattler, Aidan Hogan, Maria Keet, Valentina Presutti, João Paulo A. Almeida, Hideaki Takeda, Pierre Monnin, Giuseppe Pirrò, and Claudia d'Amato (Eds.). Springer International Publishing, Cham, 592–609.

[37] Shi Qiao and Z. Meral Özsoyoğlu. 2015. RBench: Application-Specific RDF Benchmarking. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 1825–1838. doi:10.1145/2723372.2746479

[38] Ayushman Raghuvanshi, Sravanthi Gurugubelli, Hiren Madhu, and Sundeep Prabhakar Chepuri. 2024. Variational Self-Supervised Learning on Simplicial Complexes. In *2024 58th Asilomar Conference on Signals, Systems, and Computers*. 54–58. doi:10.1109/IEEECONF60004.2024.10943073

[39] Jenni Reuben. 2018. Towards a Differential Privacy Theory for Edge-Labeled Directed Graphs. 273–278 pages. doi:10.18420/sicherheit2018_24

[40] Michael Röder, Pham Thuy Sy Nguyen, Felix Conrads, Ana Alexandra Morim da Silva, and Axel-Cyrille Ngonga Ngomo. 2021. LEMMING – Example-based Mimicking of Knowledge Graphs. In *Proceedings of the 15th IEEE International Conference on Semantic Computing (ICSC)*. IEEE Computer Society, 62–69. doi:10.

1109/ICSC50631.2021.00015
[41] Thomas Schank and Dorothea Wagner. 2005. Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study. In *Experimental and Efficient Algorithms*, Sotiris E. Nikoletseas (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 606–609.
[42] Michael Schmidt, Thomas Hornung, Michael Meier, Christoph Pinkel, and Georg Lausen. 2010. SP2Bench: A SPARQL Performance Benchmark. In *Semantic Web Information Management: A Model-Based Perspective*, Roberto de Virgilio, Fausto Giunchiglia, and Letizia Tanca (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 371–393. doi:10.1007/978-3-642-04329-1_16
[43] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. 2012. LinkedGeoData: A core for a web of spatial open data. *Semant. Web* 3, 4 (Oct. 2012), 333–354.
[44] Ruben Taelman, Pieter Colpaert, Erik Mannens, and Ruben Verborgh. 2019. Generating Public Transport Data based on Population Distributions for RDF Benchmarking. *Semantic Web Journal* 10, 2 (21 Jan 2019), 305–328. doi:10.3233/SW-180319
[45] Yannis Theoharis, Yannis Tzitzikas, Dimitris Kotzinos, and Vassilis Christophides. 2008. On graph features of semantic web schemas. *IEEE Transactions on Knowledge and Data Engineering* 20, 5 (2008), 692–702.
[46] Yucheng Wang, Yuhao Yi, Wanyue Xu, and Zhongzhi Zhang. 2021. Modeling Higher-Order Interactions in Complex Networks by Edge Product of Graphs. *Comput. J.* 65, 9 (06 2021), 2347–2359. arXiv:https://academic.oup.com/comjnl/article-pdf/65/9/2347/45882135/bxab070.pdf doi:10.1093/comjnl/bxab070
[47] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.
[48] Zhihao Wu, Giulia Menichetti, Christoph Rahmede, and Ginestra Bianconi. 2015. Emergent Complex Network Geometry. *Scientific Reports* 5, 1 (2015), 10073. doi:10.1038/srep10073
[49] Tzu-Chi Yen. 2021. Construction of simplicial complexes with prescribed degree-size sequences. *Phys. Rev. E* 104 (Oct 2021), L042303. Issue 4. doi:10.1103/PhysRevE.104.L042303

A Algorithms

The adapted algorithm for finding 2-simplices is presented in Algorithm 2.

Algorithm 2 Finding 2-simplices

Input: A graph G
Output: $S_{2,G}$ and $I_{2,G}$

```
S_{2,G} ← ∅
I_{2,G} ← ∅
for v_1 in V_G do
    N_1 ← Neighbours(v_1)\Visited
    for v_2 in N_1 do
        N_2 ← Neighbours(v_2)\Visited
        for v_3 in N_2 do
            if v_2 < v_3 and v_1 ≠ v_2 and v_1 ≠ v_3 and v_3 ∈ N_2 then
                φ_{2,G} = {v_1, v_2, v_3}
                if isIsolated(φ_{2,G}) then
                    I_{2,G} ← I_{2,G} ∪ φ_{2,G}
                else
                    S_{2,G} ← S_{2,G} ∪ φ_{2,G}
                end if
            end if
        end for
    end for
    Visited ← Visited ∪ v_1
end for
```

B Experiments

We include the average QPS values achieved by each triple store on each dataset in Tables 7, 8, 9. We include the RMSE scores

Table 7: Average QPS per triple store on SWDF dataset.

	Approach	Tentris	Virtuoso	Blazegraph	Fuseki	GraphDB
	Target	423.89	253.71	22.70	48.53	108.78
LEMMING [40]	UCS-UIS	380.06	235.02	24.56	50.10	100.64
	UCS-BIS	374.56	232.67	24.43	46.24	104.94
	CCS-UIS	401.02	255.64	25.08	54.01	114.60
	CCS-BIS	408.34	251.58	24.90	54.33	114.65
	BCS-UIS	376.31	248.72	25.22	45.43	109.35
	BCS-BIS	392.10	250.25	24.86	56.30	106.01
	BL-AB	452.16	269.14	24.06	66.75	129.59
	BL-WS	433.90	264.37	25.05	56.58	121.39
SimplexKG	BPBC	403.49	242.90	24.24	51.76	109.08
	BPUC	445.68	259.23	27.59	61.98	128.46
	UPBC	396.24	230.35	23.95	48.96	110.35
	UPUC	437.52	263.69	29.35	62.21	127.35

Table 8: Average QPS per triple store on LGD dataset.

	Approach	Tentris	Virtuoso	Blazegraph	Fuseki	GraphDB
	Target	370.62	133.10	11.91	39.62	60.86
LEMMING [40]	UCS-UIS	306.07	117.34	14.00	38.18	64.24
	UCS-BIS	330.90	115.59	14.06	41.99	66.57
	CCS-UIS	310.17	117.76	14.14	39.32	65.73
	CCS-BIS	300.91	125.32	14.19	39.57	62.57
	BCS-UIS	323.33	120.85	14.41	42.88	69.78
	BCS-BIS	330.07	121.15	14.13	42.27	68.23
	BL-AB	336.17	121.90	14.58	43.33	70.04
	BL-WS	334.61	124.53	14.53	41.87	66.57
SimplexKG	BPBC	342.01	124.35	14.16	41.82	68.33
	BPUC	734.64	208.61	31.27	103.27	201.93
	UPBC	729.64	376.76	33.68	178.99	316.31
	UPUC	1076.38	406.86	39.99	169.91	341.54

Table 9: Average QPS per triple store on ICC dataset.

	Approach	Tentris	Virtuoso	Blazegraph	Fuseki	GraphDB
	Target	1535.94	773.96	65.23	282.43	410.41
LEMMING [40]	UCS-UIS	1730.77	929.59	74.05	387.75	570.31
	UCS-BIS	1784.44	921.05	74.44	385.27	594.58
	CCS-UIS	1910.34	974.73	75.56	423.88	663.55
	CCS-BIS	1837.84	949.39	73.76	398.32	618.92
	BCS-UIS	1788.53	921.29	74.40	385.28	610.79
	BCS-BIS	1742.77	923.24	73.28	379.37	608.33
	BL-AB	1699.44	933.24	74.36	386.27	621.57
	BL-WS	1774.11	934.52	74.11	398.66	619.67
SimplexKG	BPBC	1757.86	943.55	74.96	386.56	614.51
	BPUC	1975.33	946.09	75.35	432.20	682.94
	UPBC	1759.05	949.34	75.14	391.51	600.83
	UPUC	2004.31	948.12	75.73	422.20	686.21

Table 10: QPS NRMSE of the generated graphs.

Approaches	SWDF	LGD	ICC
UCS-UIS	0.282	0.293	0.263
UCS-BIS	0.356	0.382	0.274
CCS-UIS	0.324	0.247	0.300
CCS-BIS	0.323	**0.241**	0.278
BCS-UIS	0.305	0.248	0.266
BCS-BIS	0.313	0.244	0.272
BL-AB	0.260	0.315	0.284
BL-WS	0.287	0.434	0.303
SimplexKG-BPBC	**0.213**	0.305	**0.261**
SimplexKG-BPUC	0.296	15.032	0.356
SimplexKG-UPBC	0.236	21.180	0.278
SimplexKG-UPUC	0.347	40.123	0.361

Table 11: Structuredness MAE between optimized synthetic graphs and target graph.

Approach		SWDF	LGD	ICC
LEMMING [40]	UCS-UIS	0.103 ± 0.033	0.140 ± 0.081	0.234 ± 0.130
	UCS-BIS	0.106 ± 0.032	0.123 ± 0.077	0.240 ± 0.123
	CCS-UIS	0.090 ± 0.043	0.104 ± 0.092	0.209 ± 0.137
	CCS-BIS	0.095 ± 0.039	0.105 ± 0.092	0.209 ± 0.140
	BCS-UIS	0.095 ± 0.041	0.105 ± 0.089	0.209 ± 0.139
	BCS-BIS	0.096 ± 0.040	0.105 ± 0.092	0.212 ± 0.136
SimplexKG	BPBC	0.086 ± 0.046	0.106 ± 0.099	**0.158 ± 0.169**
	BPUC	0.085 ± 0.044	0.049 ± 0.125	0.245 ± 0.117
	UPBC	0.072 ± 0.054	**0.032 ± 0.134**	0.185 ± 0.148
	UPUC	**0.074 ± 0.050**	0.143 ± 0.070	0.273 ± 0.100

normalized by the target graph's QPS values in Table 10. We observe the lowest NRMSE values for the SWDF and the ICC dataset.

C Optimization

LEMMING introduces another feature for synthetic KG generation: optimization of the synthetic graphs based on invariant arithmetic expressions. We investigate here whether the best-performing approach changes when optimized based on the same expressions.

C.1 Graph Structure

The summary of the structuredness results on the optimized graphs is presented in Table 11. On average, across all approaches, the MAE reduced in the ICC dataset, and remained constant after optimization in the SWDF and the LGD datasets.

The summary of the graph properties similarity is presented in Figure 7. We can observe that in most cases the optimization is beneficial. SimplexKG approximates the node and edge triangles in all three datasets. In SWDF and ICC datasets, SimplexKG produces the most similar graphs, as observed by the darkest rows in the heatmap.

C.2 Triple Store Benchmarking

The summary of the RMSE and NRMSE scores of the synthetic graphs after undergoing optimization is presented in Table 12. The NRMSE scores are normalized by the target graph's QPS range for a specific query. In general, the RMSE on the optimized graphs decreased for the LGD and ICC datasets and increased for the SWDF dataset. This is observed for the majority of approaches. SimplexKG-BPBC has the lowest RMSE for the LGD and ICC dataset, and the second lowest RMSE on the SWDF. These findings further corroborate our hypothesis, as SimplexKG remains the approach with the lowest error rate.

C.3 Configuration

The configuration file for LEMMING's optimization included the following metrics:

- Number of vertices ($\#v$),
- Number of edges ($\#e$),
- Average degree ($\hat{\tau}$),
- Number of node triangles (Δ_n),
- Number of edge triangles (Δ_e),
- Maximum node in-degree ($m(\tau_i)$),
- Maximum node out-degree ($m(\tau_o)$),
- In-degree standard deviation ($\sigma(\tau_i)$), and
- Out-degree standard deviation ($\sigma(\tau_o)$).

We configured LEMMING to generate 5 invariant arithmetic expressions per dataset and 50,000 optimization steps. The invariant expressions for the SWDF dataset were:

$$\beth_1 = ((\Delta_n - \#v)/((\#e + \sigma(\tau_o)) * (\hat{\tau} + 1.0))), \tag{14}$$
$$\beth_2 = ((\Delta_n - \#v)/((\#e + \sigma(\tau_i)) * (\hat{\tau} + 1.0))), \tag{15}$$
$$\beth_3 = (((\Delta_n - (\sigma(\tau_o) * \Delta_e)) - \#v)/(\#e * \hat{\tau})), \tag{16}$$
$$\beth_4 = ((\Delta_n - \#v)/(((\sigma(\tau_i) - \#e) - \sigma(\tau_o)) * \hat{\tau})), \tag{17}$$
$$\beth_5 = ((\Delta_n - \#v)/((\hat{\tau} - \#e) * (\hat{\tau} + 1.0))) . \tag{18}$$

The invariant expressions for the LGD dataset were:

$$\beth_6 = ((\#v - (\Delta_e + \sigma(\tau_o)))/(\#e * (\hat{\tau} + 1.0))), \tag{19}$$
$$\beth_7 = ((\#v - \Delta_e)/((\#e + m(\tau_o)) * (\hat{\tau} + 1.0))), \tag{20}$$
$$\beth_8 = (((\Delta_e - \#v)/\hat{\tau})/(\#e - (\Delta_n/\hat{\tau}))), \tag{21}$$
$$\beth_9 = (((\#v - \sigma(\tau_i)) - \Delta_e)/(\#e * (\hat{\tau} + 1.0))), \tag{22}$$
$$\beth_{10} = ((((\Delta_e + (\hat{\tau} * m(\tau_i))) - \#v)/\hat{\tau})/\#e) . \tag{23}$$

The invariant expressions for the ICC dataset were:

$$\beth_{11} = ((m(\tau_o) + \#v)/((\sigma(\tau_i) - (\#v/m(\tau_o))) + m(\tau_o))), \tag{24}$$
$$\beth_{12} = (((m(\tau_o) + (\hat{\tau}/m(\tau_i))) + \#v)/((\sigma(\tau_i) - \#v) + m(\tau_o))), \tag{25}$$
$$\beth_{13} = ((m(\tau_o) + (\#v - \sigma(\tau_o)))/((\sigma(\tau_i) - \#v) + m(\tau_o))), \tag{26}$$
$$\beth_{14} = (((m(\tau_i)/m(\tau_o)) + \sigma(\tau_o))/((\sigma(\tau_i) - \#v) + m(\tau_o))), \tag{27}$$
$$\beth_{15} = ((m(\tau_o) + (\#v/m(\tau_i)))/((\sigma(\tau_i) - (\#v/m(\tau_o))) + m(\tau_o))) . \tag{28}$$

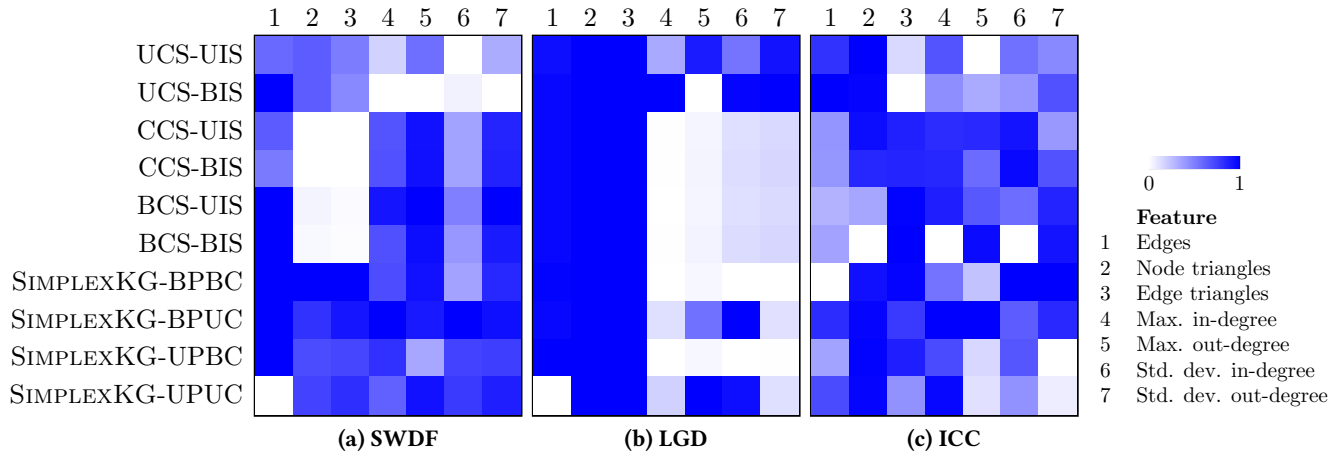


Figure 7: Comparison of graph characteristics for different datasets. Darker colours represent a higher similarity.

Table 12: QPS RMSE and NRMSE of the optimized graphs for the datasets SWDF, LGD, and ICC.

Approach	SWDF		LGD		ICC	
	QPS NRMSE	QPS RMSE	QPS NRMSE	QPS RMSE	QPS NRMSE	QPS RMSE
UCS-UIS	0.323	**50.048**	0.291	44.887	0.270	261.012
UCS-BIS	0.401	53.119	0.377	40.084	0.278	270.774
CCS-UIS	0.365	56.846	**0.241**	39.340	0.251	245.618
CCS-BIS	0.354	52.864	0.246	38.820	0.260	252.877
BCS-UIS	0.398	54.155	0.244	39.939	0.266	254.314
BCS-BIS	0.387	54.517	0.249	34.555	0.258	255.401
SimplexKG-BPBC	**0.249**	51.711	0.317	**29.734**	**0.237**	**233.279**
SimplexKG-BPUC	0.345	60.885	15.809	224.158	0.355	328.171
SimplexKG-UPBC	0.289	63.769	22.780	409.194	0.244	242.706
SimplexKG-UPUC	0.389	67.811	43.601	433.666	0.318	300.330