arXiv:2307.06698v3 [cs.AI] 25 Aug 2023

IntelliGraphs: Datasets for Benchmarking Knowledge Graph Generation

Thiviyan Thanapalasingam
University of Amsterdam
`thiviyan.t@gmail.com`

Emile van Krieken
Vrije Universiteit Amsterdam

Peter Bloem
Vrije Universiteit Amsterdam

Paul Groth
University of Amsterdam

Abstract

Knowledge Graph Embedding (KGE) models are used to learn continuous representations of entities and relations. A key task in the literature is predicting missing links between entities. However, Knowledge Graphs are not just sets of links but also have semantics underlying their structure. Semantics is crucial in several downstream tasks, such as query answering or reasoning. We introduce the *subgraph inference task*, where a model has to generate likely and semantically valid subgraphs. We propose *IntelliGraphs*, a set of five new Knowledge Graph datasets. The IntelliGraphs datasets contain subgraphs with semantics expressed in logical rules for evaluating subgraph inference. We also present the dataset generator that produced the synthetic datasets. We designed four novel baseline models, which include three models based on traditional KGEs. We evaluate their expressiveness and show that these models cannot capture the semantics. We believe this benchmark will encourage the development of machine learning models that emphasize semantic understanding.

1 Introduction

Knowledge Graphs (KGs) contain knowledge about the world structured as graphs with entities connected through different relations [Hogan et al., 2021]. Large-scale KGs are widely used in a range of applications, such as query answering [Arakelyan et al., 2020] and information retrieval [Noy et al., 2019].

To address the problem of incompleteness in KGs, Knowledge Graph Embedding (KGE) models were developed. These learn continuous representations for entities and relations [Bordes et al., 2013, Yang et al., 2014] through *link prediction*, the task of predicting missing links in large KGs, by learning scoring functions that rank entities [Ruffinelli et al., 2019]. These approaches implicitly assume that each link (also known as a *triple*) in a Knowledge Graph can be predicted *independently*. In this view of Knowledge Graphs, each triple is seen as a kind of "atomic fact" which is true or false independent of other triples.

However, in modern Knowledge Graphs, triples depend on each other. For example, the triples `value(temperature_NY, 77)` and `unit(temperature_NY, Fahrenheit)` together describe that the temperature in New York is 77 °F. In this case, the truth of the first triple depends on the content of the second. Figure 1 provides a more complex example which represents subgraph

Preprint. Under review.

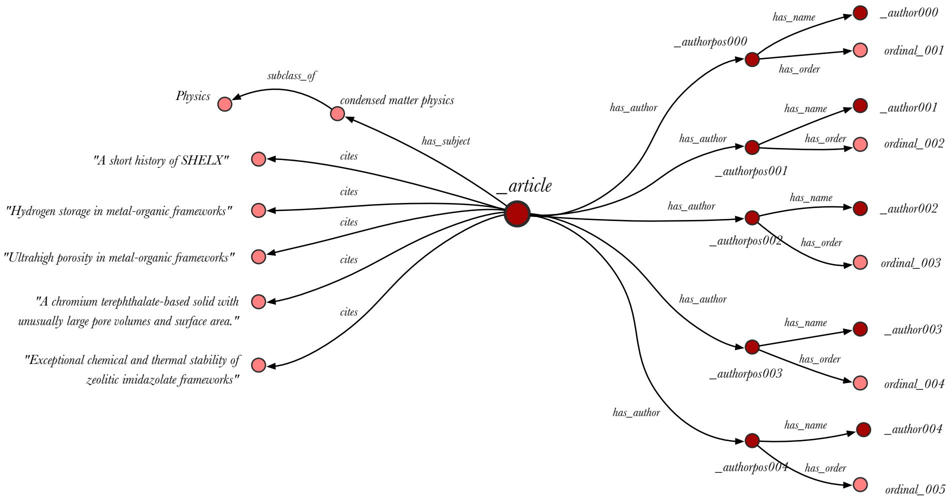

Figure 1: A subgraph about a research article representing the complex interdepencies between the relevant entities. This example was extracted from Wikidata. The the lighter nodes represent entities and darker nodes represent existential nodes (explained in Section 3).

about a research article. In our work, we are interested in subgraphs that have many interdependent facts.

Existing KGE models cannot capture such interdependencies between triples [Wen et al., 2016]. In this paper, we introduce a new task that can be used to evaluate models that generate several connected triples together, modelling their interdependencies. We call this task *subgraph inference*. The idea is that where KGE models can be used to predict missing links, subgraph inference models could be used to predict missing subgraphs: sets of interdependent links.

To simplify the problem of subgraph inference, we assume that a set of true subgraphs is provided so that the problem reduces to training a generative model on small knowledge graphs over a shared set of entities and relations.[1]Such predictions must not only capture the general structure of the graph, but they must also allow us to generalize effectively to graphs not explicitly shown in the data.

So far, the lack of datasets with well-understood semantics has hampered studying how effectively KGE models capture semantics. Existing datasets commonly used for benchmarking KGE models, such as FB15k-237 and WN18RR, lack sufficient logical constraints to investigate semantics thoroughly. Logical constraints play an important role as they help maintain the logical consistency of facts in a structured knowledge base.

Thus, the three main contributions of our work are as follows:

1. **Subgraph Inference.** We define a new task, where the goal is to generate, from a set of examples, novel subgraphs that follow certain logical rules. We specified new evaluation metrics that help empirically assess generated graphs' semantic validity and novelty.
2. **IntelliGraphs**
   (a) **Synthetic Datasets.** We propose three synthetic datasets, each designed to capture different levels of semantics. We also describe the underlying semantics using First-Order Logic.
   (b) **Real-world Datasets.** We extract subgraphs from Wikidata according to simple basic patterns to generate two *real-world* datasets.[2]
3. **Data Generator.** We developed a Python package that randomly generates and verifies subgraphs using pre-defined logical constraints.

[1] That is, for a true generalization of link prediction to subgraph prediction, we would provide a single, large knowledge graph and require a model to predict missing subgraphs. In the interest of separating concerns, we ask here only if generative models over small knowledge graphs are feasible.

[2] https://www.wikidata.org/

The datasets and generators are publicly available on: `https://github.com/thiviyanT/IntelliGraphs`. The generator is available as a Python package which can be installed through PyPy, and Conda package managers. [3] To ensure long-term preservation and easy access, we made the datasets available on Zenodo. [4]

2 Benchmark Tasks

In this Section, we discuss related benchmark tasks and their limitations, and then, we introduce a novel research task we call *subgraph inference*.

2.1 Limitations of Link Predictors

Binary relations KGE models exploit structural regularities to perform Knowledge Graph completion. The last decade has seen the developments of several KGE models [Ruffinelli et al., 2019], which predict the likelihood that a pair of entities are related by a given binary relation. However, a set of binary relations cannot represent an N-*ary* relation because the links depend on each other. Regardless of the context, KGE models assign a set of probabilities on links, and those probabilities are independent of each other.

N-*ary* relations Link prediction has been extended to cover N-*ary* relations, where the goal is to predict a missing link in an N-*ary* fact. N-*ary* relation can operate on any arbitrary number of entities. Modelling N-*ary* relations as triples and treating them as entities in binary relations results in a loss of structural information [Wen et al., 2016]. Wen et al. [2016] define N-*ary* relations as the mappings from the attribute sequences to the attribute values, such that each N-*ary* fact is an instance of the corresponding N-*ary* relation. GRAN is a graph-based approach which uses a Transformer decoder to score N-*ary* facts [Wang et al., 2021]. NeuInfer uses fully-connected neural networks to embed N-*ary* relations and score candidate triples [Guan et al., 2020]. These models were evaluated by inferring an element in an N-*ary* fact. Because a single N-ary relation can be represented in a set of binary relations (i.e. triples), subgraphs can be used to represent N-ary relations. This means that Subgraph models could be used to solve N-ary relation prediction, but the task is strictly broader than that: every single N-ary relation can be represented as a subgraph, but not every class of subgraphs can be naturally captured by a single n-ary relation.

Link prediction evaluation The standard link prediction evaluation framework [Ruffinelli et al., 2019] uses ranking-based evaluation metrics, such as Hits@k and Mean Reciprocal Rank, which do not explicitly check for the semantics of the predicted links. Instead, the evaluation protocol assumes that the underlying semantics can be indirectly validated if a missing link has been correctly predicted. In our work, we set out to *explicitly* check the semantics of newly generated subgraphs.

2.2 Subgraph Inference

A *Knowledge Graph*, G, is a tuple $G = (V, E, \mathcal{E}, \mathcal{R}, L)$. E is a set of edges where $E = V \times \mathcal{R} \times V$ and $\mathcal{R}$ is the set of relations. V is set of nodes drawn from the set of possible entities $\mathcal{E}$ in G. L is a set of functions that define the semantics of G by determining which structures are permissible or not in G.

Given a Knowledge Graph G, we call a *subgraph* F a tuple $\left(V^f, E^f, \mathcal{R}\right)$ where $V^f \subset V$ and $E^f = \left\{(u, r, v) \mid u \in V^f, v \in V^f\right\}$, $r \in \mathcal{R}$ and $(u, r, v) \in E$. We require subgraphs to be connected graphs. Every subgraph complies with the semantics of the Knowledge Graph, L_G.

For example, using the example from the introduction, the statement “The temperature is 77°F.” can be expressed as a simple two-triple subgraph F of some larger graph with the triples `value(temperature, 77)`, `unit(temperature, Fahrenheit)`.[5]The meaning of the entity `temperature` is dependent on both triples. Notably, the unit “Fahrenheit” gives value

[3] PyPy: `https://pypi.org/project/intelligraphs` & Conda:`https://anaconda.org/thiv/intelligraphs`

[4] `https://doi.org/10.5281/zenodo.7824818`

“77” additional context. Here, the semantics could be captured by a function that checks that `is_instance(value, integer)` $\wedge$ `is_unit(unit, units_of_temperature)`.

Problem Statement *Subgraph Inference* is the task of inferring missing subgraphs given a set of existing subgraphs from a Knowledge Graph, G.[6]The inferred subgraphs must adhere to the semantics of the original KG. We define the task as follows: Given a set of known subgraphs S_G^k from a given Knowledge Graph G, infer missing subgraphs S_G^m that comply with the same logical constraints of the initial Knowledge Graph L_G. We assume we have access to L_G. The model does not have access to L_G during learning. Indstead, it is used to evaluate the semantics of the model output during evaluation.

These subgraphs can be added back to the KG; therefore, this task can be seen as an extension of link prediction. To make this extension complete, we should also specify how the training subgraphs are extracted from G. However, to isolate the question of generative modelling of knowledge graphs, we take this process as given in our tasks. For instance, in the two real-world datasets, we extract subgraphs from Wikidata according to a hand-designed pattern. In the synthetic datasets, we simply provide a set of small knowledge graphs over a shared set of entities and relations, leaving the larger graph G entirely implicit. With this choice, the task reduces to training a generative model over small knowledge graphs with a shared set of entities and relations.

Key Challenge. The subgraphs need to adhere to specific semantics which, in a learning setting, have to be inferred from a limited set of examples, such as learning the types of entities.

3 IntelliGraphs

Motivated by the aforementioned limitations of link prediction datasets and the new task of subgraph inference, we introduce five new benchmark datasets where each dataset tests different semantics. Table 1 shows key statistics about the synthetic and real-world graphs. The appendix (see Section 7.3.1) describes the algorithm used to generate the datasets.

Data Generator The sampler D samples subgraphs according to a probability distribution P defined in the Python implementation of IntelliGraphs. For each dataset’s logical constraints L, the sampler samples a graph F from the probability distribution P, ensuring that F satisfies all the logical constraints in L.

Existential Nodes In some settings, it is necessary to have nodes that refer to entities that only occur in one instance. For example, in the `wd-movies` dataset introduced below, each subgraph in the data represents a movie. Its actors, directors and genres are entities that occur in multiple instances, so a model can learn representations by observing the different contexts in which they occur. However, each instance also contains one node representing the movie the graph describes. These only occur in one instance, so a model cannot learn a representation for the specific instance, only a general representation which expresses that some movie exists for which this subgraph is true. We call such nodes *existential nodes* (in analogy to existentially quantified variables in logical formulas) and use a special label, such as `_movie`, to refer to them in all instances.[7]Strictly speaking, this turns the predicted subgraphs into subgraph *patterns* of the Knowledge Graph G, but we refer to them as subgraphs to keep the terminology simple.

3.1 Semantics

We use First-Order Logic (FOL) to express the underlying logical rules of the datasets. These logical rules, L, were hand-crafted for subgraphs for every dataset, and we ensured that all subgraphs complied with the logical rules. Section 7.4 (in the Appendix) provides a complete set of logical constraints for each IntelliGraphs dataset.

[5] We do not specify what larger graph this graph is a subgraph *of*. In most of our tasks, only the set of entities and relations of the larger graph is given, and the rest of the graph is left implicit.

[6] In symbolic AI, the term *inference* refers to *formal reasoning*. Here we use it to mean estimating the probability distribution over the model’s unobserved variables given observed data (*i.e.* probabilistic inference).

[7] For most models, the difference will only be in the interpretation. For example, our baseline models will learn one embedding vector for the node labelled `_movie`, which we use wherever movies occur. As such, we do not treat it differently from the node labelled `Antonio_Banderas`, although when we interpret the graph, these nodes mean different things.

Table 1: The size of the training, validation and test split for the five datasets used in this work. The number of edges is fixed for the synthetic datasets and is variable for the Wikidata-based graphs.

Dataset	Split (train/val/test)	Entities	Relations	Triples
syn-paths	60000/20000/20000	49	3	3
syn-types	60000/20000/20000	30	3	3
syn-tipr	50000/10000/10000	130	5	5
wd-movies	38267/15698/15796	24093	3	2 – 21
wd-articles	54163/22922/22915	60932	6	4 – 212

Logical Constraint Verifier. The *Logical Constraint Verifier* v is a function that verifies whether the logical constraints L hold in a generated subgraph F. We wrote a logic constraint verifier within the IntelliGraphs Python package.[8]The logical constraint verifier $v(F, L)$ returns true if and only if the subgraph F is consistent with all logical rules L.

3.2 Synthetic Datasets

Synthetic datasets allow complete control over the problem setup and provide a convenient testbed for developing new machine learning models. The dataset is generated by the generator, D. We checked if the generated subgraphs satisfy the logical rules L.[9]Here is a brief description of the synthetic datasets:

- `syn-paths` is a dataset with path graphs. Path graphs have simple semantics that can be algorithmically verified in linear time. Path graphs have a single directed path of length 3 and no other edges.
- `syn-types` contains entities with types `Language`, `Country` and `City`. These are connected by three relations according to the relation's type constraints: `sam_type_as` can only exist between the same entity types, `could_be_part_of` between a capital city and country, and `could_be_spoken_in` between a language and a country. The connections are otherwise random.
- `syn-tipr` contains subgraphs based on the *Time-indexed Person Role* (tipr) ontology pattern.[10] Here, the semantics are defined by the tipr graph pattern. The semantics include the fact that the start of an interval must precede its end.

3.3 Real-World Datasets

Wikidata [Vrandečić and Krötzsch, 2014] is a large graph-structured knowledge base which consists of crowdsourced factual knowledge on various topics.[11] We created two datasets from Wikidata using specific graph patterns to extract subgraphs about movies and research articles. Here is a brief description of the two datasets:

- `wd-movies` contains small graphs extracted from Wikidata that describe movies. Each graph contains one existential node representing the movie, entity nodes for the movie's director(s) connected by a `has_director` relation, entity nodes for the movie's cast connected by a `has_actor` relation and an entity for the movie's genre connected by a `has_genre` relation.
- `wd-articles` contains small graphs that describe research articles extracted from Wikidata. Each article is annotated by an ordered list of authors, implemented by a blank node for each author linked to a node representing the author and to a node representing the order in the author list. We add a list of the other articles that the current article references, and a list

[8] A reasoning engine could also be used for checking the subgraphs for logical consistency. We wrote a set of functions in Python for constraint verification and embedded it into the IntelliGraphs Python package to easily verify graphs without loading them into a reasoning engine.

[9] It is important to note that logical consistency does not equate to factual accuracy. We simply want to ensure that the generated dataset is consistent with the logical rules.

[10] `http://ontologydesignpatterns.org/wiki/Submissions:Time_indexed_person_role`

[11] `https://www.wikidata.org`

of subjects, together with selected superclasses of those subjects. In this dataset, most node types, including the article's node, may be existential or entity nodes.

4 Evaluation

4.1 Evaluation by bits-per-graph

The most common objective for a generative model is probably maximum likelihood: the probability of a graph from the test data under the model should have maximal probability, or, equivalently, minimal negative log probability. When base 2 logarithms are used, the latter quantity, $-\log_2 p(S, E)$, can be interpreted as the number of bits required to compress the graph [Rissanen, 1978, Grünwald, 2007]. Averaging over all graphs, we arrive at a metric of *bits-per-graph* to evaluate how well our model satisfies the maximum likelihood objective.

4.2 Semantics

We evaluate the semantics of graphs generated by our baseline models using the following evaluation metrics: 1) *% Valid Graphs* is the probability of sampling graphs that are logically valid according to the logical constraints for each dataset, 2) *% Novel Graphs* is the probability of sampling graphs that are not in the training data, 3) *% Novel & Valid Graphs* is the probability of sampling graphs that are logically valid and are not in the training data, and 4) *% Empty Graphs* is the probability of sampling graphs that did not yield any graphs, due to either $p(E)$ or $p(S \mid E)$ being too low. An ideal model gives a high probability of sampling logically valid graphs but uses a minimal number of code lengths to compress graphs.

4.3 Baseline Models

To the best of our knowledge, no probabilistic models in the literature can infer new subgraphs for knowledge graphs. Therefore, we developed a set of simple baselines inspired by traditional KGE models: ComplEx [Trouillon et al., 2016], DistMult [Yang et al., 2014] and TransE [Bordes et al., 2013]. Traditional KGE models are trained to rank all possible triples to give the correct triple the highest score [Ruffinelli et al., 2019]. ComplEx, DistMult and TransE all use different scoring functions. TransE represents relations as translation between entities, whereas DistMult models relations as bilinear interactions. ComplEx extends DistMult using complex-valued embeddings.

We model a subgraph F by decomposing it into its entities and structure $F = (E, S)$, that is, $p(F) = p(S \mid E)\ p(E)$. Unlike traditional KGE models, we train our baseline models with a maximum likelihood objective.

We decompose the objective function as follows:

$$-\log_2 p(F) = -\log_2 p(S|E) - \log_2 p(E). \tag{1}$$

Each of the terms in Equation 1 can be read as separate codelengths: $-\log_2 P(E)$ describes the bits required to encode the entities, and $-\log_2 P(S \mid E)$ describes the bits required to encode the structure once the entities are known.

We model $p(E) = \prod_{e \in E} p(e)$, with $p(e)$ estimated as the relative frequency of e in the training data (the proportion of training subgraphs it occurs in). We train KGE models to estimate $p(S \mid E)$. We use

$$p(S \mid E) = \prod_{(s,p,o) \in S_T} p((s,p,o) \mid E) \prod_{(s,p,o) \in S_N} 1 - p((s,p,o) \mid E), \tag{2}$$

where S_T represents the triples in the subgraph F, and S_N represents all possible triples that are not in the subgraph (*i.e.* all possible *negatives*).

Our *random* baseline model generates a random graph prediction by sampling $p(E)$ and $p(S|E)$ from a uniform distribution. It then computes the exact number of bits required to represent these probabilities, using $-log_2(p)$ to determine the entropy of each probability value. This model does not need to be trained.

Table 2 shows that the KGE baselines learn more compact representations than the random baseline. The ComplEx baseline is most effective at compressing the structure of these graphs $p(S|E)$, despite

Table 2: Estimate of the codelengths, $-\log_2 p(F)$, (the number of bits) required to compress a graph using the four baseline models for IntelliGraphs datasets. We used the test split for this. We rounded the numbers up/down to two decimal points.

Datasets	Baseline Models	$-\log_2 p(S\|E)$	$-\log_2 p(E)$	$C(S, E)$
`syn-paths`	*random*	95.94	98.04	193.98
	TransE	16.19	33.69	49.89
	DistMult	14.90	33.69	48.58
	ComplEx	20.71	33.69	54.39
`syn-tipr`	*random*	360.27	259.80	620.08
	TransE	28.70	40.81	69.51
	DistMult	26.70	40.81	67.51
	ComplEx	23.15	40.81	63.96
`syn-types`	*random*	187.11	59.99	247.1
	TransE	19.05	29.21	48.26
	DistMult	18.24	29.21	47.46
	ComplEx	18.48	29.21	47.69
`wd-movies`	*random*	483.81	48185.97	48669.78
	TransE	51.39	157.21	208.60
	DistMult	51.29	157.21	208.50
	ComplEx	45.46	157.21	202.68
`wd-articles`	*random*	12623.20	122366.51	134989.71
	TransE	280.67	629.98	910.65
	DistMult	271.94	629.98	901.91
	ComplEx	257.33	629.98	887.30

requiring real and imaginary parts. The scale of complexity, represented by code length, seems to increase rapidly from synthetic to real-world datasets. For instance, the highest code length for `syn-paths` is 69.51 (for the TransE baseline), while the lowest code length for `wd-movies` is 202.68 (for ComplEx). `wd-movies` and `wd-articles` have many more entities to sample, making them more challenging to compress.

4.4 Subgraph Inference

Table 3 shows the probabilities of sampling graphs that are logically consistent. We perform subgraph inference under two different settings:

- **Sampling $P(E)$ and $P(S \mid E)$.** Here, the baseline models sample both the entities that are relevant for a subgraph and infer their edge connectivity. Our results indicate that the probability of sampling valid graphs is consistently 0%. Selecting the incorrect entities negatively impacts the structure prediction. Our results indicate that this task is challenging, especially for the random baseline, as it consistently fails to infer valid subgraphs.
- **Sampling only $P(S \mid E)$.** In this setup, the model is given an advantage by having access to the correct set of entities (*i.e.*, we give $p(E)$), such that it only needs to predict the edge connections between the given entities. It is worth noting that under this setting, the baseline model collapses into a link predictor as it just predicts the edge connections between the given entities. Despite giving the advantage, the baseline models could not generate many logically consistent subgraphs. Interestingly, this also reveals the complexity of the datasets and what semantics these KGE models can learn. Most KGE models are able to generate some valid path graphs, while for `syn-tipr`, which requires some temporal reasoning, seems more challenging for all baseline models. Inferring the correct entity types from `syn-types` was possible for a few graphs.

4.5 N-ary Link Prediction

The simplest tasks in IntelliGraphs can indeed be modeled directly as an N-ary link prediction problem. For instance, the `syn-paths` graphs is a 4-ary hypergraph, and predicting hyperlinks

Table 3: Semantic validity of the graphs produced by our baseline models. High values for *% Novel & Valid Graphs* is desirable. We have tested subgraph inference under two settings: 1) Sampling from *both* $P(E)$ and $P(S \mid E)$, and 2) Sampling from $P(S \mid E)$ *only*, taking E from the test data. We check the novelty of the sampled graphs by comparing them against the training and validation set. We used the same hyperparameters from the model compression experiments here. Best performing models for each dataset is **bolded**.

Setting	Dataset	Model	% Valid Graphs	% Novel & Valid Graphs	% Novel Graphs	% Empty Graphs
Sampling from $P(E)$ and $P(S \mid E)$	`syn-paths`	random	0	0	100	0
		TransE	0.25	0.25	23.45	76.55
		DistMult	0.69	0.69	14.59	85.41
		ComplEx	**0.71**	**0.71**	**14.27**	**85.73**
	`syn-tipr`	random	0	0	100	0
		TransE	0	0	5.58	94.42
		DistMult	0	0	13.34	86.66
		ComplEx	0	0	4.95	96.05
	`syn-types`	random	0	0	100	0
		TransE	**0.21**	**0.21**	**15.44**	**84.56**
		DistMult	0.13	0.13	12.46	87.53
		ComplEx	0.07	0.07	10.25	89.75
	`wd-movies`	random	0	0	100	0
		TransE	0	0	14.61	85.39
		DistMult	0	0	12.93	87.07
		ComplEx	0	0	1.87	98.13
	`wd-articles`	random	0	0	100	0
		TransE	0	0	4.58	95.42
		DistMult	0	0	0	100.00
		ComplEx	0	0	2.46	97.54
Sampling from $P(S \mid E)$ only	`syn-paths`	random	0	0	100	0
		TransE	5.25	5.25	95.52	4.48
		DistMult	9.69	9.69	95.28	4.71
		ComplEx	**10.10**	**10.10**	**95.58**	**4.42**
	`syn-tipr`	random	0	0	100	0
		TransE	0	0	99.45	0.55
		DistMult	0	0	99.43	0.57
		ComplEx	0	0	99.64	0.36
	`syn-types`	random	0	0	100	0
		TransE	1.43	1.43	95.42	4.58
		DistMult	**1.44**	**1.44**	**96.19**	**4.81**
		ComplEx	1.01	1.01	94.17	5.83
	`wd-movies`	random	0	0	100	0
		TransE	0.07	0.07	97.01	2.99
		DistMult	0.10	0.10	95.86	4.17
		ComplEx	**0.41**	**0.41**	**93.04**	**6.96**
	`wd-articles`	random	0	0	100	0
		TransE	0	0	98.35	1.65
		DistMult	0	0	98.77	1.23
		ComplEx	0	0	100.00	0.00

on this graph is one way to solve the problem. However, as the tasks increase in complexity, the limitations of N-ary link prediction become clear. To model a task like `wd-articles` purely with link prediction, a single n-ary relation would need to capture the entirety of one subgraph describing an article, its author in order, the subjects of the article, and other features of the subgraph. Moreover, for the more complex tasks, the size of the subgraph is variable, which means that a single N-ary relation cannot capture the entire subgraph, unless the arity of the relation is somehow made variable. We leave the empricial study for future work.

5 Related Work

Datasets for Query Embedding. Query Embedding (QE) involves interpreting complex logical queries, commonly represented as a small graph, and evaluated on QE datasets, such as GQE [Hamilton et al., 2018], Query2Box [Ren et al., 2020], and BetaE [Ren and Leskovec, 2020]. Ren et al. [2023] presents a comprehensive comparison of datasets. As Ren et al. [2023] highlight in their recent surrvey, query embedding datasets lack logical rules and types. Although the datasets in IntelliGraphs are similar to query embedding datasets, there is a difference in the purpose and applications. Our datasets can be used for learning distributions to infer new logically consistent subgraphs. In contrast, QA datasets are concerned with reasoning using logical rules to find a missing entity.

Datasets for n-*ary* Relations. N-ary relations are relations involving more than two entities. Various methods have been studied in the literature that embeds complex N-ary relations, often in non-euclidean spaces [Wang et al., 2021, Wen et al., 2016]. The difference between N-*ary* relations and subgraphs is explained in Section 2.1.

Datasets for Neurosymbolic methods. If we interpret knowledge graphs as a set of logical statements, we can see that the task of subgraph prediction is a neurosymbolic method: it combines symbolic systems with neural networks. Datasets have been proposed to test various aspects of such systems: interpretability, reasoning, and generalization capabilities. Several datasets were proposed to evaluate the understanding and reasoning of complex rules and abstract concepts. Table 4 (in the appendix) compares different datasets for Neurosymbolic AI from the literature. Existing datasets focus primarily on the image and text modalities, neglecting background knowledge expressed in graphs.

6 Conclusion

Existing KG datasets used for representation learning lack well-understood semantics, which limits studying how well KGE models capture new semantics. In our work, we propose *Subgraph Inference* as a new research problem and *IntelliGraphs*, a collection of five new datasets for benchmarking models. Furthermore, we used baseline models inspired by traditional KGE models to estimate the code lengths of these graphs and sample logically valid subgraphs. Our findings show that traditional KGE models show a limited understanding of semantics after training. We observed a rapid increase in complexity, represented by code lengths, from synthetic to real-world datasets. This complexity makes real-world datasets more challenging to compress, which is an essential consideration for future research in graph compression. We found that the probability of sampling valid graphs was consistently low, emphasizing the complexity and difficulty of the task.

Limitations. *Subgraph inference* assumes that the semantics of a KG is known. However, in some cases, this assumption may not hold. Furthermore, our datasets assume we test the machine learning models in a transductive setting; entities and relations not seen during training will not be handled well.

Ethics Statement. Our synthetic graphs are based on the logical rules we constructed and should not be used for applications where factual accuracy matters. However, `wd-movies` and `wd-articles` are based on real-world factual knowledge retrieved from Wikidata. Therefore, certain biases may be inherited from Wikidata. Since these datasets are likely unsuitable for training production models or for pretraining, we do not expect that these biases will ever affect systems making real-world decisions. Transparency about dataset creation and maintenance is critical for adopting new machine learning datasets [Gebru et al., 2021]. In the appendix, we provide a data card for IntelliGraphs to provide further information about the datasets.

Applications of IntelliGraphs. It is imperative to have guarantees for safety-critical applications to prevent machine learning models from making fatal mistakes. To develop these systems, datasets with logical constraints are helpful. In some problem domains, there is little or no data available such as cases where training machine learning models on sensitive data for medical or industrial use cases. While we do not provide datasets that are directly applicable to these use cases, IntelliGraph's dataset generation framework can be used to generate synthetic datasets using background knowledge about the problem domain.

Acknowledgements We would like to thank Frank van Harmelen and Patrick Koopmann for their feedback on this work.

References

Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. In *International Conference on Learning Representations (ICLR)*, 2020.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*, pages 1–9, 2013.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 632–642, 2015.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12): 86–92, 2021.

Eleonora Giunchiglia, Mihaela Cătălina Stoian, Salman Khan, Fabio Cuzzolin, and Thomas Lukasiewicz. Road-r: The autonomous driving dataset with logical requirements. *arXiv preprint arXiv:2210.01597*, 2022.

Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

Saiping Guan, Xiaolong Jin, Jiafeng Guo, Yuanzhuo Wang, and Xueqi Cheng. NeuInfer: Knowledge inference on N-ary facts. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6141–6151, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.546. URL `https://aclanthology.org/2020.acl-main.546`.

Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. *Advances in neural information processing systems*, 31, 2018.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37, 2021.

Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6700–6709, 2019.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 4088–4095, 2017.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR, 2018.

Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: Lessons and challenges: Five diverse technology companies show how it's done. *Queue*, 17(2):48–75, 2019.

Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726, 2020.

Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint arXiv:2002.05969*, 2020.

Hongyu Ren, Mikhail Galkin, Michael Cochez, Zhaocheng Zhu, and Jure Leskovec. Neural graph reasoning: Complex logical query answering meets graph databases. *arXiv preprint arXiv:2303.14617*, 2023.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2019.

Adam Santoro, David Raposo, David GT Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *Advances in Neural Information Processing Systems*, 30, 2017.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

Alane Suhr and Yoav Artzi. A corpus of natural language for visual reasoning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 1, pages 217–231, 2017.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

Quan Wang, Haifeng Wang, Yajuan Lyu, and Yong Zhu. Link prediction on n-ary relational facts: A graph-based approach. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 396–407, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.35. URL `https://aclanthology.org/2021.findings-acl.35`.

Jianfeng Wen, Jianxin Li, Yongyi Mao, Shini Chen, and Richong Zhang. On the representation and embedding of knowledge bases beyond binary relations. *arXiv preprint arXiv:1604.08642*, 2016.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

Jiale Yang, Jinnan Li, and Yuke Zhu. A dataset and architecture for visual reasoning with a working memory. *arXiv preprint arXiv:1803.06092*, 2018.

Xianda Zhang, Yifan Zhang, and Mirella Lapata. Metaqa: A dataset of metaphorically annotated movieqa questions. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1954–1964, 2018.

7 Supplementary Material

Contents

7.1 Datasets for Neurosymbolic Methods

Neurosymbolic methods aim to combine neural networks with symbolic representations. As mentioned in Section 5, several datasets already exist in the literature for evaluating the performance of neurosymbolic methods. Table 4 highlights widely used datasets used for benchmarking neurosymbolic systems.

7.2 Reproducibility Statement

To make our work fully reproducible, we make the codebase of our experiments public and open. Our code is available on `https://github.com/thiviyanT/IntelliGraphs`. For each experiment, we also provide the hyperparameter configurations we used. Furthermore, we have released a new Python package for interacting with the IntelliGraphs datasets through the following software package repositories: **conda** (`https://anaconda.org/thiv/intelligraphs`) and **pypi** (`https://pypi.org/project/intelligraphs`). To ensure long-term preservation and easy access, we made the datasets available on Zenodo (`https://doi.org/10.5281/zenodo.7824818`). Experimental details can be found in the next Section.

7.3 Experimental Details

We used the PyTorch library [12] to develop and test the models. All experiments were performed on a single-node machine with an Intel(R) Xeon(R) Gold 5118 (2.30GHz, 12 cores) CPU and 64GB of RAM, with four NVIDIA RTX A4000 GPUs (16GB of VRAM). We used PyTorch's GPU acceleration for training the models. We used the Adam optimiser with variable learning rates [Kingma and Ba, 2014].

[12] `https://pytorch.org/`

Table 4: Brief comparison of commonly used datasets for benchmarking neurosymbolic methods, listed in ascending order of publication year. For each dataset, we provide an overview of the task, domain, modality, key characteristics, and whether the dataset is synthetic.

Dataset	Task	Domain	Modality	Key Characteristics	Synthetic
bAbI Weston et al. [2015]	Language Reasoning	Natural Language	Text	Basic reasoning, generalization	Yes
SNLI Bowman et al. [2015]	Logical Reasoning	Natural Language	Text	Entailment, contradiction, neutral relationships	No
CLEVR Johnson et al. [2017]	Visual Reasoning	Computer Vision	Images & Text	Object counting, comparison, querying attributes	Yes
NLVR Suhr and Artzi [2017]	Visual Reasoning	Computer Vision	Images & Text	Visual reasoning, natural language understanding	Yes
Sort-of-CLEVR Santoro et al. [2017]	Relational Reasoning	Computer Vision	Images & Text	Spatial and relational reasoning	Yes
Visual Genome Krishna et al. [2017]	Visual Reasoning	Computer Vision	Images & Text	Object recognition, relationships, attributes	No
Aristo Clark et al. [2018]	Science Reasoning	Natural Language	Text	Natural language understanding, applying knowledge	No
COG Yang et al. [2018]	Cognitive Capabilities	Computer Vision	Images & Text	Temporal and logical reasoning	Yes
MetaQA Zhang et al. [2018]	Multi-hop Reasoning	Graph	Knowledge Graph	Multi-step reasoning, knowledge base	No
SCAN Lake and Baroni [2018]	Compositional Generalization	Command-based Language	Text	Understanding and generating novel commands	No
Math Dataset Saxton et al. [2019]	Math Reasoning	Natural Language	Text	Language understanding, symbolic reasoning	No
GQA Hudson and Manning [2019]	Visual Reasoning	Computer Vision	Images	Text & Spatial and relational reasoning	No
ROAD-R Giunchiglia et al. [2022]	Visual Reasoning	Computer Vision	Videos & (handcrafted) Logical Rules	Logical reasoning	No

7.3.1 Hyperparameters

For each dataset, we performed hyperparameter sweeps using every baseline model (TransE, DistMult, ComplEx) using Weights&Biases [13]. For this, we used a random search strategy with the goal of finding the hyperparameter configurations that yield the minimum compression bits on the validation set. We do not include the reciprocal relation model, and we used the highest batch size that we could fit in memory. Table 5 shows the hyperparameter values we obtained via the sweeps. The random baseline did not require hyperparameter finetuning. We also used Weights & Biases for monitoring our experiments.

7.4 Semantics of IntelliGraphs

Logical rules provide a formal framework for expressing and reasoning about the semantics of a system. In this section, we discuss the logical rules we use to verify the semantics of the IntelliGraphs datasets. We express each logical rule using First-Order Logic (FOL) unless otherwise stated. We opted for First Order Logic (FOL) as the formal language to communicate logical constraints due to

[13] `https://wandb.ai/`

Table 5: The results of a random hyperparameter search, presenting the chosen hyperparameters for different datasets and baseline models. The hyperparameters include *batch size, embedding size, learning rate, biases usage, and initialization method*. The batch size indicates the number of training subgraphs processed together before updating the model. The embedding size represents the dimensionality of the entity and relation embeddings. The learning rate controls the step size taken during model optimization. The biases denote whether bias terms are included in the model, and the initialization method refers to the technique used to initialize the model's parameters.

Dataset	**Model**	**Batch Size**	**Emb.**	**Learning Rate**	**Biases**	**Init.**
syn-paths	transe	4096	1531	7.029817939842623e-05	`False`	`uniform`
syn-paths	distmult	4096	158	0.0697979730927795	`False`	`uniform`
syn-paths	complex	4096	587	5.264944612887405e-05	`False`	`uniform`
syn-tipr	transe	2048	147	0.0008716274682049251	`True`	`normal`
syn-tipr	distmult	2048	168	0.005497983171450242	`True`	`normal`
syn-tipr	complex	2048	350	0.0015597556675205502	`True`	`normal`
syn-types	transe	2048	376	0.003017403610019781	`True`	`uniform`
syn-types	distmult	2048	273	0.0006013105272716594	`True`	`uniform`
syn-types	complex	2048	996	5.603405855158606e-05	`False`	`uniform`
wd-movies	transe	4096	68	0.000638003263107625	`False`	`normal`
wd-movies	distmult	4096	181	0.00307853821840767	`True`	`uniform`
wd-movies	complex	4096	102	0.019520125878695407	`False`	`uniform`
wd-articles	transe	32	888	6.094053758340765e-05	`True`	`normal`
wd-articles	distmult	32	65	0.03833121378755901	`False`	`uniform`
wd-articles	complex	32	283	0.002251396972378282	`False`	`normal`

its ability to effectively express the necessary constraints and its widespread understanding within the machine learning community [14].

Although we provide the general FOL rules to check the semantics of graphs of *any arbitrary lengths*, we apply a size constraint (*i.e.* checking for graphs with a fixed number of triples) for the synthetic datasets. This is because the synthetic data generator produces graphs with fixed length and we defined it as part of our semantics. The size constraint can also be expressed in FOL, but we specify this constraint in *natural language* for brevity.

Traditionally, a reasoning engine is used to check logical consistencies in knowledge bases. We wrote a semantic checker in Python. This was more convenient to use within our framework as the graphs could be evaluated, without having to manually load them into a reasoning engine individually. Our semantic checker was written to closely follow the logical rules, and it is accessible through the IntelliGraph python package.

7.4.1 Logical Rules of `syn-paths`

$$\begin{aligned}
\forall x, y, z &: \mathit{connected}(x, y) \wedge \mathit{connected}(y, z) \Rightarrow \mathit{connected}(x, z) \\
\forall x, y &: \mathit{edge}(x, y) \Rightarrow \mathit{connected}(x, y) \\
\exists x &: \mathit{root}(x) \\
\forall x, y &: \mathit{root}(x) \wedge \mathit{root}(b) \Rightarrow a = b \\
\forall x &: \mathit{root}(x) \Leftrightarrow \forall y : \neg \mathit{edge}(y, x) \\
\forall x, y &: \mathit{connected}(x, y) \Rightarrow x \neq y \\
\forall x &: \mathit{root}(x) \Rightarrow \forall y : (\mathit{connected}(x, y) \vee x = y) \\
\forall x, y, z &: \mathit{edge}(y, x) \wedge \mathit{edge}(z, x) \Rightarrow y = z \\
\forall x, y, z &: \mathit{edge}(x, y) \wedge \mathit{edge}(x, z) \Rightarrow y = z \\
\forall x, y &: \mathit{edge}(x, y) \Leftrightarrow \mathit{cycle\_to}(x, y) \vee \mathit{drive\_to}(x, y) \vee \mathit{train\_to}(x, y) \\
\textit{Number of edges} &: 3
\end{aligned}$$

[14] These FOL logical constraints can also be rewritten into data specification languages, such as DataLog.

7.4.2 Logical Rules of `syn-types`

$$\begin{aligned}
\forall x, y &: spoken\_in(x, y) \Rightarrow language(x) \land country(y)\\
\forall x, y &: could\_be\_part\_of(x, y) \Rightarrow city(x) \land country(y))\\
\forall x, y &: (same\_type\_as(x, y) \Rightarrow (language(x) \land language(y))\\
&\qquad \lor (city(x) \land city(y)) \lor (country(x) \land country(y))\\
\forall x &: language(x) \Rightarrow \neg country(x) \land \neg city(x)\\
\forall x &: country(x) \Rightarrow \neg language(x) \land \neg city(x)\\
\forall x &: city(x) \Rightarrow \neg language(x) \land \neg country(x)\\
\textit{Number of edges} &: 3
\end{aligned}$$

7.4.3 Logical Rules of `syn-tipr`

$$\begin{aligned}
\forall x, y &: has\_role(x, y) \Rightarrow academic(x) \land role(y)\\
\forall x, y &: has\_name(x, y) \Rightarrow academic(x) \land name(y)\\
\forall x, y &: has\_time(x, y) \Rightarrow academic(x) \land time(y)\\
\forall x, y &: start\_year(x, y) \Rightarrow time(x) \land year(y)\\
\forall x, y &: end\_year(x, y) \Rightarrow time(x) \land year(y)\\
\forall x, y, z &: end\_year(x, y) \land start\_year(x, z) \Rightarrow before(y, z)\\
\forall x &: \neg has\_role(x, x)\\
\forall x &: \neg has\_name(x, x)\\
\forall x &: \neg has\_time(x, x)\\
\forall x &: \neg start\_year(x, x)\\
\forall x &: \neg end\_year(x, x)\\
\forall x &: academic(x) \Rightarrow \neg role(x) \land \neg time(x) \land \neg name(x) \land \neg year(x)\\
\forall x &: role(x) \Rightarrow \neg academic(x) \land \neg time(x) \land \neg name(x) \land \neg year(x)\\
\forall x &: time(x) \Rightarrow \neg academic(x) \land \neg role(x) \land \neg name(x) \land \neg year(x)\\
\forall x &: year(x) \Rightarrow \neg academic(x) \land \neg role(x) \land \neg name(x) \land \neg time(x)\\
\forall x &: name(x) \Rightarrow \neg academic(x) \land \neg role(x) \land \neg year(x) \land \neg time(x)\\
\textit{Number of edges} &: 5
\end{aligned}$$

7.4.4 Logical Rules of `wd-movies`

$$\begin{aligned}
\forall x, y &: connected(x, y) \Leftrightarrow has\_director(x, y) \lor has\_actor(x, y) \lor has\_genre(x, y)\\
\exists x &: has\_director(x, \texttt{existential\_node})\\
\exists x &: has\_actor(x, \texttt{existential\_node})\\
\exists x &: has\_genre(x, \texttt{existential\_node})\\
\forall x &: x \neq \texttt{existential\_node} \Rightarrow connected(\texttt{existential\_node}, x)\\
\forall x, y &: x \neq \texttt{existential\_node} \land y \neq \texttt{existential\_node} \Rightarrow \neg connected(x, y)\\
\forall x &: \neg connected(x, \texttt{existential\_node})\\
\forall x, y &: has\_director(x, y) \lor has\_actor(x, y) \Rightarrow person(y)\\
\forall x &: \neg person(x) \lor \neg genre(x)\\
\forall x, y &: has\_genre(x, y) \Rightarrow genre(y)
\end{aligned}$$

7.4.5 Logical Rules of `wd-articles`

$$
\begin{aligned}
\exists x &: \mathit{has\_author}(\mathtt{article\_node}, x)\\
\forall x, y &: \mathit{connected}(x, y) \Leftrightarrow \mathit{has\_author}(x, y) \vee \mathit{has\_name}(x, y) \vee \mathit{has\_order}(x, y) \vee\\
&\quad \mathit{cites}(x, y) \vee \mathit{has\_subject}(x, y) \vee \mathit{subclass\_of}(x, y)\\
\forall x, y &: \mathit{connected}(x, y) \Rightarrow \neg \mathit{connected}(y, x) \vee \mathit{cites}(y, x)\\
\forall x &: \neg \mathit{connected}(x, x)\\
\forall x, y &: \mathit{has\_author}(x, y) \Rightarrow x = \mathtt{article\_node}\\
&\quad \mathit{article}(\mathtt{article\_node}) \vee \mathit{iri}(\mathtt{article\_node})\\
\forall x &: \mathit{has\_author}(\mathtt{article\_node}, x) \Rightarrow \mathit{authorpos}(x)\\
\forall x &: \mathit{authorpos}(x) \Leftrightarrow \exists y : \mathit{has\_order}(x, y) \wedge \exists y : \mathit{has\_name}(x, y)\\
\forall x, y &: \mathit{has\_order}(x, y) \Rightarrow \mathit{authorpos}(x) \wedge \mathit{ordinal}(y)\\
\forall x, y &: \mathit{has\_name}(x, y) \Rightarrow \mathit{authorpos}(x) \wedge \mathit{name}(y) \vee \mathit{iri}(y)\\
\forall x, y, z &: \mathit{has\_order}(x, y) \wedge \mathit{has\_order}(x, z) \Rightarrow y = z\\
\forall x, y, z &: \mathit{has\_name}(x, y) \wedge \mathit{has\_order}(x, z) \Rightarrow y = z\\
\forall x &: \mathit{author}(x) \Rightarrow \neg \mathit{subject}(x) \wedge \neg \mathit{iri}(x) \wedge \neg \mathit{name}(x) \wedge \neg \mathit{ordinal}(x) \wedge \neg \mathit{author\_pos}(x)\\
\forall x &: \mathit{subject}(x) \Rightarrow \neg \mathit{author}(x) \wedge \neg \mathit{iri}(x) \wedge \neg \mathit{name}(x) \wedge \neg \mathit{ordinal}(x) \wedge \neg \mathit{author\_pos}(x)\\
\forall x &: \mathit{iri}(x) \Rightarrow \neg \mathit{author}(x) \wedge \neg \mathit{subject}(x) \wedge \neg \mathit{name}(x) \wedge \neg \mathit{ordinal}(x) \wedge \neg \mathit{author\_pos}(x)\\
\forall x &: \mathit{name}(x) \Rightarrow \neg \mathit{subject}(x) \wedge \neg \mathit{iri}(x) \wedge \neg \mathit{author}(x) \wedge \neg \mathit{ordinal}(x) \wedge \neg \mathit{author\_pos}(x)\\
\forall x &: \mathit{ordinal}(x) \Rightarrow \neg \mathit{subject}(x) \wedge \neg \mathit{iri}(x) \wedge \neg \mathit{name}(x) \wedge \neg \mathit{author}(x) \wedge \neg \mathit{author\_pos}(x)\\
\forall x &: \mathit{author\_pos}(x) \Rightarrow \neg \mathit{subject}(x) \wedge \neg \mathit{iri}(x) \wedge \neg \mathit{name}(x) \wedge \neg \mathit{author}(x) \wedge \neg \mathit{ordinal}(x)\\
\forall x, y, z &: \mathit{subclass\_trans}(x, y) \wedge \mathit{subclass\_trans}(y, z) \Rightarrow \mathit{subclass\_trans}(x, z)\\
\forall x, y &: \mathit{subclass\_of}(x, y) \Rightarrow \mathit{subclass\_trans}(x, y) \wedge\\
&\qquad\qquad (\mathit{iri}(x) \vee \mathit{subject}(x)) \wedge (\mathit{iri}(y) \vee \mathit{subject}(y))\\
\forall x, y &: \mathit{subclass\_of}(x, y) \Rightarrow \exists z : \mathit{subclass\_trans}(x, z) \wedge \mathit{has\_subject}(\mathtt{article\_node}, z)\\
\forall x, y &: \mathit{cites}(x, y) \Rightarrow \mathit{iri}(y) \wedge x = \mathtt{article\_node}\\
\forall x, y &: \mathit{has\_subject}(x, y) \Rightarrow (\mathit{subject}(y) \vee \mathit{iri}(y)) \wedge x = \mathtt{article\_node}
\end{aligned}
$$

In addition to the aforementioned rules for `wd-articles`, our semantic checker checks the ordinal of the author's position to make sure that they are a complete list of consecutive numbers (*i.e.* `ordinal_000`, `ordinal_001`, `ordinal_002`, ..., etc.), but we leave it out of the rules for brevity.

7.5 Synthetic Dataset Generation

The synthetic dataset generator contains two main modules: 1) a *triple sampler* is a module that samples new triples one by one, 2) a *triple verifier* module checks each triple for semantic validity before they are added to a subgraph. The generator builds a subgraph by sampling one triple at a time and verifiying. If the triple passes the semantic check, it is added to a subgraph. To avoid duplicate triples within the sam subgraph, we check if triple already exists before adding it to a subgraph. This is done until a certain number of valid triples are samples. For reproducibility, we use the same seed for all random data generations (`seed=42`).For each dataset, we generate *training*, *validation* and *test* sets. To avoid data leakage, we check that these graphs are unique before splitting the dataset. In this section, we briefly describe how IntelliGraphs efficiently samples valid subgraphs.

7.5.1 `syn-paths`

The entities are labelled after 49 Dutch cities and the relations are different modes of transport (`train_to, drive_to, cycle_to`). This dataset primarily checks whether baseline models can do structure learning.

A path graph $P_k(G)$ of a graph G has vertex set $\Pi_k(G)$ and edges joining pairs of vertices that represent two paths P_k, the union of which forms either a path P_{k+1}. We denote by $\Pi_k(G)$ the set of all paths of G on k vertices ($k \geq 1$), and we randomly sample n edges from $\Pi_k(G)$ to generate each path graph.

To generate a path graph, we begin by selecting a head (*i.e.* source node) by randomly selecting a Dutch city and then we sample relation and a tail (*i.e.* target node). For the next triple in the subgraph, we use the previous target node as the source node and then sample a relation and a target node. We can repeat the last step $k - 2$ number of times to build a path-graph with k edges. We ensure that each subgraph includes all three different relations. We avoid generating cyclic path-graphs.

7.5.2 `syn-types`

This dataset contains three types of entities (`cities`, `countries & languages`), 30 entities in total (10 instances of each entity type), and three relations (`same_type_as`, `could_be_part_of &` `could_be_spoken_in`). This dataset primarily checks whether baseline models can learn the types of entities correctly.

For each relation, we sample a head and a tail entity of the corresponding type. For instance, for the relation `could_be_spoken_in` we sample a language for the head of a triple and a country for the tail. Similarly, we sample other triples to be added to the same subgraph, until a certain number of valid triples have been sampled.

It is important to note that the `syn-types` dataset is not meant to be factually accurate but rather serves as a way to study the type semantics learned by machine learning models.

7.5.3 `syn-tipr`

This datasets contain three entity types (`names`, `roles`, `years`) and (`has_role`, `has_name`, `start_year` and `end_year`). We used a random name generator to generate 50 names. For simplicity, we treat *years* are entities rather than literals. In each subgraph, there are two existential nodes: `_academic` and `_time`). The main purpose of this dataset is to check structure learning and check *basic* temporal reasoning (in this case, whether `end_year` appears after `start_year`).

The subgraphs in this dataset was modeeled after the *time-indexed person role* (tipr) pattern in Semantic Web. For generating these subgraphs, we take the tipr pattern as a template and randomly sampled entities the correct entity type. For instance, the relation `has_role` would always have an academic_node in the head position of a triple and a role as a tail. Similarly, we sample triples for the other relations (`has_name`, `has_time`, `start_year`, `end_year`). Valid triples containing every relations is sampled. In total, every subgraph will contain five triples.

7.6 Wikidata Dataset Generation

For reproducibility, we use a specific Wikidata dump to extract the data, rather than the live version. For both datasets, we use the Wikidata HDT dump from 3 March 2021, available from the HDT website [15].

In both cases, we first extract all data that fits the template of the graph, for instance, for every movie we extract all actors, directors and genres. We then *prune* this data to ensure that every entity occurs in enough instances to allow a model to learn a representation for it. Depending on the dataset we either remove the infrequent nodes or replace them by existential nodes. We set the minimal frequency to 6 in both datasets.

To avoid the situation where certain entity nodes are only present in the validation or test data, we must make our splits carefully. Ideally, we'd like for each entity to be present in all three splits of the data, and where this is not possible, for it to be present in at least the training data.

To achieve this, we use the following algorithm: for each instance we collect "votes" among all its entities for which of the three splits it should be part of. Simultaneously, for each entity, we collect the splits of which it is a member. The aim is to have all entity in each instance vote for the same split, and for each entity to be represented in all splits. We first alternate fixing one of the two problems: we unify the votes by choosing a random entity and setting the votes of the other entities in the instance to that vote. After all votes have been fixed, we fix the split memberships by, for each entity that is not representing in all splits, taking the most frequent split and changing the vote of one of its instances to the missing split, repeating until all splits are represented.

[14] `https://www.behindthename.com/random/`

[15] `https://www.rdfhdt.org/datasets/`

We alternate these steps for 50 iterations. Then, in the first step, we move any instance with conflicting votes to the training data and repeat the iteration in this fashion for another 20 steps. For both datasets, this leads to all entities being represented in the training data, and only a small number present in only the test or only the training data.

For both datasets, the labels are Wikipedia IRIs, but a mapping to human-readable labels is provided. In this paper, we replace IRIs with these for readability.

7.6.1 `wd-movies`

We collect all entities that are labeled as "instance of" the class "film". For each we extract all entities connected by the relations "cast member", "director" and "genre" as its actors, directors and genres respectively.

We then prune the data by removing all actors, directors and genres that do not appear in at least 6 instances. We then remove any movies that are left with no actors or no directors. We allow movies with no genres. We iterate these two steps until no changes are made. Finally, we make a test, train and validation split by the process described above. The following Wikidata properties and entities were used:

label	**wikidata IRI**
instance of	`http://www.wikidata.org/prop/P31`
film	`http://www.wikidata.org/entity/Q11424`
cast member	`http://www.wikidata.org/prop/P161`
director	`http://www.wikidata.org/prop/P57`
genre	`http://www.wikidata.org/prop/P136`

7.6.2 `wd-articles`

We collect all entities from wikidata that are the object of a triple with the relation "cites".

For each article we collect the full list of authors, using the relations "author" and "author name string". The former is used to refer to authors that are represented in Wikidata as an entity, and the latter is used for authors represented only by their name as a string literal. We require at least one of the authors to be represented by an entity. If not, the article is filtered out.

Such statements are commonly annotated in Wikidata with an *ordinal*, representing the order of the author in the author list. We extract these as well. If any author does not have an ordinal or if the collection of these ordinals does not coincide exactly with the sequence $1, \ldots, n$, with n the number of authors, the article is filtered out.

We then collect all articles that, as recorded in Wikidata, the current article cites. If there are no such references, the article is filtered out.

Finally, we collect the article subjects, and for each subject, every superclass and its superclass, that are an instance of "academic discipline". We do not filter based on the subjects (no subjects or superclasses is allowed).

We collect the first 100 000 such articles for the dataset `wd-articles`, and all such articles for the dataset `wd-articles-large`.

As with `wd-movies`, we prune the data to eliminate any entities that occur in fewer than 6 instances. For the authors, the article itself and the subjects, we replace these with existential nodes. These have node labels specific to the role they play in the graph: `_article`, `_author001`, and `_subject001`. Any references to infrequent entities are removed. As before. this removal process is iterated until the dataset stabilizes.

Splits are then made using the algorithm described above. In the construction of the dataset, we add authors by introducing a blank node (using label `_authorpos` and the relation `has_author`), to which the author identity (`has_name`) and the ordinal (`has_order`) are connected. References are added by a single edge with the relation `cites` and subjects and superclasses with the relations `has_subject` and `subclass_of`.

syn-paths

```
[Nieuwegein drive_to Lelystad, Lelystad drive_to IJmuiden, IJmuiden cycle_to Zaanstad]
[IJmuiden cycle_to Maastricht, Maastricht train_to Roermond, Roermond drive_to Groningen]
[Hilversum cycle_to Emmen, Emmen drive_to Spijkenisse, Spijkenisse train_to Sittard]
```

syn-tipr

```
[_academic has_name Cleophas Erős, _academic has_role masters researcher, _academic has_time _time, _time start_year
2016, _time end_year 2018]
[_academic has_name Romana Sitk, _academic has_role professor, _academic has_time _time, _time start_year 1982, _time
end_year 2009]
[_academic has_name Drusus Krejči, _academic has_role assistant professor, _academic has_time _time, _time start_year
1996, _time end_year 2000]
[_academic has_name Božidar Bullard, _academic has_role professor_academic has_time _time, _time start_year 1973, _time
end_year 1988]
```

syn-types

```
[Dutch same_type_as English, Budapest could_be_part_of United Kingdom, Czech spoken_in Serbia]
[Serbia same_type_as Spain, Paris could_be_part_of Norway, Dutch spoken_in Greece]
[Greek same_type_as Italian, Budapest could_be_part_of Ireland, French spoken_in Serbia]
```

wd-movies

```
[_movie has_director P. Pullaiah, _movie has_actor Gummadi Venkateswara Rao, _movie has_actor Akkineni Nageswara Rao,
_movie has_actor Anjali Devi, _movie has_actor Chittoor Nagaiah, _movie has_actor Ramana Reddy, _movie has_actor Relangi
Venkata Ramaiah, _movie has_actor S. V. Ranga Rao, _movie has_actor Santha Kumari, _movie has_genre historical film,
_movie has_genre biographical film]
[_movie has_director Albert Brooks, _movie has_actor Kathryn Harrold, _movie has_actor Albert Brooks, _movie has_actor
Bruno Kirby, _movie has_genre comedy film]
[_movie has_director Dragoslav Lazić, _movie has_actor Vesna Malohodžić, _movie has_actor Snežana Savić, _movie
has_genre comedy film]
[_movie has_director Balu Mahendra, _movie has_actor Silk Smitha, _movie has_actor Sridevi, _movie has_actor Kamal
Haasan, _movie has_genre romance film]
```

wd-articles

```
 [_article has_author _authorpos000, _authorpos000 has_name _author000, _authorpos000 has_order ordinal_001,
_article has_author _authorpos001, _authorpos001 has_name _author001, _authorpos001 has_order ordinal_002, _article
has_author _authorpos002, _authorpos002 has_name _author002, _authorpos002 has_order ordinal_003, _article
has_author _authorpos003, _authorpos003 has_name _author003, _authorpos003 has_order ordinal_004, _article
has_author _authorpos004, _authorpos004 has_name _author004, _authorpos004 has_order ordinal_005, _article has_author
_authorpos005, _authorpos005 has_name _author005, _authorpos005 has_order ordinal_006, _article has_author _authorpos006,
_authorpos006 has_name _author006, _authorpos006 has_order ordinal_007, _article has_author _authorpos007, _authorpos007
has_name _author007, _authorpos007 has_order ordinal_008, _article cites http://www.wikidata.org/entity/Q25938995,
_article cites http://www.wikidata.org/entity/Q28242060, _article cites http://www.wikidata.org/entity/Q28286732,
_article cites http://www.wikidata.org/entity/Q34453213, _article cites http://www.wikidata.org/entity/Q34541710,
_article cites http://www.wikidata.org/entity/Q35758845, _article cites http://www.wikidata.org/entity/Q37942996,
_article cites http://www.wikidata.org/entity/Q37972005, _article cites http://www.wikidata.org/entity/Q42642132,
_article has_subject http://www.wikidata.org/entity/Q214781, http://www.wikidata.org/entity/Q214781 subclass_of
http://www.wikidata.org/entity/Q413]
```

Figure 2: IntelliGraphs contains five datasets: `syn-paths`, `syn-tipr`, `syn-types`, `wd-movies`, and `wd-articles`. Here we showcase a few example subgraphs from each dataset. The subgraphs are presented as a list of triples, where each list item represents a subgraph.

The following Wikidata properties were used.

label	wikidata IRI
cites	`http://www.wikidata.org/prop/P2860`
author	`http://www.wikidata.org/prop/P50`
author name string	`http://www.wikidata.org/prop/P2093`
main subject	`http://www.wikidata.org/prop/P921`
subclass of	`http://www.wikidata.org/prop/P279`
academic discipline	`http://www.wikidata.org/entity/Q11862829`

7.6.3 Example subgraphs

Figure 2 showcases a selection of example subgraph from each dataset: `syn-paths`, `syn-tipr`, `syn-types`, `wd-movies`, and `wd-articles`.

8 Datacard

An up-to-date version of the data card can be found on `https://github.com/thiviyanT/IntelliGraphs/blob/main/Datacard.md`.