UNIVERSITY
OF OSLO

Master's thesis

Snippet Generation with Reasoning and Embedding Techniques

Alva Marie Hørlyk

Data Science: Database Integration and Semantic Web
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences

Autumn 2023

Alva Marie Hørlyk

Snippet Generation with Reasoning and Embedding Techniques

Supervisor:
Jieying Chen

Abstract

An increasing amount of Linked Open Data is now available on the Web, resulting in the expansion of knowledge graphs as more triples are added to them. To prevent information overload and improve task efficiency, multiple methods exist to summarize a knowledge graph. Entity summarization is one of these methods and involves producing a small subset, a snippet, of the entity description(s), for a given entity or group of entities. The snippet can then be used in tasks, instead of a lengthy entity description. However, entity summarization is limited to instance level entities and cannot produce snippets for properties or classes. These are other components of a knowledge graph that could be of interest, particularly in combination with instance level entities. Therefore, in this thesis, we propose approaches for generating snippets for not only instance level entities, but also properties and classes. We present two approaches: one based on reasoning with RDFS entailment rules and another based on knowledge graph embedding using RDF2Vec. Additionally, we created a benchmark to evaluate the performance of these two approaches. The results, especially for the reasoning-based approach, were promising.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor Jieying Chen for her great guidance and support throughout this thesis. She has taught me a lot about academic writing, and difficult problems were always more understandable after discussing them with her. Thank you for always being patient with me, and always ready to provide ideas and insight.

I would also like to thank my family and friends for all their support during the writing of this thesis. You were always there to listen, and help me get my thoughts in order when things felt overwhelming. Thank you for always believing in me.

Alva Marie Hørlyk

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The open data movement has caused an immense amount of data to be available on the Web. Some of this data are linked, and referred to as Linked Open Data (LOD). LOD is based on the RDF standards of the semantic web, [1] and to publish data in this format has become increasingly popular. Many companies like Google and Microsoft have their own knowledge graphs, and the DBpedia knowledge graph contains information from Wikipedia, but presented in the RDF format. As new data is published on the Web, these and other knowledge graphs and RDF datasets, both of which can be seen as a set of RDF triples, grow in size, causing increasingly large descriptions for the entities present. A human user interested in the entity will then be overloaded with information, and spend an unnecessary amount of time processing it. In certain tasks like linking an entity mentioned in a document to an entity in a knowledge base, or use the entity description to gain some simple background knowledge about an entity, a user could benefit from receiving a short summary of the entity description instead. Such a summary, known as an entity summary, would help the user feel less overwhelmed and quicker be able to solve tasks.

Many approaches already exist to produce entity summaries. RELIN [CTQ11], one of the first entity summarizers, is based on a random surfer model, and produces a general summary for one entity. Another entity summarizer for one entity is FACES [GTS15], which groups triples in the entity description together, and then triples are selected from each group to form a summary. C3D+P [CXQ15a] creates summaries for two entities by solving a combinatorial optimization problem, with the focus on relatedness and differences between the two entities. REMES [Gun+17] and COMB [CXQ15b] produces summaries for a collection of entities, also by solving a combinatorial optimization problem.

In addition to entities, a triple can also contain properties, classes and strings. All the entity summarizers mentioned above take an entity or a collection of entities as input, but a user might need information about a class or a property as well. We want to create a summarizer that not only can take entities, but also properties and classes as input. Different from the aforementioned entity summarizers, this summarizer would have to choose triples from the full knowledge graph, not only the entity descriptions.

Another line of work related to summarizing data on the web is snippet generation. In [Che+17] they present an approach for generating a small subset,

[1] For more information see https://www.w3.org/egov/wiki/Linked_Open_Data

a snippet, of an RDF dataset to illustrate its content. In [WCK19] they also presents an approach for generating a snippet for an RDF dataset, but in addition to illustrate the content, the snippet should also cover some keywords given by a user, to better accommodate user needs.

Example 1.0.1 (A knowledeg graph with four triples.)**.**

t_1: $\langle$`person1`, `type`, `Student`$\rangle$

t_2: $\langle$`person1`, `university`, `UiO`$\rangle$

t_3: $\langle$`person2`, `type`, `Student`$\rangle$

t_4: $\langle$`person2`, `knows`, `person3`$\rangle$

t_5: $\langle$`Student`, `subClassOf`, `Person`$\rangle$

Namespaces are omitted for better readability.

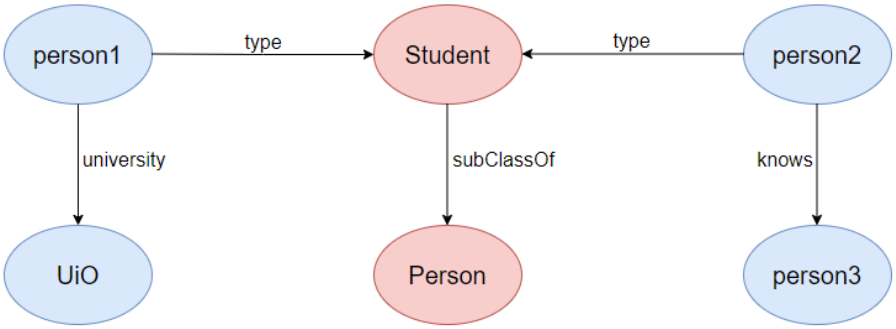

Figure 1.1: A graph representation of the triples in Example 1.0.1. Blue represents an entity, red represents a class, and an arrow represents a property.

We want to combine the idea behind entity summarizers and snippet generators, and create a snippet for a knowledge graph based on a set of keywords, where the keywords can be entities, properties and classes. The snippet should include the triples most related to the keywords. For example, given a knowledge graph with the triples in Example 1.0.1, and the keywords `person1` and `Student`, t_1 is the most related triple to the keywords, since it shows the relationship between the two keywords, and should therefore be included in the snippet.

We propose two approaches to snippet generation with given keywords. Both approaches produce a score for each triple in the knowledge graph, and the triples are ranked accordingly, and then the top ranked triples make up the snippet.

Approach 1 is a reasoning based snippet generator, named RBSG, where the score of a triple depends on materialization of the knowledge graph. We materialize the knowledge graph with the RDFS entailement rules, and score the original triples based on how many new relevant triples they can help entail. A new triple needs to contain a keyword for it to be considered relevant, and therefore the snippet will be dependent on the keywords. For example can

t_1 and t_5 in Example 1.0.1 entail $\langle$`person1`, `type`, `Person`$\rangle$, and if `person1` or `Person` were a keyword, that should positively affect t_1´s and t_5´s score.

Approach 2 is an embedding based snippet generator, named EBSG, where we obtain a vector representation of each entity, property and class in the knowledge graph, and use these vectors to score the triples. We use RDF2Vec [Ris+19] as the embedding method. To score the triples in the knowledge graph, we use the cosine similarity between the vector representation of an element in the triple and the vector representation of a keyword. For example, given the the triples in Example 1.0.1, `Student` and `university` should have a larger cosine similarity score than `Student` and `knows`, since `Student` is more closely related to `university`. Due to the multiple combinations of triple elements and keywords when calculating cosine similarity scores, we explore different scoring functions, but the total score of a triple depends on an individual score for each of its elements.

To evaluate our two approaches we consider:

1. Run time of our algorithms
2. How much overlap the generated snippets have with snippets in a benchmark

Since existing benchmarks only works for entities as input, they cannot be used to evaluate our approaches. Therefore we have created a new benchmark that also consider properties and classes as input. The new benchmark is based on the benchmark ESBM [Liu+20], and for simplicity we only combined the entities summarized in ESBM with one more element. We made three different types of ground-truth snippets, where they where based on:

1. Two entities
2. A combination of one entity and one property
3. A combination of one entity and one class

Nonetheless, our approaches can also handle a larger set of keywords than two, and both properties and classes can be in the same keyword set.

Outline

The rest of the thesis is organised as follows:

Chapter 2 provides some necessary preliminary knowledge.

Chapter 3 discusses related work.

Chapter 4 defines the problem and describes RBSG and EBSG.

Chapter 5 describes how the benchmark for evaluation was made.

Chapter 6 presents and discusses the experimental results.

Chapter 7 concludes the thesis with future work.

Appendix A provides a table with the RDFS entailment rules.

Appendix B provides a list of properties with similarity score 0 in EBSG.

Appendix C evaluates an earlier version RBSG

CHAPTER 2

Preliminaries

2.1 Data Model

In this thesis we work with semantic data, which is structured data represented such that it can easily be processed by machines. An example of sematic data is the Resource Description Framework (RDF), which is a framework for representing information on the Web, where the information is expressed using a triple pattern. We employ a data model corresponding to RDF.

Let $\mathcal{L}$ be a set of literals, that is values such as strings, numbers or dates, and let $\mathcal{I}$ be the set of non-literal RDF resources. We consider three types of resources in our work: entities, properties and classes. Literals are also resources, but they are not given much attention in this thesis, so henceforth resource will be a common word for entities, properties and classes.

Definition 2.1.1 (Entity)**.** A non-literal resource at the instance level [WCK19]. The set of entities is denoted by $\mathcal{E}$. In Example 1.0.1 `person1`, `person2`, `person3` and `Ui0` are considered entities.

Definition 2.1.2 (Property)**.** A binary relation. The set of properties is denoted by $\mathcal{P}$. In Example 1.0.1 `type`, `university`, `subClassOf` and `knows` are considered properties.

Definition 2.1.3 (Class)**.** Entity types. The set of classes is denoted by $\mathcal{C}$. In Example 1.0.1 `Student` and `Person` are classes.

$\mathcal{I} = \mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$, and $\mathcal{I}$ and $\mathcal{L}$ are the components of a triple.

Definition 2.1.4 (Triple)**.** A triple t is a 3-tuple on the form $\langle$subject, predicate, object$\rangle$, or $\langle s, p, o \rangle$ for short, where $s \in \mathcal{I}$, $p \in \mathcal{P}$, and $o \in \mathcal{I} \cup \mathcal{L}$. A triple can also be referred to as a statement, or fact. Define $\Omega(t)$ as the set of elements that constitute a triple, that is $\Omega(t) = \{s, p, o\}$.

Triples can also contain blank nodes, which do not identify specific resources. Blank nodes can be in the subject or object position of a triple to show a relationship with an unspecified thing. However, [Liu+21] notes that they are difficult to handle and usually ignored in entity summarization. Hence, they will also be ignored in our work.

A triple can carry either data knowledge or ontology knowledge. Ontology knowledge is terminological knowledge, for example information about the domain and range of properties, subclass relationships, and subproperty

relationships. Data knowledge is assertional knowledge, which is information about entities, and relationships between them.

2.2 Knowledge Graph

The data we are working with in this thesis is in the form of a knowledge graph. It can be thought of as just a set of triples, but below we give a formal definition.

Definition 2.2.1 (Knowledge Graph)**.** Let T be a set of triples on the form $\{\langle s, p, o\rangle \mid s \in \mathcal{I}, p \in \mathcal{P}, o \in \mathcal{I} \cup \mathcal{L}\}$. A knowledge graph is characterized by the 3-tuple $\langle \mathcal{I}, \mathcal{L}, T\rangle$. We denote a knowledge graph as KG. When we talk about a subset of the knowledge graph, we mean a subset of the triple-set T.

A KG can be visualized as a graph where the subjects and objects are the nodes, and the predicates are the edges. Note, however, that this graph is not a real graph, just a set of triples. From Definition 2.1.4, the subject, predicate and object of a triple can all be in $\mathcal{P}$, hence the set of nodes and the set edges of an RDF graph are not necessarily disjoint.

2.3 Knowledge Graph Summarization

Since KGs can be very large, the problem of creating a small subset, a snippet, for it is a popular line of research.

Definition 2.3.1 (Snippet)**.** Given a KG $G = \langle \mathcal{I}, \mathcal{L}, T\rangle$, and a positive integer k, a snippet S of G is a subset of T, such that $S \subseteq T$ and $|S| \leq k$.

A snippet can be generated for a KG without any specifications, or it can depend on a *signature* specifying elements of interest.

Definition 2.3.2 (Signature)**.** A set $\Sigma = \{q_1, q_2, ..., q_n\}$, where for example:

- $q_i \in \mathcal{E}, i = 1, ..., n$, or
- $q_i \in \mathcal{I}, i = 1, ..., n$, or
- q_i is a keyword, $i = 1, ..., n$. A set of keywords is a dataset search query where the stop words are removed [Che+19].

Note that in the introduction the signature where referred to as a set of keywords, but will henceforth be referred to as a signature.

If we want to find a snippet for a KG, and a signature was given together with the KG, we want the snippet to summarize the elements of the signature, using triples from the KG.

A special case of a snippet is an entity summary. Then the signature is a set of entities, and the triples in the snippet are part of the entities descriptions.

Definition 2.3.3 (Description of entity)**.** Let T be a set of triples. The description of an entity $e \in \mathcal{E}$ is a subset of triples $t \in T$ where e appears in the subject or object position of t. Denote the description of e as $Desc(e)$, such that:

$$Desc(e) = \{\langle e, p, o\rangle \in T\} \cup \{\langle s, p, e\rangle \in T\}$$

If all triples in $Desc(e)$ on the form $\langle s, p, e\rangle$ are rewritten to $\langle e, p^-, s\rangle$, where p^- is the inverse of p, then all triples in $Desc(e)$ have e as subject. Therefore, only the predicate and object is of interest when choosing triples for the summary [Liu+21]. This predicate-object pair is called a *feature*.

Definition 2.3.4 (Entity Summary)**.** Given a KG $G = \langle \mathcal{I}, \mathcal{L}, T\rangle$, a positive integer k, and a signature $\Sigma = \{e_1, e_2, ..., e_n\}$ where $e_i \in \mathcal{E}, i = 1, ..., n$, an entity summary is a snippet S with the added constraint $S \subseteq Desc(e_1) \cup Desc(e_2) \cup ... \cup Desc(e_n)$.

Entity summarization is the task of finding an optimal entity summary, where *optimal* depends on what the summary is meant to be used for. An *entity summarizer* is a method for solving the problem of entity summarization.

CHAPTER 3

Related Work

This chapter briefly presents related approaches to solving two problems: the problem of snippet generation and the problem of entity summarization. As defined earlier, a snippet is a size restricted subset of the knowledge graph, and the problem of entity summarization is a problem of snippet generation. However, in this chapter we distinguish them since entity summarization only works with entities as input, while the problem of snippet generation can be solved for different kinds of input, or no input at all (except for the KG itself). Therefore, we use entity summary, or only summary, when we talk about a snippet generated with an entity summarizer. In addition, we offer a brief introduction to knowledge graph embedding.

3.1 Snippet Generation

Snippet generation provides a solution for how to effectively reuse existing datasets on the Web, and several methods have been proposed to cover different needs. For example can snippets be generated to more efficiently answer queries against [Dol+07; Rie+14], or to provide an overview of the content of the dataset, either on a higher level [BBP14] or the data level [Che+17].

In [Che+17] they propose a snippet to illustrate the contents of a dataset, so users are able to quickly inspect it. The quality of the snippet depends on three aspects: coverage, familiarity, and cohesion, and the optimal snippet is found by solving a combinatorial optimization problem. The optimal snippet should cover all the most important classes and properties, while also containing cohesive information familiar to the user.

In [WCK19] they take reusing existing datasets on the Web one step further and generate snippets which cover both query keywords and the content of the dataset, and hence also consider users needs. The approach was given the name KSD (Keyword, Schema, Data), and the generated snippet was expected to have a good coverage of the query keywords and the content of the dataset at both the schema and the data level. As for the multi-entity summarizers, KSD is presented as a combinatorial optimization problem, but solved using the weighted maximum coverage (WMC) problem.

3.2 Entity Summarization

Given an entity $e \in \mathcal{E}$, $Desc(e)$ can be very large, and possibly growing when new information, that is, new triples, are added to the KG. The goal of entity summarization is, as mentioned in Chapter 2, to find a subset of $Desc(e)$ that best describes the entity e.

Entity summarizers can be developed for general purposes, or they can be designed to improve the performance of a specific task. A task can for instance be entity resolution (C3D, C3D+P [CXQ15a]), entity linking (COMB [CXQ15b]), document understanding (REMES [Gun+17]) or entity search (MMR-QSFS [ZZC12]). The summarizers developed for general purposes can also be used for specific tasks (e.g. RELIN [CTQ11]), but they are generally outperformed, because they were not designed for the task. Both general purpose summaries and task specific summaries can be created for one entity, or for a collection of entities.

RELIN [CTQ11] is an entity summarizer which produces general purpose summaries for one entity. RELIN is a random surfer model, where the surfer can chose between a relational move and an informational jump. Given a triple t, the surfer can chose between moving to a new triple with related information to t, or a triple that carries new information. RELIN is implemented using probability matrices.

Unlike RELIN, REMES [Gun+17], C3D+P [CXQ15a] and COMB [CXQ15b] are multi-entity summarizers (C3D+P is a two-entity summarizer), with REMES creating general purpose summaries, and C3D+P and COMB creating task specific summaries. They have a small difference in their problem definitions, but all three entity summarizers are presented as a combinatorial optimization problem, solved using the Quadratic Knapsack Problem/Quadratic Multidimensional Knapsack Problem (QKP/QKMP).

REMES [Gun+17] (RElatedness-based Multi-Entity Summarization) was originally developed to help users understand documents better, but the summaries are general purpose summaries. The point was to give users a summary of entities mentioned in the document as background knowledge, and see if the summaries were useful. To create useful summaries for each entity the goal is to maximize the intra-entity importance in order to include the most important triples, while simultaneously minimize intra-entity relatedness of features to improve diversity and coverage. To show relatedness between the different entities, inter-entity relatedness should also be maximized.

C3D and C3D+P [CXQ15a] produce summaries meant for entity resolution, which is the task of finding entity descriptions that refer to the same real-world entity. C3D is an approach meant to help human users judge whether two entity descriptions refer to the same real-world entity. The users are given a summary of the two entity descriptions, instead of the whole entity descriptions, to help them judge more efficiently. C3D+P extends C3D by grouping and ordering the selected features, to further improve efficiency. When triples are selected for the two summaries, four kinds are preferred: common, conflicting, characteristic, and diverse. Triples with common and conflicting features are useful to have in a summary to help indicate a match or a non-match, respectively. However, the triples chosen should also carry sufficient information about the real world, and minimize intra-entity relatedness of features to improve diversity, as in REMES.

COMB [CXQ15b] was created for entity linking, which is the task of linking

an entity mentioned in a document to an entity in a knowledge base, like for instance DBpedia. COMB is an approach developed to help human users link entities together accurately and efficiently. The users select matches based on the context of the document, and a summary of the candidate entities in the knowledge base. To best help the users, the triples chosen for the summary should have high characterizing and differentiating power, to determine the identity of a candidate entity and show the difference between candidate entities. The triples in the summaries should also best reflect the relevance of each candidate entity to the entity mention and its context.

3.3 Knowledge Graph Embedding

An *embedding* for a group of objects (e.g. words, relations, or entities) is an injective function that maps each object to a real-valued vector, so that the intrinsic relations between the objects are maintained [Bos+22; Suc+19]. In our case we are working with a KG, and want to embed the resources in the set $\mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$. We want the resources that are semantically similar to be mapped to vectors that are close to each other in the vector space.

There exists multiple Knowledge Graph Embedding (KGE) methods, and in [Bos+22] they present and discusses many of them. The embedding models they present are divided into three different types of models. First there are geometric models, which interpret relations as geometric operations in the vector space. TransE [Bor+13], TransH [Wan+14] and RotatE [Sun+19] are examples of geometric models. Second there are semantic matching models, where the vector of the subject and the vector of the object is directly compared in order to assess how likely the triple is to be true. Examples are RESCAL [NTK11], DistMult [Yan+15], ComplEx [Tro+16] and SimplE [KP18]. Lastly there are deep models, which use deeper neural architectures to create embeddings. Some examples are NTN [Soc+13], Graph Convolutional Networks (GCNs), which uses a type of neural networks introduced in [KW17], and Relational GCNs [Sch+18].

CHAPTER 4

Approaches

4.1 Problem Definition

Given a KG $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$, a signature $\Sigma = \{q_1, q_2, ..., q_n\}$ where $q_i \in \mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$ for $i = 1, ..., n$, and a positive integer k, we want to find the optimal snippet S for G, of length k and with respect to Σ.

The *optimal snippet* will the solution to:

$$\text{find } \arg\max_{S \subseteq T} \sum_{t \in S} \text{score}(t), \qquad \text{subject to } |S| = k, \tag{4.1}$$

where score(t) is a function returning a relatedness score between triple t and the signature Σ. Also note that since $|S| = k$, only KGs where $|T| \geq k$ can be summarized.

Algorithm 1 presents the general approach to how we solve (4.1). The function A is doing something to the KG, that is, either materialization or embedding with RDF2Vec, and the results of applying A are used to score the triples. The triples are then ranked and the k highest ranked triples are used to create a snippet S.

Algorithm 1 General snippet generation algorithm

Input: A knowledge graph $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$, a signature Σ and an integer k
Output: A snippet S

$S \leftarrow \emptyset$
$A(G)$
$scores \leftarrow \emptyset$
for $t \in T$ **do**
    compute score(t)
    add score(t) to *scores*
end for
sort *scores* high to low
for $i \leftarrow 1$ to k **do**
    if $scores[i] =$ scores(t) **then**
        add t to S
    end if
end for
return S

In the rest of this chapter we present two approaches to snippet generation, one based on reasoning, and one based on embedding. We describe the function A for each approach, and discuss score functions.

4.2 Reasoning Based Snippet Generation (RBSG)

In our first snippet generation approach we let A in Algorithm 1 be a function that does reasoning on a KG. Based on the logical conclusions from the reasoning, we defined a scoring function for the triples in the KG, and created a snippet containing the highest scored triples, as described in Algorithm 1. The reasoning method we used was materialization.

Definition 4.2.1 (Materialization)**.** Materialization applies rules recursively to a graph, adding entailments back to the graph until nothing new can be added [Hog+21]. In this thesis the rules will be the RDFS entailment rules, which are provided in Appendix A.

Given a KG G and a signature Σ, the triples in G can be scored and ranked based on materialization of G. We want to use the scoring to create an optimal snippet S of G. There are many different ways to score the triples in G, but here it will involve the amount of entailments the triples can generate about the elements in Σ. A mathematical definition of score will be given below.

4.2.1 RDFS Entailment Rules

An RDFS entailment rule (hereafter referred to as RDFS rule or just rule) consists of one or two premises, and a conclusion, the entailment, where both the premises and the entailment are triples. The RDFS rules are applied to a set of triples in order to add more triples to the set, and therefore create a richer KG. The entailments contain new information based on domain or range knowledge, or a property/class hierarchy.

However, we are using the RDFS rules for materialization of a KG, and not all the rules are relevant for materialization. We want to use the rules to generate not only new triples, but new triples that carry useful information. Therefore we only consider the rules with two premises for the materialization, that is we only consider rule 2, 3, 5, 7, 9, and 11. The other rules generates too general knowledge, for example that every property is a subproperty of itself (rdfs6), and will hence be disregarded. Using this incomplete set of RDFS rules, instead of full RDFS, is fairly common in practice [GM10; Sub+16].

4.2.2 Materialization with RDFS Rules

Let $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$ be a knowledge graph and $G' = \langle \mathcal{I}, \mathcal{L}, T' \rangle$ be the materialized version of G . By letting the triples in T work as the premises of the RDFS rules, we can entail new triples, and add these triples to T. We do this recursively, until no new triples can be produced, and G is materialized.

Example 4.2.2. The second RDFS rule used on G: If $\langle p, \texttt{domain}, c \rangle \in T$ and $\langle x, p, y \rangle \in T$, then add $\langle x, \texttt{type}, c \rangle$. In this case $\langle x, \texttt{type}, c \rangle$ is the conclusion, but since rules are applied recursively, it can later be used as a premise for generation of more new triples.

If a signature Σ is given together with G, G can be materialized with respect to Σ. Instead of adding all the entailments to G, only the entailments containing an element of Σ is added.

Example 4.2.3. Continuing Example 4.2.2, if a signature Σ was given, $\langle x, \texttt{type}, c\rangle$ would only be added if $x \in \Sigma$ or $c \in \Sigma$.

Given a knowledge graph $G = \langle \mathcal{I}, \mathcal{L}, T\rangle$, and an RDFS rule, let $P_1 \subseteq T$ be the set of triples of G that can be used as premise 1 for the RDFS rule. Then, given P_1, let $P_2 \subseteq T$ be the set of triples of G that can be used as premise 2 for the RDFS rule. Also, let t_{concl} be a conclusion of the RDFS rule. Algorithm 2 describes the process of materializing a knowledge graph with respect to a signature, using the incomplete set of RDFS rules.

Algorithm 2 Materialization algorithm for KGs using RDFS rules

Input: A knowledge graph $G = \langle \mathcal{I}, \mathcal{L}, T\rangle$ and a signature Σ
Output: A knowledge graph G'

$T' \leftarrow T$
$l \leftarrow 0$ ▷ Previous number of triples in T', initially 0
while $l < length(T')$ **do** ▷ $length(T')$ is the number of triples in T'
  $l \leftarrow length(T')$
  for each RDFS rule **do**
    for each $u \in P_1$ **do** ▷ Feasible triples for premise 1
      for each $v \in P_2$ **do** ▷ Feasible triples for premise 2, given $P1$
        $t_{concl} \leftarrow$ the conclusion given premises u and v.
        if t_{concl} contains some $q \in \Sigma$ **then**
          add t_{concl} to T'
        end if
      end for
    end for
  end for
end while
$G' \leftarrow \langle \mathcal{I}, \mathcal{L}, T'\rangle$
return G'

4.2.3 RBSG Score

To score a triple t in G we sum over the new information about each $q \in \Sigma$ that t can generate, and divide by $|\Sigma|$. The new information t can generate about $q \in \Sigma$, is given as the proportion of entailments containing q, generated with t as a premise, out of all entailments with q that has been generated.

In addition to how many entailments with a $q \in \Sigma$ t can generate, we also consider whether or not t itself contains a $q \in \Sigma$. If t contains $q \in \Sigma$, that is considered equally important as if t could be used to generate a triple with q. To take this into consideration we say that if t contains $q \in \Sigma$, t can generate itself, and hence add t to the set of triples with q that t can generate. Denote this set as $T_{t,q}$.

Let $T_{t,q}$ be a set containing triples w where:

- w contains element q, and $q \in \Sigma$.
- either 1 or 2 is true:
  1. w was the conclusion of an RDFS rule where triple t was one of the premises
  2. $w = t$

The score of a triple t in G can then be calculated as:

$$score(t) = \frac{\sum_{q \in \Sigma} \frac{|T_{t,q}|}{|\bigcup_{t' \in G} T_{t',q}|}}{|\Sigma|}$$

The numerator of the score function is a calculation of how much t contributes to the overall knowledge, that is both original and entailed, about each $q \in \Sigma$. For example if t is one of only two triples that can entail new triples with q, that has a large positive influence on t´s score. However, if t can only entail one of 10 new triples with q, that will have a limited influence on t´s score. Additionally, we divide by $|\Sigma|$ to have $score(t) \in [0, 1]$.

Given a positive integer k, an optimal snippet S will be one that contains the k triples with the largest score value. Example 4.2.4 shows how to calculate the score for a small KG.

Example 4.2.4 (RBSG score)**.** Let $\Sigma = \{\texttt{UiO}, \texttt{University}\}$ and $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$ with T containing the triples:

t_1: $\langle \texttt{UiO}, \texttt{type}, \texttt{University} \rangle$

t_2: $\langle \texttt{University}, \texttt{subClassOf}, \texttt{EducationalInstitution} \rangle$

t_3: $\langle \texttt{UiO}, \texttt{city}, \texttt{Oslo} \rangle$

t_4: $\langle \texttt{UiB}, \texttt{county}, \texttt{Vestland} \rangle$

To score the triples, first materialize G. The only entailed triple will be $\langle \texttt{UiO}, \texttt{type}, \texttt{EducationalInstitution} \rangle$, using rdfs9 (see Appendix A) with t_1 and t_2 as the premises. Then calculate the sets $T_{t,q}$ for each $t \in T$ and $q \in \Sigma$:

$T_{t_1,\texttt{UiO}} = \{\langle \texttt{UiO}, \texttt{type}, \texttt{University} \rangle, \langle \texttt{UiO}, \texttt{type}, \texttt{EducationalInstitution} \rangle\}$

$T_{t_1,\texttt{University}} = \{\langle \texttt{UiO}, \texttt{type}, \texttt{University} \rangle\}$

$T_{t_2,\ \texttt{UiO}} = \{\langle \texttt{UiO}, \texttt{type}, \texttt{EducationalInstitution} \rangle\}$

$T_{t_2,\ \texttt{University}} = \{\langle \texttt{University}, \texttt{subClassOf}, \texttt{EducationalInstitution} \rangle\}$

$T_{t_3,\ \texttt{UiO}} = \{\langle \texttt{UiO}, \texttt{city}, \texttt{Oslo} \rangle\}$

$T_{t_3,\ \texttt{University}} = \{\}$

$T_{t_4,\ \texttt{UiO}} = \{\}$

$T_{t_4,\ \texttt{University}} = \{\}$

Lastly, calculate the scores:

$score(t_1) = (\frac{2}{3} + \frac{1}{2})/2 = 0.583$

$score(t_2) = (\frac{1}{3} + \frac{1}{2})/2 = 0.417$

$score(t_3) = (\frac{1}{3} + \frac{0}{2})/2 = 0.167$

$score(t_4) = (\frac{0}{3} + \frac{0}{2})/2 = 0$

Note that the triples in the KG are mainly scored based on how many entailments (containing a signature element) they can generate. Since entailments are generated using the RDFS rules, and all interesting rules contain a triple with property `domain`, `range`, `subPropertyOf` or `subClassOf`, KGs without these properties should not be summarized using RBSG. Running the materialization algorithm for such a KG would be unnecessary, because no new triples would be added, and the scoring would only be based on how many signature elements a triple contains.

4.2.4 Knowledge Graph Size Reduction

Assume we have a very large knowledge graph $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$ that we want to summarize using the RBSG approach. Since we only care about triples related to the signature Σ when materializing G, we can shrink G to only contain relevant triples, before we materialize it. Then we have a smaller KG to work with, and hence materialization should be less time consuming.

Start with an empty set T_s. Remember that $\Omega(t)$ is the set of elements that constitutes t. First we add all triples containing an element in Σ to T_s, that is if $q \in \Sigma$ and $q \in \Omega(t)$ for $t \in T$, then $t \in T_s$. Secondly we add all $t \in T$ containing an element in a triple in T_s. To make this task easier, we store all $x \in \Omega(t)$ for all $t \in T_s$ in a new signature Σ'. Then if $x \in \Sigma'$ and $x \in \Omega(t)$ for $t \in T$, we add t to T_s. This second step is repeated as long as new triples are added to T_s. When no new triples are added to T_s, we have a new smaller graph $G_s = \langle \mathcal{I}, \mathcal{L}, T_s \rangle$ to work with.

However, some elements occur in very many of the triples, or only provide very general information. These elements should not be added to Σ', in order to keep T_s as small as possible. For example the property `type` is a universal property that can be used with any subject of a triple. Therefore, if $\texttt{type} \in \Sigma'$, then most likely $T_s = T$, which we want to avoid. The following example illustrates the difference between $\texttt{type} \in \Sigma'$ and $\texttt{type} \notin \Sigma'$.

Example 4.2.5. Let $\Sigma = \{\texttt{UiO}\}$ and $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$ with T containing the triples:

t_1: $\langle \texttt{UiO}, \texttt{type}, \texttt{University} \rangle$

t_2: $\langle \texttt{UiB}, \texttt{type}, \texttt{University} \rangle$

t_3: $\langle \texttt{Apple}, \texttt{type}, \texttt{Fruit} \rangle$

Since $\Sigma = \{\texttt{UiO}\}$, $t_1 \in T_s$. If $\Sigma' = \{\texttt{UiO}, \texttt{University}\}$, $t_2 \in T_s$, which makes sense since both UiO and UiB are Universities. However, if $\Sigma' = \{\texttt{UiO}, \texttt{type}, \texttt{University}\}$, $t_2 \in T_s$ and $t_3 \in T_s$. t_3 is not related to Σ, and would not be chosen to be in a snippet, and hence does not need to be part of the materialization.

Let GP (general predicates) be a set of predicates from the RDF/RDFS vocabulary that are too general, and will cause unnecessary triples to be added to T_s. Similarly, let GO (general objects) be a set of objects from the Web Ontology Language (OWL) vocabulary that are also too general. If $x \in \Omega(t)$ for a $t \in T_s$, and $x \in$ GP or $x \in$ GO, then $x \notin \Sigma'$. GP and GO are defined as:

$$\begin{aligned}\text{GP} = \{&\texttt{rdf:type}, \texttt{rdf:Property}, \texttt{rdfs:subPropertyOf}, \texttt{rdfs:Class},\\ &\texttt{rdfs:subClassOf}, \texttt{rdfs:Resource}, \texttt{rdfs:domain}, \texttt{rdfs:range},\\ &\texttt{rdfs:label}\}\end{aligned}$$

$$\text{GO} = \{\texttt{owl:Class}, \texttt{owl:DatatypeProperty}, \texttt{owl:Thing}\}$$

Algorithm 3 describes how the size of a KG is reduced with respect to a signature.

Algorithm 3 Shrinking algorithm for KGs

Input: A knowledge graph $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$ and a non-empty signature Σ
Output: A knowledge graph G_s

$T_s \leftarrow \{\}$
$\Sigma' \leftarrow \Sigma$
$l \leftarrow 0$ ▷ Previous number of elements in Σ', initially 0
while $l < |\Sigma'|$ **do**
    $l \leftarrow |\Sigma'|$
    for $q \in \Sigma'$ **do**
        for $t \in T$ **do**
            if t contains q **then**
                add t to T_s
                for $x \in \Omega(t)$ **do**
                    if x is not in GP or GO **then**
                        add x to Σ'
                    end if
                end for
            end if
        end for
    end for
end while
$G_s \leftarrow \langle \mathcal{I}, \mathcal{L}, T_s \rangle$
return G_s

4.3 Embedding Based Snippet Generation (EBSG)

In this section we present the second approach for snippet generation, which is based on knowledge graph embedding (KGE). We let A in Algorithm 1 be an embedding method for a KG G. Given a signature Σ, the triples in G are then scored using the vector representation of the triples´ elements, and the vector representation of Σ's elements. We use RDF2Vec as the embedding method.

4.3.1 The RDF2Vec Tool

Resource Description Framework To Vector (RDF2Vec) is a tool made for transforming RDF data to real-valued vectors. It was first introduced in [RP16], but later significantly extended in [Ris+19]. The LOD movement has resulted in a large amount of available data on the Web, but this data is in graph form, usually RDF. Hence, it is not directly usable with data mining tools. RDF2Vec was therefore developed to transform RDF data, so it also can be used with embedding-based approaches.

RDF2Vec is an extension of Word2Vec [Mik+13], which is a popular word embedding method. In RDF2Vec the RDF data is first converted into a set of sequences of entities, using graph walks or Weisfeiler-Lehman Subtree RDF Graph Kernels. As more advanced walking techniques have been tested with good results, using graph walks has gained popularity, while the Weisfeiler-Lehman algorithm is now less popular. When the RDF data is converted to sequences, Word2Vec is used to learn embeddings from the sequences. Word2Vec uses Continuous Bag of Words which predicts a word from context words, or Skip-Gram which predicts the context words from a given word.

RDF2Vec was originally written in Java, but a python implementation was developed in [Ste+23], since Python is a more popular programming language than Java. The python implementation is called pyRDF2Vec and is based on the extended RDF2Vec approach presented in [Ris+19]. We used pyRDF2Vec in this thesis, and the package can be downloaded from GitHub. [1]

4.3.2 Snippet Generation with RDF2Vec

Let $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$ be a knowledge graph, $\Sigma = \{q_1, q_2, ..., q_n\}$ a signature where $q_i \in \mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$ for $i = 1, ..., n$, and k a positive integer. In this approach we follow Algorithm 1 with A being RDF2Vec. Using RDF2Vec we obtain a vector representation for all resources in G and all elements in Σ, and then we use these vectors to produce a similarity score for each triple in T. The snippet S will contain the k triples with the largest score.

Figure 4.1 shows the main parts of the EBSG approach. The input of RDF2Vec is a list of resources containing the elements of the signature and the elements of the triples of G. Some of the very general properties, like `type`, are discarded from the resource list since they do not need a vector representation (see Section 4.3.3). RDF2Vec also takes a KG as input. This KG can be a SPARQL endpoint, a file , or be created from scratch.

In RDF2Vec, a transformer takes the list of resources and the KG, together with a walking strategy, and outputs a vector representation of each resource.

[1] https://github.com/IBCNServices/pyRDF2Vec

These vectors can then be used to score the triples of the KG, and given a k we can find the top-k ranked triples, and output a snippet.

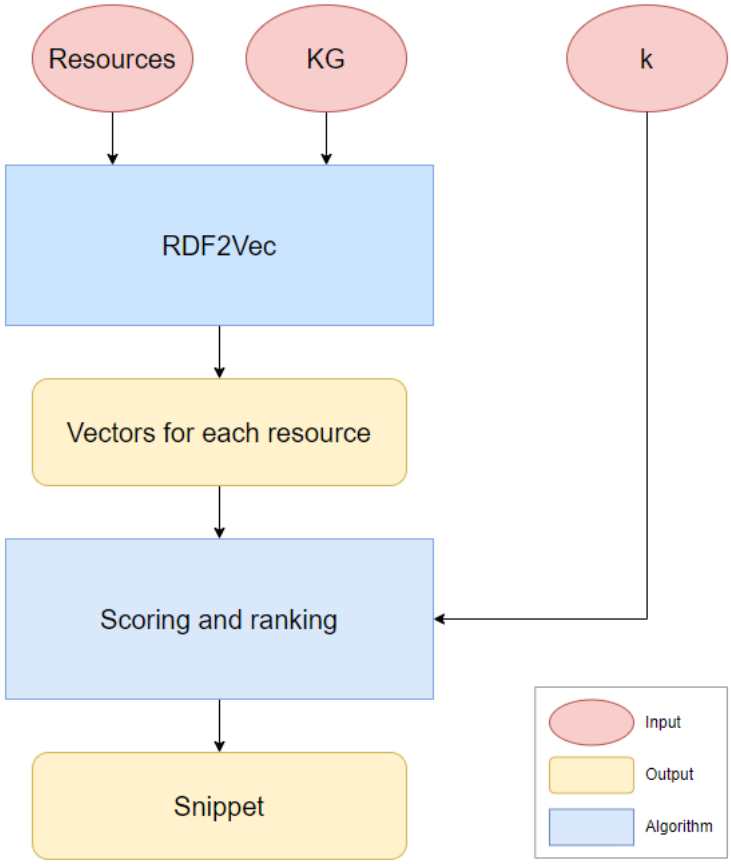

Figure 4.1: Diagram showing the EBSG approach

4.3.3 EBSG Score

The score of a triple $t = \langle s, p, o \rangle$ will depend on the score of each element s, p and o, which are scored based on relatedness to the signature Σ. Remember that $\Omega(t)$ is defined as the set of elements that constitute a triple, that is $\Omega(t) = \{s, p, o\}$. For each $x \in \Omega(t)$ we calculate how similar x is to each $q \in \Sigma$. As a similarity measure we use cosine similarity, which is calculated between two vectors. The cosine similarity of vectors $\vec{u}$ and $\vec{v}$ is given as:

$$\text{cosine}(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \; ||\vec{v}||}$$

Also note that cosine similarity is symmetric, so:

$$\text{cosine}(\vec{u}, \vec{v}) = \text{cosine}(\vec{v}, \vec{u})$$

For $x \in \mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$ we calculate cosine$(\vec{x}, \vec{q})$, where $\vec{x}$ is the vector representation of $x \in \Omega(t)$ and $\vec{q}$ is the vector representation of $q \in \Sigma$. On the other hand, if $x \in \mathcal{L}$, that is x is a literal, $\vec{x}$ is not produced because RDF2Vec does not natively incorporate literals [Pau23]. Literals were also ignored in the evaluation of RDF2Vec in [RP16] and [Ris+19], but pyRDF2Vec has an option for including literals [Ste+23]. However, using the option for literals does not directly produce an embedding for the literals. Therefore, for simplicity, if $x \in \mathcal{L}$ we let the similarity between x and $q \in \Sigma$ be 0. There are also several properties that are not particularly related to any other resource, for example `type` and `subject`. Let P^* denote this set of properties, such that if $x \in P^*$, then the similarity between x and $q \in \Sigma$ is also 0. A list of elements in P^* is given in Appendix B.

Let sim be the function that calculates similarity between two elements, so that for each pair $\{x, q\}$ where $x \in \Omega(t)$ and $q \in \Sigma$, we calculate $\text{sim}(x, q)$.

$$\text{sim}(x, q) = \begin{cases} \text{cosine}(\vec{x}, \vec{q}) & \text{if } x \in \mathcal{E} \cup \mathcal{P} \cup \mathcal{C} \\ 0 & \text{if } x \in \mathcal{L} \text{ or } x \in P^* \end{cases}$$

Each $x \in \Omega(t)$ will have $|\Sigma|$ sim scores, and t will have $3 \times |\Sigma|$ sim scores, hence there are multiple options for combining the sim scores to produce one score for t. We will explore the three score functions below:

1. $score_{\max}(t) = \sum_{x \in \Omega(t)} \max_{q \in \Sigma}(\text{sim}(x, q))$
2. $score_{\text{mean}}(t) = \sum_{x \in \Omega(t)} \left(\left(\sum_{q \in \Sigma} (\text{sim}(x, q)) \right) / |\Sigma| \right)$
3. $score_{\text{sum}}(t) = \sum_{x \in \Omega(t)} \sum_{q \in \Sigma} \text{sim}(x, q)$

We also consider three different types of signatures:

i $|\Sigma| = 1, \Sigma = \{e\}$, where $e \in \mathcal{E}$

ii $|\Sigma| > 1, \Sigma \subseteq \mathcal{E} \cup \mathcal{C}$

iii $|\Sigma| > 1, \Sigma \subseteq \mathcal{E} \cup \mathcal{C} \cup \mathcal{P}$

In Example 4.3.1, we present a sample KG to facilitate the comparison of score functions and signature types.

Example 4.3.1. Let $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$ be a KG where $T = \{t_1, t_2, ..., t_{11}\}$, and

$t_1 = \langle \texttt{UiO}, \texttt{type}, \texttt{University} \rangle$

$t_2 = \langle \texttt{UiO}, \texttt{city}, \texttt{Oslo} \rangle$

$t_3 = \langle \texttt{UiO}, \texttt{type}, \texttt{EducationalInstitution} \rangle$

$t_4 = \langle \texttt{UiB}, \texttt{type}, \texttt{University} \rangle$

$t_5 = \langle \texttt{UiB}, \texttt{type}, \texttt{EducationalInstitution} \rangle$

$t_6 = \langle \texttt{UiB}, \texttt{county}, \texttt{Vestland} \rangle$

$t_7 = \langle \texttt{Haukeland}, \texttt{affiliation}, \texttt{UiB} \rangle$

$t_8 = \langle \texttt{University}, \texttt{subClassOf}, \texttt{EducationalInstitution} \rangle$

The triples t_9, t_{10} and t_{11} states that `city`, `county` and `affiliation` have type `Property`, and are included solely for the purpose of obtaining a vector representation for these three properties using pyRDf2Vec.

Below we provide some of the similarity scores, which have been computed using pyRDF2Vec embeddings.

- $\text{sim}(\texttt{UiO}, \texttt{University}) = 0.044$
- $\text{sim}(\texttt{University}, \texttt{University}) = 1.0$
- $\text{sim}(\texttt{Oslo}, \texttt{University}) = 0.117$
- $\text{sim}(\texttt{city}, \texttt{University}) = -0.163$
- $\text{sim}(\texttt{UiO}, \texttt{Oslo}) = 0.072$
- $\text{sim}(\texttt{city}, \texttt{Oslo}) = 0.060$
- $\text{sim}(\texttt{Oslo}, \texttt{Oslo}) = 1.0$

First consider the case $\Sigma = \{e\}$, where $e \in \mathcal{E}$. Then the three score functions will give the same results:

$$score_{\max}(t) = score_{\text{mean}}(t) = score_{\text{sum}}(t) = \sum_{x \in \Omega(t)} \text{sim}(x, e)$$

Example 4.3.2 shows the calculated scores for t_1 and t_2 with $\Sigma = \{\texttt{Oslo}\}$.

Example 4.3.2. Let $\Sigma = \{\texttt{Oslo}\}$. For t_1 and t_2 we then have the scores:

- $score_{\max}(t_1) = score_{\text{mean}}(t_1) = score_{\text{sum}}(t_1) = 0.072 + 0 + 0.117 = 0.189$
- $score_{\max}(t_2) = score_{\text{mean}}(t_2) = score_{\text{sum}}(t_2) = 0.072 + 0.060 + 1.0 = 1.132$

Next, consider $|\Sigma| > 1$ and $\Sigma \subseteq \mathcal{E} \cup \mathcal{C}$. In this case the score functions may not produce identical results. If $score_{\max}$ is employed, the triples containing elements similar to at least one element in Σ will receive the highest ranking. On the other hand, utilizing $score_{\text{mean}}$ or $score_{\text{sum}}$ will rank triples related to the most elements in Σ first. The following example demonstrates that t_1 and t_2 are ranked differently depending on score function.

Example 4.3.3. Let $\Sigma = \{\texttt{Oslo}, \texttt{University}\}$. For t_1 we have the scores:

- $score_{\max}(t_1) = 0.072 + 0 + 1.0 = 1.072$
- $score_{\text{mean}}(t_1) = (0.072 + 0.044)/2 + (0 + 0)/2 + (0.117 + 1.0)/2 = 0.617$
- $score_{\text{sum}}(t_1) = (0.072 + 0.044) + (0 + 0) + (0.117 + 1.0) = 1.233$

and for t_2 we have the scores:

- $score_{\max}(t_2) = 0.072 + 0.060 + 1.0 = 1.132$

- $score_{\text{mean}}(t_2) = (0.072+0.044)/2+(0.060+(-0.163))/2+(1.0+0.117)/2 = 0.565$
- $score_{\text{sum}}(t_2) = (0.072+0.044)+(0.060+(-0.163))+(1.0+0.117) = 1.130$

Example 4.3.3 shows that when we used $score_{\text{max}}$ t_2 got a higher score and was therefore ranked before t_1, while $score_{\text{mean}}$ and $score_{\text{sum}}$ ranked t_1 better than t_2. When $score_{\text{max}}$ was used t_2 had the benefit of only considering that `city` is related to `Oslo`, but for the other score functions t_2 was penalized for `city` not being related to `University`.

For the same Σ, using $score_{\text{mean}}$ or $score_{\text{sum}}$ will result in the same ranking of the triples, since the $score_{\text{sum}}$ values will just be $|\Sigma|$ times larger than the $score_{\text{mean}}$ values. However, Example 4.3.4 indicates that $score_{\text{mean}}$ can be more affected by the size of Σ.

Example 4.3.4. Consider an additional triple $\langle$`Apple`, `type`, `Fruit`$\rangle$ in G from Example 4.3.1, and let $\Sigma = \{$`Oslo`, `University`, `Apple`$\}$ with cosine(r, `Apple`) = 0 for all resources r in G. For t_1 we have the scores:

- $score_{\text{mean}}(t_1) = (0.072+0.044+0)/3+(0+0+0)/3+(0.117+1.0+0)/3 = 0.411$
- $score_{\text{sum}}(t_1) = (0.072+0.044+0)+(0+0+0)+(0.117+1.0+0) = 1.233$

Comparing Example 4.3.4 to Example 4.3.3 shows that if Σ is extended with an element not particularly related to any resource in G, $score_{\text{sum}}$ could be unaffected, but $score_{\text{mean}}$ would be lower.

Lastly, consider $|\Sigma| > 1$ and $\Sigma \subseteq \mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$. Then an option is to distinguish properties from entities and classes when scoring a triple t, and only calculate similarity between two elements if they are both in $\mathcal{P}$ or both in $\mathcal{E} \cup \mathcal{C}$. Example 4.3.5 shows how to score t_7 this way, and also how to score t_7 by considering any pair of resource and signature element.

Example 4.3.5. Let $\Sigma = \{$`Oslo`, `University`, `city`$\}$. One option for scoring t_7 is to only calculate similarity between the two properties `affiliation` and `city`, and between the two entities in t_7 and the entity and class in Σ. The cosine similarities for t_7 and Σ are:

- cosine(`Haukeland`, `Oslo`) = 0.041
- cosine(`Haukeland`, `University`) = 0.006
- cosine(`Haukeland`, `city`) = 0.013
- cosine(`affiliation`, `Oslo`) = 0.085
- cosine(`affiliation`, `University`) = 0.137
- cosine(`affiliation`, `city`) = −0.134
- cosine(`UiB`, `Oslo`) = −0.108
- cosine(`UiB`, `University`) = 0.013
- cosine(`UiB`, `city`) = 0.051

Then the scores for t_7 are:

- $score_{\max}(t_7) = 0.041 + -0.134 + 0.013 = -0.080$
- $score_{\text{mean}}(t_7) = (0.041 + 0.006)/2 + (-0.134)/1 + (-0.108 + 0.013)/2 = -0.158$
- $score_{\text{sum}}(t_7) = (0.041 + 0.006) + (-0.134) + (-0.108 + 0.013) = -0.182$

However, if similarity can be calculated between any type of resource the scores for t_7 are:

- $score_{\max}(t_7) = 0.041 + 0.137 + 0.051 = 0.229$
- $score_{\text{mean}}(t_7) = (0.041 + 0.006 + 0.013)/3 + (0.085 + 0.137 + (-0.134))/3 + (-0.108 + 0.013 + 0.051)/3 = 0.035$
- $score_{\text{sum}}(t_7) = (0.041 + 0.006 + 0.013) + (0.085 + 0.137 + (-0.134)) + (-0.108 + 0.013 + 0.051) = 0.104$

Example 4.3.5 shows that if properties are only scored against other properties important similarities can be lost. The property `affiliation` in t_7 is for example more similar to the entity `Oslo` and the class `University`, than to the property `city`. Only comparing `affiliation` to `city` therefore causes t_7 to have a low score. Resources in G will thus not be restricted to only be scored against signature elements with the same type.

CHAPTER 5

Benchmark

An entity summarization benchmark consists of human-made ground-truth summaries that machine-generated summaries can be compared to. Evaluation with ground-truth summaries is a popular evaluation method for entity summarizers because it is easy to perform and and the results are reproducible. For example was RELIN [CTQ11] evaluated with its own benchmark, and in [Liu+20] they created a method independent benchmark that can be used to evaluate general-purpose entity summarizers.

Since benchmarks are a popular way to evaluate entity summarizers, we also wanted to use a benchmark for evaluation. However, existing benchmarks only cover signatures composed of entities, while the signatures in this thesis can contain properties and classes as well. Therefore we had to make a new benchmark that was compatible with our signature. In order to make this task easier, we started with an existing benchmark, and extended it to fit with our snippet generators.

As in Chapter 3, summary is used instead of snippet to distinguish entity summarization from snippet generation.

5.1 ESBM

The base for our benchmark is the **E**ntity **S**ummarization **B**ench**M**ark (ESBM) created in [Liu+20], which is made for single entity signatures. This is a large benchmark tested on entity summarizers like RELIN [CTQ11], FACES [GTS15] and BAFREC [KNB18]. ESBM contains summaries for 175 entities, where 125 entities are sampled from DBpedia and 50 entities are sampled from LinkedMDB. DBpedia and LinkedMDB are both real datasets that are popular to use in entity summarization.

Furthermore, for DBPedia they only considered the triples in the following dump files: *instance types*, *instance types transitive*, *YAGO types*, *mappingbased literals*, *mappingbased objects*, *labels*, *images*, *homepages*, *persondata*, *geo coordinates mappingbased*, and *article categories*. Additionally, the sampled entities all belong to one of the classes `Agent`, `Event`, `Location`, `Species`, and `Work`. For LinkedMDB the entities were sampled from `Film` and `Person`. To avoid that summarization would be a trivial task, the description of a sampled entity needed to contain at least 20 triples.

For each of the 175 entities, ESBM contains 12 human-made summaries, and more precisely, 6 people created one summary with the top-5 triples, and one summary with the top-10 triples. There were 30 participants in total, and

they were asked to create general-purpose summaries. ESBM can be found on GitHub.[1]

5.2 ESBM*, An Extension of ESBM

To make a snippet for our benchmark, we started with en entity summarized in ESBM. Then, from the entity´s description, we chose another entity, property, or class, and combined these two elements to make a signature of size two. Given this new two-element signature, we created a general purpose snippet that covered information about both of the signature elements. They were given the same weight, that is the information about each of the elements have equal importance. We only made snippets for two-element signatures, since having a large signature would make the creation of snippets a difficult and time consuming task. Additionally, since ESBM is a very large benchmark, we only considered the 125 entities from DBpedia. We used the same version of DBpedia they used in [Liu+20], which was the English version of DBpedia 2015-10, [2] and we only considered the triples in the dump files mentioned in Section 5.1. This new benchmark was named ESBM*, and is available on GitHub.[3]

5.2.1 Extended Signature

Let $\mathcal{E}_{\text{ESBM}}$ be the set of entities from DBpedia used as signatures in ESBM. Given an entity $e \in \mathcal{E}_{\text{ESBM}}$ and its ESBM summaries, let r be the element we combined with e to make a new signature. r had to be present in the summaries (except for some chosen class extensions), $r \neq e$, and $r \in \mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$. When choosing r we primarily looked for a resource that occurred multiple times in the same summary, but since that was rare we chose r based on which resources occurred multiple times in $Desc(e)$ or in most of the 6 top-5 summaries. However, note that properties like `type` and `subject` were popular in summaries, but since they are very general (everything has a type) they were never chosen for a signature. In fact, for properties, only the ones in the DBpedia ontology namespace were considered for a signature. They are more specific for a group of entities, and therefore more interesting. Additionally, when choosing r, we examined the set of triples containing r. Define this set as the description of r, denoted $Desc^*(r)$:

Definition 5.2.1 (Description of resource)**.** Let T be a set of triples. The description of a resource $r \in \mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$ is a subset of triples $t \in T$ where r appears in subject, predicate or object position of t. Denote the description of r as $Desc^*(r)$, such that:

$$Desc^*(r) = \{\langle r, p, o\rangle \in T\} \cup \{\langle s, r, o\rangle \in T\} \cup \{\langle s, p, r\rangle \in T\}$$

Note that this is a modification of the description of an entity defined in Definition 2.3.3, extended to work for properties and classes as well.

If $Desc^*(r)$ was too large to easily navigate, or small enough for entity summarization to be a trivial task, we chose a different resource. Our

[1] https://github.com/nju-websoft/ESBM

[2] http://downloads.dbpedia.org/wiki-archive/Downloads2015-10.html

[3] https://github.com/alvamh/RBSG-EBSG-Benchmark

two approaches do allow for any triple in a KG to be chosen for the snippet, so a small $Desc^*(r)$ is not actually a problem, however since we are working with a very large KG, choosing triples from $Desc^*(r)$ is a much easier and less time consuming job. Different tests on our approaches also show that the triples in the machine-generated snippets are from the set $Desc^*(q_1) \cup Desc^*(q_2) \cup ... \cup Desc^*(q_n)$, if the signature was $\Sigma = \{q_1, q_2, ..., q_n\}$.

For some of the ESBM entities with small descriptions, we were not able to find a resource r to make an extended signature with, because $Desc^*(r)$ was to small, r was a general property or r was a very large class. Therefore we were only able to make 99 extended signature, where 33 were extended with an entity, 33 with a property, and 33 with a class. For each of the 99 signatures, we made one top-5 snippet and one top-10 snippet, as in ESBM. However, in ESBM the top-5 summary was not required to be a subset of the top-10 summary, but we created the top-5 snippet by choosing triples from the top-10 snippet. In [Liu+20] they showed that the top-5 summary is usually a subset of the top-10 summary, so that will not constitute a large difference.

5.2.2 Triple Selection

To easily navigate and locate relevant triples in the DBpedia datasets we used Apache Jena, [4], which is, as stated on the homepage, 'A free and open source Java framework for building Semantic Web and Linked Data applications'. We first loaded the datasets into Apache Jena TDB, which is Jena´s RDF store. Then we ran Jena´s SPARQL server, Apache Jena Fuseki, against the TDB store, in order to have a local endpoint we could query. A set of triples to consider for a snippet were then easily obtained.

Given the signature $\Sigma = \{e, r\}$, we first created a top-10 snippet by adding triples to the ESBM summaries for e. Let $S_1, S_2, ..., S_6$ be the ESBM summaries for e and S_r be the extended snippet for Σ. The triples in S_r were chosen from $Desc(e) \cup Desc^*(r)$. Let t be a triple in $Desc(e) \cup Desc^*(r)$. We primarily wanted the triples in S_r to provide a relationship between e and r, so if $e \in \Omega(t)$ and $r \in \Omega(t)$, $t \in S_r$.

For the rest of the triples in S_r, about 50% were chosen from $Desc(e)$, and then primarily from $S_1 \cup S_2 \cup ... \cup S_6$. The triples most related to r were chosen. The other 50% were chosen from $Desc^*(r)$, and how they were chosen depends on whether r was an entity, a property or a class.

If r was an entity, ESBM summaries for a resource of similar type were used as inspiration. For example did summaries for a resource e_{ex}, where e_{ex} is a person, tend to include triples with the properties: `birthDate`, `deathDate`, `type`, and `subject`, hence if r was a person, we chose triples that contained these properties. However, we also wanted the subject, predicate and object of the triple to be as related to e as possible.

If r was a property p or a class c, $t \in Desc^*(r)$ was on the form $\langle s, p, o \rangle$ or $\langle s, \texttt{type}, c \rangle$ respectively. The goal was to find the triple with s and o most related to e. This was done by using $Desc(e)$, and checking whether s or o had some of the same features as e. The triples with s (and o) most related to e were chosen for S_r.

[4] https://jena.apache.org/

Lastly, if r was unfamiliar, we used the abstract of its DBpedia page, or Wikipedia page, to obtain the most important information about it, and chose triples based on this.

When the top-10 snippets were done, we created the top-5 snippets by choosing the 5 most important triples in the top-10 snippet.

Example 5.2.2 (Creating a snippet)**.** Let $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$ with T containing the triples:

t_1: $\langle \texttt{UiO}, \texttt{location}, \texttt{Oslo} \rangle$

t_2: $\langle \texttt{UiO}, \texttt{type}, \texttt{University} \rangle$

t_3: $\langle \texttt{UiO}, \texttt{location}, \texttt{Blindern} \rangle$

t_4: $\langle \texttt{UiO}, \texttt{foundingDate}, 1811\text{-}09\text{-}02 \rangle$

t_5: $\langle \texttt{UiO}, \texttt{numberOfStudents}, 28000 \rangle$

t_6: $\langle \texttt{OsloMet}, \texttt{type}, \texttt{University} \rangle$

t_7: $\langle \texttt{OlsoMet}, \texttt{location}, \texttt{Oslo} \rangle$

t_8: $\langle \texttt{Oslo}, \texttt{type}, \texttt{City} \rangle$

t_9: $\langle \texttt{Bergen}, \texttt{type}, \texttt{City} \rangle$

t_{10}: $\langle \texttt{UiB}, \texttt{location}, \texttt{Bergen} \rangle$

t_{11}: $\langle \texttt{UiB}, \texttt{type}, \texttt{University} \rangle$

Also let $S_{\text{orig}} = \{t_1, t_2, t_3, t_4, t_5\}$ be an original benchmark snippet for the entity `UiO`, that is for a signature with only one entity. Below we describe how we chose the extended signature, and made a snippet for ESBM*. Denote this snippet S_{ext}. The example includes how we made a snippet for a signature extended with first an entity, then a property, and lastly a class. The example is for top-5 snippets.

Entity

To make an extended signature Σ with two entities, we can for example choose $\Sigma = \{\texttt{UiO}, \texttt{Oslo}\}$. Then the triple t_1 should be in the snippet. t_2 and t_8 should also be in the snippet, because presenting the type of an entity was popular in the ESBM summaries. t_3 and t_7 were also chosen for the snippet, because Blindern is in Oslo, and OlsoMet is, as UiO, a university in Oslo. Then $S_{\text{ext}} = \{t_1, t_2, t_3, t_7, t_8\}$.

Property

To make an extended signature Σ with one entity and one property, we can for example choose $\Sigma = \{\texttt{UiO}, \texttt{location}\}$. Then t_1 and t_3 shuld be in the snippet. t_7 is also very related to the signature since OsloMet and UiO are very related (they are both universities in Oslo), and should be in the snippet. For the last two triples we chose one with `UiO` and one with `location`, for example t_2 and t_{10}. Then $S_{\text{ext}} = \{t_1, t_2, t_3, t_7, t_{10}\}$.

Class

To make an extended signature Σ with one entity and one class, we consider two options: $\Sigma = \{\texttt{UiO}, \texttt{University}\}$ and $\Sigma = \{\texttt{UiO}, \texttt{City}\}$. First, if $\Sigma = \{\texttt{UiO}, \texttt{University}\}$, then t_2 should be in the snippet. We also chose t_6 and t_5, since `OsloMet` is very related to the signature, and `numberOfStudents` is very related to `University`. For the last two triples we chose one with `UiO` and one with `University`, for example t_1 and t_{11}. Then $S_{\text{ext}} = \{t_1, t_2, t_5, t_6, t_{11}\}$.

Secondly, if $\Sigma = \{\texttt{UiO}, \texttt{City}\}$, then no triples with both entities exist. Then we chose triples with one element in the signature, and other elements related to the signature. Therefore t_1 should be in the snippet and t_8 should be in the snippet. We also added t_3 and t_9. Lastly, we added a triple with some important information about one of the entities, for example t_2. Then $S_{\text{ext}} = \{t_1, t_2, t_3, t_8, t_9\}$.

5.3 ESBM* Statistics

The full DBpedia knowledge graph that consists of the triples in the 11 dump files mentioned in Section 5.2, and the DBpedia 2015-10 ontology, contains 163598792 triples. The dump files contain a total of 163568474 triples, and the ontology contains 30318 triples. Since this is a very large KG, we chose the triples in the extended signature from the descriptions of the signature elements. The descriptions ranges between around 30 triples for the smallest entity descriptions, up to almost 2000000 triples for the most general properties.

Let $\mathbb{S}_{\{e,e\}}$ be the set of ESBM* snippets created with a signature with two entities, $\mathbb{S}_{\{e,p\}}$ be the set of benchmark snippets created with a signature with one entity and one property, and $\mathbb{S}_{\{e,c\}}$ be the set of benchmark snippets created with one entity and one class. We have $|\mathbb{S}_{\{e,e\}}| = |\mathbb{S}_{\{e,p\}}| = |\mathbb{S}_{\{e,c\}}| = 33$.

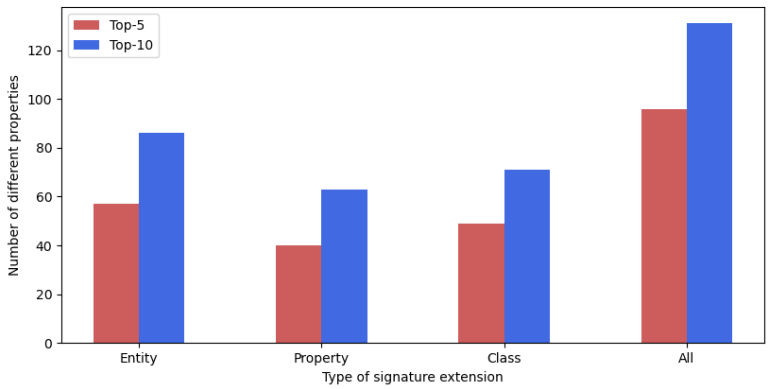

Figure 5.1: Number of different properties used in the benchmark snippets, for signatures extended with an entity, property, and class, and for all signatures.

Figure 5.1 shows how many different properties the ESBM* snippets collectively contain, divided in top-5 and top-10 snippets. A total of 96 different properties are used in the top-5 snippets, and for top-10 snippets the number is

131. Figure 5.1 also shows the difference between $\mathbb{S}_{\{e,e\}}$, $\mathbb{S}_{\{e,p\}}$ and $\mathbb{S}_{\{e,c\}}$. The snippets in $\mathbb{S}_{\{e,e\}}$ contain more different properties than the snippets in $\mathbb{S}_{\{e,p\}}$ and $\mathbb{S}_{\{e,c\}}$. This is not surprising since a snippet in $\mathbb{S}_{\{e,p\}}$ will contain multiple triples with the same property (the property in the signature) and a snippet in $\mathbb{S}_{\{e,c\}}$ will contain multiple triples with the property `type`.

Table 5.1 gives a closer look at which properties are the most popular in the snippets. It shows the properties present in the most snippets in the sets $\mathbb{S}_{\{e,e\}}$, $\mathbb{S}_{\{e,p\}}$, and $\mathbb{S}_{\{e,c\}}$, for top-5 and top-10 snippets. The number of snippets with the property in it is also presented. Note that each set of snippets contains 33 triples. For all three sets the property `type` is the most popular. For top-10 snippets, `type` is in all snippets in $\mathbb{S}_{\{e,e\}}$ and $\mathbb{S}_{\{e,c\}}$, and 85% of the snippets in $\mathbb{S}_{\{e,p\}}$. This is expected since all the signatures contain an entity, all entities are an instance of at least one class, and such information is popular to have in the snippets.

Top-5			Top-10		
Entity	Property	Class	Entity	Property	Class
type (26)	type (15)	type (33)	type (33)	type (28)	type (33)
country (8)			subject (18)	family (6)	subject (11)
			country (11)	genus (6)	
			birthDate (10)	isPartOfMilitaryConflict (6)	
			location (8)		
			deathDate (6)		

Table 5.1: The most popular properties in the snippets created with different signature extensions. Number of snippets with the property in it is given in the parenthesis.

For the top-5 snippets, only `country` with its presence in 24% of the snippets in $\mathbb{S}_{\{e,e\}}$, stands out as a somewhat popular property. An explanation is that `country` can be used with multiple types of entities as subject, for example people, cities, mountains and infrastructure.

For the top-10 snippets, `subject` is the second most popular property in the snippets in $\mathbb{S}_{\{e,e\}}$ and $\mathbb{S}_{\{e,c\}}$. `subject` is, like `type`, a universal property, and most entities have a subject relationship to some category. However, `subject` is not popular for snippets in $\mathbb{S}_{\{e,p\}}$. More triples contain the property in the signature, or properties related to that property, and for the rest of the triples type relationships is favored over subject relationships. For snippets in $\mathbb{S}_{\{e,e\}}$ one third of the snippets contain the properties `country` and `birthDate`. For the rest of the properties in Table 5.1, they are only used in around 20% of the snippets, showing that except for `type` (and `subject`), the snippets does not share many properties.

Next we look at the relationship between a snippet and the signature. Let S denote a snippet and Σ denote the signature used to create S. Σ is on the form $\Sigma = \{e, r\}$, where $e \in \mathcal{E}$ is an entity summarized in ESBM, and $r \in \mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$ is the resource used to extend the signature. Let $m_{\geq 1}$ be the number of snippets that contains one or more triples with both its signature´s elements.

$$m_{\geq 1} = |\{S | t \in S \text{ and } e, r \in \Omega(t)\}|$$

Also let $m_{\geq 2}$ be the number of snippets that contain two or more triples with both its signature´s elements.

$$m_{\geq 2} = |\{S | t_1, t_2 \in S \text{ and } e, r \in \Omega(t_1) \text{ and } e, r \in \Omega(t_2) \text{ and } t_1 \neq t_2\}|$$

Let n be the number of snippets in a set of snippets. Table 5.2 shows the ratios $m_{\geq 1}/n$ and $m_{\geq 2}/n$ for the three sets of snippets $\mathbb{S}_{\{e,e\}}$, $\mathbb{S}_{\{e,p\}}$, and $\mathbb{S}_{\{e,c\}}$, and for the set of all ESBM* snippets.

Snippet size	Top-5				Top-10			
Snippet set	$\mathbb{S}_{\{e,e\}}$	$\mathbb{S}_{\{e,p\}}$	$\mathbb{S}_{\{e,c\}}$	All	$\mathbb{S}_{\{e,e\}}$	$\mathbb{S}_{\{e,p\}}$	$\mathbb{S}_{\{e,c\}}$	All
$\boldsymbol{m_{\geq 1}/n}$	33/33	33/33	27/33	93/99	33/33	33/33	27/33	93/99
$\boldsymbol{m_{\geq 2}/n}$	4/33	21/33	0/33	25/99	4/33	21/33	0/33	25/99

Table 5.2: $m_{\geq 1}/n$ and $m_{\geq 2}/n$ for the three different sets of snippets, and for the set of all benchmark snippets.

Table 5.2 shows that $m_{\geq 1}/n$ and $m_{\geq 2}/n$ are the same for top-5 and top-10 snippets, for all snippet sets. This shows that among the triples in a top-10 snippet, the triples containing both signature elements are considered the most important. For $\mathbb{S}_{\{e,e\}}$ and $\mathbb{S}_{\{e,p\}}$, all snippets contained at least one triple with both signature elements, since $r \in \Omega(t)$ for a $t \in Desc(e)$. However, for $\mathbb{S}_{\{e,c\}}$, only 82% of the snippets did. That is because for some signatures belonging to snippets in $\mathbb{S}_{\{e,c\}}$, no triple existed with both the signature´s elements.

For snippets in $\mathbb{S}_{\{e,p\}}$, 64% of the snippets contained two or more triples with both signature elements. Since a property chosen for a signature was chosen because it was a popular property in the triples in $Desc(e)$, this is not surprising. For some of the signatures belonging to snippets in $\mathbb{S}_{\{e,p\}}$, 6-8 triples existed with both signature elements.

Snippet size	Top-5				Top-10			
Snippet set	$\mathbb{S}_{\{e,e\}}$	$\mathbb{S}_{\{e,p\}}$	$\mathbb{S}_{\{e,c\}}$	All	$\mathbb{S}_{\{e,e\}}$	$\mathbb{S}_{\{e,p\}}$	$\mathbb{S}_{\{e,c\}}$	All
$\boldsymbol{CT(S,e)}$	3.21	4.12	3.15	3.49	5.45	6.73	5.45	5.88
$\boldsymbol{CT(S,r)}$	2.97	3.21	2.64	2.94	5.73	5.85	5.30	5.63

Table 5.3: Average $CT(S,e)$ and $CT(S,r)$ for different sets of snippets, and for the set of all benchmark snippets.

To analyze the distribution of triples between the two elements in a signature, let $CT(S,q)$ be the number of triples in a snippet S that contain $q \in \Sigma$, where Σ is the signature belonging to S.

$$CT(S,q) = |\{t | t \in S \text{ and } q \in \Omega(t)\}|$$

Table 5.3 shows the average $CT(S,e)$ and $CT(S,r)$ for the snippets in $\mathbb{S}_{\{e,e\}}$, $\mathbb{S}_{\{e,p\}}$, and $\mathbb{S}_{\{e,c\}}$, and also for all snippets in ESBM*. The signatures are on the form $\{e,r\}$, where e is the original entity from ESBM and r is the signature extension we chose. Note that since a triple can contain both e and r, the average $CT(S,e)$ and average $CT(S,r)$ can sum to more than 5 or 10.

For both top-5 and top-10 snippets, triples with the original entity e are a little more popular in the snippets, except for in top-10 snippets in $\mathbb{S}_{\{e,e\}}$. Choosing related triples for a snippet is easier for entities, since they have shorter descriptions, which could be the reason for this. The largest difference between e and the extension r is for snippets in $\mathbb{S}_{\{e,p\}}$. One reason is that more information was needed to understand the entity than the property.

5.4 Discussion

Since ESBM* are made by only one person, it reflects an individual opinion about which triples are important. In ESBM there are six top-5 summaries and six top-10 summaries, and in [Liu+20] they calculated the overlap between the six summaries to be 1.99 triples and 5.42 triples, respectively. We can therefore assume that about 50% of the triples we chose for the snippets might have been chosen differently by someone else.

Also, we created the approaches before ESBM*, and knew which triples they would favor in a snippet, which could have affected the benchmarks. For example, let a signature be $\Sigma = \{\texttt{SriLankanCivilWar}, \texttt{MiltitaryConflict}\}$, and two triples in the KG be:

t_1: $\langle \texttt{SriLankanCivilWar}, \texttt{type}, \texttt{MilitaryConflict} \rangle$

t_1: $\langle \texttt{BattleOfSampur}, \texttt{isPartOfMilitaryConflict}, \texttt{SriLankanCivilWar} \rangle$

Independent of our approaches, only t_2 would be chosen for a top-5 benchmark snippet, because the predicate and object give the same information as t_1 provides, and there are limited space in a top-5 snippet. However, knowing that both approaches would have t_1 in a snippet, both t_1 and t_2 were chosen to be in the benchmark snippet.

CHAPTER 6

Evaluation

In this chapter we evaluate the two approaches presented in Chapter 4. First we evaluate the materialization algorithm by analyzing the size of the materialized graph. Secondly, we evaluate our two approaches by analyzing the running time of the materialization algorithm and the embedding process with RDF2Vec, as well as the belonging score functions. Lastly we evaluate the quality of the snippets generated by our approaches using ESBM*, which was described in Chapter 5.

6.1 Evaluation of Graph Size and Running Time

For the first evaluation of RBSG and EBSG we performed an experiment were we generated 100 snippets, given 100 different signatures, for each of the signature sizes $|\Sigma| = 1$, $|\Sigma| = 5$ and $|\Sigma| = 10$. We timed the different parts of the two approaches for each of the $100 \times 3 = 300$ executions, and we also reported the size of the materialized graphs.

For RBSG, we ran the experiment with three different signature types for comparison:

1. Random signature
2. Seed signature
3. No signature

A signature contained at least one element from the class $\mathcal{E}$. This element was chosen randomly. For the random signature, the rest of the elements were also chosen randomly, but from the set $\mathcal{E} \cup \mathcal{P} \cup \mathcal{C}$. For the seed signature, the rest of the elements were chosen in relation to the previously mentioned element from $\mathcal{E}$. In the case of no signature, we ran the materialization algorithm with some modifications to work without a signature, and also omitted the scoring part.

Since signature type does not affect the running time of any part of EBSG, this approach was only evaluated with a random signature in the running time evaluation. However, two score types were evaluated for this approach, $score_{\max}$ and $score_{\text{mean}}$.

The KG we used for the experiments contained triples from the DBpedia 2015-10 datasets (English version) and the DBpedia 2015-10 ontology. Since the DBpedia datasets and the ontology collectively contain 163598792 triples, we

only used a subset of these triples in our KG. The subset contained 45925 triples, taken from the dump files *instance types* and *mappingbased objects*, as well as the ontology. The KG also contained 262 triples on the form $\langle p, \texttt{type}, \texttt{Property} \rangle$ where p is a property in the DBpedia subset. This was done to be able to use pyRDF2Vec to find vector representations for the properties as well. The KG contained a total of 46187 triples.

The experiments were ran on a computer with an 11th Gen Intel Core i5-1135G7 2.40GHz processor and 8,00 GB of memory. The approaches were implemented in Python 3, specifically we ran Python 3.9.12.

For the rest of this section, let G be the name of the KG, and therefore $|G| = 46187$. Also let Σ denote a signature, and G' denote the materialized version of G given Σ. Lastly, let $\Sigma \subset \mathcal{I}$ for $G = \langle \mathcal{I}, \mathcal{L}, T \rangle$.

6.1.1 Size of Materialized Graph

Signature type	Random			Seed		
$\lvert\Sigma\rvert$	1	5	10	1	5	10
min	0	14	30	0	11	18
max	53	68	389	37	8198	8295
mean	7.1	29.37	57.01	7.82	1479.59	2242.25

Table 6.1: Number of new triples generated by materializing G with different signature types and different sizes of Σ.

Table 6.1 presents the minimum, maximum, and mean number of new triples generated by running the materialization algorithm on G with different Σ. There is a clear connection between the number of new triples and the size of Σ. When Σ increases, so does the minimum, and mean number of new triples. This is expected since a larger $|\Sigma|$ value means a larger chance that an entailed triple will contain an element $q \in \Sigma$, and hence a larger chance that the entailed triple will be in G'.

Furthermore, Table 6.1 shows that the mean number of new triples generated are 50 times larger for a seed signature than a random signature when $|\Sigma| = 5$ and 40 times larger for a seed signature than a random signature when $|\Sigma| = 10$ (note that a seed signature with $|\Sigma| = 1$ is just a randomly chosen element, and therefore the same as a random signature). Using a seed signature increases the chance that a new generated triple can be used to create more new triples in the next recursion step.

There is also a larger difference between minimum and maximum number of new triples for random signature and seed signature. If the seed signature does not contain the correct related elements, that is elements occurring in an entailed triple, the number of new triples will be no better than for a random signature. However, if the seed signature does contain the correct elements, the number of new generated triples has potential to be 120 times more than for a random signature for $|\Sigma| = 5$.

We also ran the materialization algorithm without a signature, that is without any restriction to which rule conclusions will be in G'. Then the

number of new generated triples in G' were $|G' \setminus G| = 151123$, which is 3.27 times as large as the original graph G.

6.1.2 Running Time of Algorithms

	Materialization						Score					
Signature type	Random			Seed			Random			Seed		
$\|\Sigma\|$	1	5	10	1	5	10	1	5	10	1	5	10
min	0.72	2.17	3.20	0.70	1.62	2.42	0.32	0.68	1.01	0.33	0.69	1.17
max	2.17	5.53	8.60	2.37	6.09	9.96	1.12	2.95	5.06	1.10	3.55	6.02
mean	1.55	3.19	4.82	1.48	2.97	4.50	0.51	1.43	2.25	0.48	1.39	2.45

Signature type	No signature
min	4.45
max	9.76
mean	4.93

Table 6.2: Running time (s) for different parts of the RBSG approach for different signature types and sizes of Σ.

	RDF2Vec			Score MAX			Score MEAN		
$\|\Sigma\|$	1	5	10	1	5	10	1	5	10
min	41.18	42.39	42.2	8.55	30.25	65.87	8.63	30.38	64.07
max	45.42	46.79	51.10	17.63	56.51	103.08	17.27	57.20	105.39
mean	43.48	43.76	43.95	12.80	43.68	80.71	12.78	43.81	80.82

Table 6.3: Running time (s) for different parts of the EBSG approach for different sizes of Σ.

Table 6.2 presents the running time in seconds for the materialization algorithm and the combined running time for scoring, ranking and snippet creation (called Scoring in the table), also in seconds. The running times are calculated for a random signature and a seed signature, and running time of the materialization algorithm is also calculated in the case of no signature. For both random and seed signature the mean time it takes to materialize a KG increases when the size of Σ increases. This is not surprising, considering that each element in an entailment has to be tested for a match in Σ, and larger $|\Sigma|$ means more tests. Table 6.1 also showed that a larger $|\Sigma|$ resulted in more new triples. Unless all new triples were generated in the same recursive loop, more new triples would mean more recursive loops, which takes time. However, since the mean time for running the materialization algorithm without a signature is only 0.11 and 0.43 seconds longer than running it with a random signature and seed signature, respectively, testing for a match in Σ is probably what slows down the materialization. If not, running materialization without a signature should be much slower, considering 151123 new triples were generated. Comparing the mean time of materialization without a signature to materialization with a random signature of size 10, there is only a 2.23% decrease in time, but a 99.96% decrease in number of new triples.

Comparing the mean running time for the materialization algorithm for random and seed signature, also support the claim that the size of Σ affects the running time more than the number of new generated triples. The running time when using a seed signature actually decreased with 7% compared to a random signature for both $|\Sigma| = 5$ and $|\Sigma| = 10$, but the number of new triples increased with almost 5000% and 4000%, respectively. This also shows that using a seed signature instead of a random signature, has only a small effect on running time, but a large effect on $|G'|$.

Seed max time being larger than random max time could be because the entailed triples are related to more elements in the signature, and there are a larger chance that the new triples can be used in the next recursive loop of the algorithm, causing it to run for longer.

Furthermore, Table 6.2 also presents the running time for the second part of RBSG: the scoring and snippet generation. The mean times are similar for random and seed signature, which makes sense since the same set of triples, that is all triples in G, have to be scored regardless of signature type. Also regardless of signature type, the the mean running time for scoring is around 1/3 of the materialization time for $|\Sigma| = 1$, and around 1/2 of the materialization time for $|\Sigma| = 5$ and $|\Sigma| = 10$. Scoring triples when $|\Sigma| = 1$ is faster because many triples will be scored 0, since they could not generate new triples containing the signature element.

Table 6.3 presents the running time in seconds for the embedding with pyRDF2Vec and the combined running time for scoring, ranking and snippet creation, also in seconds. The running time for pyRDF2Vec includes making a list with elements to create an embedding for, setting up a transformer with a walking strategy, and create vectors for each element in the list. For the scoring part of EBSG, two score functions were evaluated, but they are used only one at a time in EBSG, not together. The table shows that the two score functions have very similar running times for each size of $|\Sigma|$, so choosing one over the other will not affect the total time for EBSG.

Furthermore, the running time of pyRDF2Vec does not increase as much as the running time of the materialization algorithm, when $|\Sigma|$ increases. However, looking at the mean time for pyRDf2Vec it suggests a small increase in running time when $|\Sigma|$ increases, but it is a less than 1% increase in time for each increase in size. Considering that the embedding of triple-elements are independent of the signature, this increase in time should not be significant.

Comparing the mean running time for the pyRDf2Vec part of EBSG and the scoring part of EBSG, when $|\Sigma| \geq 5$ the scoring will be the slowest part of EBSG. For $|\Sigma| = 5$, the two parts of EBSG will have approximately the same running time, and for $|\Sigma| = 1$ the scoring is fast compared to embedding. In EBSG the triples are scored based on cosine similarity between triple-elements and signature-elements. Increasing the size of Σ by one element, means that another cosine similarity score has to be calculated for each triple-element, which will slow down the scoring. For $|\Sigma| = 10$, the mean running time for scoring is almost twice as large as for embedding.

Moreover, comparing the results in Table 6.2 and Table 6.3, the materialization algorithm is faster than pyRDF2Vec. Comparing mean running time for $|\Sigma| = 10$, pyRDF2Vec is almost 10 times slower than materialization (with random signature). One reason is that creating an embedding is a more complicated task than locating triples that can be premises in RDFS rules. The scoring

for RBSG is also faster than the scoring for EBSG. In EBSG, cosine scores have to be calculated before the triples can be scored, while in RBSG the triples can be scored immediately. During materialization, elements are added to a dictionary used to calculate RBSG score. The time spent on adding elements to this dictionary is subtracted from the materialization time. Including this time would result in at least a two second increase in total running time for RBSG, and at most a 15 second increase. Even with a 15 second increase, the longest total time for RBSG (seed signature and $|\Sigma| = 10$), would be less than the shortest total time for EBSG (Score MAX and $|\Sigma| = 1$)

Lastly, an earlier version of RBSG also included the shrinking algorithm. The intention behind the shrinking algorithm was to decrease the time spent on materialization, but an experiment showed that the shrinking algorithm was to slow to be helpful. The materialization algorithm is also fairly fast on its own, so the shrinking algorithm was disregarded. The results from the experiment are presented and discussed in Appendix C.

6.2 Evaluation with ESBM*

In this section we evaluate both approaches using an intrinsic method, that is we directly measures the quality of our machine-generated snippets by comparing them with human-made ground-truth snippets from our benchmark ESBM*.

Let S_m be a machine-generated snippet and S_h a human-made snippet from ESBM*. Three popular information retrieval metrics for evaluation are:

$$Precision = \frac{|S_m \cap S_h|}{|S_m|}$$

$$Recall = \frac{|S_m \cap S_h|}{|S_h|}$$

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

In our case $|S_m| = |S_h|$, hence $Precision = Recall = F1$. However, some entity summarizers have the restriction that $|S_m| \leq k$ instead of $|S_m| = k$, where k is the predefined entity summary size restriction. Then $|S_m| \neq |S_h|$ is possible, which would cause $Precision \neq Recall$, and hence $F1$ score should be used. Therefore we also use the $F1$ score as the evaluation metric.

We generated 99 snippets for each of our two approaches, using the signatures from ESBM*. Since the DBpedia datasets we used are very large, we created smaller individual datasets for each signature, and generated snippets from these smaller datasets. A dataset is composed of triples t, where at least one $x \in \Omega(t)$ has maximal depth 2 from a $q \in \Sigma$. Then we calculated the $F1$ score for each pair of machine-generated snippets and the ESBM* snippets. The results are presented in Table 6.4.

Extension type	Entity	Property	Class	All
$\boldsymbol{k = 5}$	0.33	0.44	0.24	0.33
$\boldsymbol{k = 10}$	0.20	0.26	0.20	0.22

Extension type	Entity		Property		Class		All	
Score type	max	mean	max	mean	max	mean	max	mean
$\boldsymbol{k = 5}$	0.15	0.08	0.33	0.13	0.07	0.05	0.18	0.09
$\boldsymbol{k = 10}$	0.13	0.08	0.30	0.14	0.07	0.05	0.17	0.09

Table 6.4: Mean $F1$ score for snippets generated with RBSG (top) and EBSG (bottom), for different signature extension types.

Extension type	Entity		Property		Class		All	
Score type	max	mean	max	mean	max	mean	max	mean
$\boldsymbol{k = 5}$	0.15	0.08	0.36	0.12	0.08	0.07	0.20	0.09
$\boldsymbol{k = 10}$	0.12	0.08	0.27	0.12	0.08	0.06	0.16	0.09

Table 6.5: Mean $F1$ score for snippets generated with EBSG using a materialized graph, for different signature extension types.

Table 6.4 presents the mean $F1$ score for snippets generated with RBSG and EBSG, for snippets in $\mathbb{S}_{\{e,e\}}$ (the set of snippets created with a two-entity signature), $\mathbb{S}_{\{e,p\}}$ (the set of snippets created with a property-extended signature), and $\mathbb{S}_{\{e,c\}}$ (the set of snippets created with a class-extended signature), as well as the overall mean score. For both approaches the snippets in $\mathbb{S}_{\{e,p\}}$ had the best scores, then the snippets in $\mathbb{S}_{\{e,e\}}$, and lastly the snippets in $\mathbb{S}_{\{e,c\}}$ had the worst scores. For most of the property-extended signatures, at least two triples in the KG contained both signature elements. These triples are typically included in ground-truth snippets because they are obviously very related to the signature, and they are also given a high score in RBSG and EBSG. Therefore, the more of these triples, the larger agreement between S_h and S_m, and hence a higher $F1$ score. For many of the two-entity signatures, two triples also existed with both signature elements, but for class-extended signature at most one triple existed with both signature elements. Therefore the triples in the snippets in $\mathbb{S}_{\{e,c\}}$ are more difficult to chose, and the human-made snippets are open for more individual differences.

Furthermore, RBSG performed better than EBSG for all signature extensions and both k-values, except for $\mathbb{S}_{\{e,p\}}$ with $k = 10$. In RBSG the KG is materialized, which produces a richer graph with more knowledge. To investigate how much RBSG benefitted from the richer graph compared to EBSG, we computed new $F1$ scores for EBSG, but with materialized graphs as input to RDF2Vec. Then, in RDF2Vec the embeddings were generated from a richer graph, which could have improved the quality of the embeddings, and therefore snippets, and hence resulted in a larger $F1$ score. However, Table 6.5 shows that this is not the case. The difference in mean $F1$ scores between Table 6.5 and Table 6.4 are at most 0.03, and mainly 0 or 0.01, showing that a materialized

graph does not improve the performance of EBSG. The only snippet set with a small improvement in $F1$ score is $\mathbb{S}_{\{e,c\}}$, which could be because four of the six RDFS rules used in materialization, entail triples with class knowledge.

Another reason why RBSG performed better than EBSG could be that in RBSG a triple t is scored in full, while in EBSG each $x \in \Omega(t)$ is scored separately. Therefore, if one $x \in \Omega(t)$ is unrelated to some (or all) $q \in \Sigma$, t is punished when scored in EBSG, but not in RBSG. Example 4.3.3 showed that $t_2 = \langle \texttt{UiO}, \texttt{city}, \texttt{Oslo} \rangle$ was punished for `city` not being related to $\texttt{University} \in \Sigma$ when scored in EBSG, even though `UiO` and `Oslo` were related to Σ. On the other hand, in RBSG, if `UiO` and `Oslo` were the main reason why t_2 was good for reasoning, then `city` would not have affect the score because t_2 would have been scored in full.

In addition, the KG used for each signature contained triples t where one or more $x \in \Omega(t)$ is in at most depth 2 from a $q \in \Sigma$. Having depth 3 or depth 4 instead might have been better, because more relationships between resources would have been included, but then the benchmark evaluation would have been very slow, because the KG would have been larger.

RBSG and EBSG cannot be compared to any of the entity summarizers evaluated with ESBM, because the entity summarizers only work with entities in the signature. Also, ESBM only contains summaries for single entities, so the results in Table 6.4 are not directly comparable to the results in [Liu+20]. Nevertheless, for $k = 5$ RBSG performed better on ESBM* for two-entity signatures and property-extended signature, than every entity summarizer performed on ESBM. For RBSG with $k = 10$ and EBSG with both k value, the results from the ESBM evaluation shows that both RBSG and EBSG have room for improvement. RELIN [CTQ11] had the lowest $F1$ score with $F1 = 0.455$ for $k = 10$ in the ESBM evaluation, which is significantly better than any $F1$ score for RBSG and EBSG for $k = 10$. However, the number of triples containing a property or a class can greatly exceed the number of triples containing an entity, making the summarization of a property or a class more difficult.

Lastly, RBSG favors ontology knowledge, like subclass and subproperty relationships, in the snippets, since every RDFS rule includes a triple carrying ontology knowledge. However, the human-made ESBM* snippets typically do not contain ontology-triples, because ontology knowledge is more interesting for a machine than a human. This also explains why the $F1$ score is not better for RBSG.

CHAPTER 7

Conclusions and Future Work

Many approaches for summarizing a KG given an entity or a collection of entities have been developed over the last years, however summarizing a KG given a set of entities, properties and classes had, to the best of our knowledge, not been explored yet. In this thesis we proposed two approaches for KG summarization given such a set: RBSG and EBSG. We scored each triple in the KG, and given a size restriction k, we created a snippet for the KG from the k triples with the highest score. For RBSG this score was based on reasoning, or more specifically, materialization with RDFS rules. For EBSG the score was based on KGE, and we chose RDF2Vec as the embedding method.

The experimental results regarding running time showed that RBSG is fairly fast, but the running time increases when the size of the signature increases. The materialization algorithm is the slowest part, but multiple techniques exist to improve the performance of materialization. For example has parallelization techniques been used in some approaches to make materialization with RDFS rules more efficient [Sub+16; Tsa+21; Urb+10]. Including parallelization in our materialization algorithm could be explored more in the future, in order to decrease the running time of RBSG.

On the other hand, EBSG has room for improvement regarding running time. The embeddings require a lot of time to be created, so reducing the number of resources to embed would also reduce the running time. Similar to the general properties in P^*, some classes, for example `Thing`, could also automatically be scored 0, instead of getting a cosine similarity score. This would also decrease the scoring time, which gets very large as $|\Sigma|$ increases. Testing other implementation choices for the scoring functions could also decrease the time. Moreover, for future work other embedding methods should also be explored.

Furthermore, the benchmark evaluation showed promising results for RBSG, especially for $k = 5$. The results from the evaluation with ESBM, showed that the more advanced entity summarizers performed better. Currently the scoring in RBSG is very simple, so an interesting direction for future work would be to create a more advanced scoring, where for example diversity between triples are included.

The results from the benchmark evaluation of EBSG were not as promising as for RBSG. However, it has been shown that the best walking strategy for RDF2Vec varies for different tasks, and that tuning the strategy therefore is important. With more testing of walking strategies, the performance could be improved. Also, the current scoring in EBSG only includes relatedness to signature. As with RBSG, including both diversity and relatedness between

triples in the scoring could improve the result.

In addition the benchmark itself could also be improved. It is small, with only 99 snippets, and only includes one dataset. A good direction for future work would be to make it as large as ESBM, with six people creating a snippet given each signature, and also include more datasets, for example LinkedMDB. Then the benchmark would be more robust, and the evaluation results less affected by an individual opinion.

Lastly, the idea of reducing the size of the KG before materializing it was quickly abandoned, because the shrinking algorithm was to slow, and not significantly affected the size of the KG. However, some small changes could have a positive effect on the running time. For example finding a good balance of the sizes of GP and GO.

Appendices

APPENDIX A

RDFS Entailment Rules

Rule name	Premise(s)	Conclusion
rdfs1	x p l . where l a literal	_ : bn rdf:type rdfs:Literal . where _ : bn a blank node
rdfs2	p rdfs:domain c . x p y .	x rdf:type c .
rdfs3	p rdfs:range c . x p y .	y rdf:type c .
rdfs4a	x p y .	x rdf:type rdfs:Resource .
rdfs4b	x p y .	y rdf:type rdfs:Resource .
rdfs5	p_1 rdfs:subPropertyOf p_2 . p_2 rdfs:subPropertyOf p_3 .	p_1 rdfs:subPropertyOf p_3 .
rdfs6	p rdf:type rdf:Property .	p rdfs:subProperty p .
rdfs7	p_1 rdfs:subPropertyOf p_2 . x p_1 y .	x p_2 y .
rdfs8	c rdf:type rdfs:Class .	c rdfs:subClassOf rdfs:Resource .
rdfs9	c_1 rdfs:subClassOf c_2 . x rdf:type c_1 .	x rdf:type c_2 .
rdfs10	c rdf:type rdfs:Class .	c rdfs:subClass c .
rdfs11	c_1 rdfs:subClassOf c_2 . c_2 rdfs:subClassOf c_3 .	c1 rdfs:subClassOf c_3 .
rdfs12	x rdf:type rdfs:ContainerMembershipProperty .	x rdfs:subPropertyOf rdfs:member .
rdfs13	x rdf:type rdfs:Datatype .	x rdfs:subClassOf rdfs:Literal .

Table A.1: The RDFS Entailment rules. Grey color indicates that the rule is used in the materialization algorithm.

APPENDIX B

Properties with Similarity Score 0

- <http://purl.org/dc/terms/subject>
- <http://xmlns.com/foaf/0.1/homepage>
- <http://xmlns.com/foaf/0.1/depiction>
- <http://www.w3.org/2000/01/rdf-schema#label>
- <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
- <http://www.w3.org/2000/01/rdf-schema#seeAlso>
- <http://xmlns.com/foaf/0.1/name>
- <http://www.w3.org/2002/07/owl#differentFrom>
- <http://www.w3.org/2003/01/geo/wgs84_pos#lat>
- <http://www.w3.org/2003/01/geo/wgs84_pos#long>
- <http://www.georss.org/georss/point>
- <http://www.w3.org/2004/02/skos/core#subject>
- <http://xmlns.com/foaf/0.1/nick>
- <http://xmlns.com/foaf/0.1/givenName>
- <http://xmlns.com/foaf/0.1/page>
- <http://purl.org/dc/elements/1.1/description>
- <http://xmlns.com/foaf/0.1/surname>
- <http://purl.org/dc/elements/1.1/type>
- <http://xmlns.com/foaf/0.1/thumbnail>
- <http://xmlns.com/foaf/0.1/logo>
- <http://xmlns.com/foaf/0.1/familyName>
- <http://purl.org/dc/elements/1.1/rights>
- <http://dbpedia.org/ontology/category>
- <http://dbpedia.org/ontology/type>
- <http://dbpedia.org/ontology/otherName>

APPENDIX C

Evaluation of an Earlier Version of RBSG

The first version of the RBSG approach consisted of the shrinking algorithm, the materialization algorithm, and the scoring and ranking of triples. The primary objective of the shrinking algorithm was to reduce the running time of the materialization algorithm by creating a smaller graph for materialization. However, the experiment with the shrinking algorithm below shows that the RBSG approach works better without including the shrinking algorithm.

To evaluate the shrinking algorithm we performed a similar experiment as in Section 6.1.1. For each of the three signature sizes $|\Sigma| = 1$, $|\Sigma| = 5$, and $|\Sigma| = 10$, we created 25 signatures. Let G denote the original graph, G' denote the materialized graph, and G_s denote the output graph from running the shrinking algorithm with G as input. For each of the signatures, we first ran the materialization algorithm on G, then ran the shrinking algorithm on G, and lastly we ran the materialization algorithm again, but on G_s. In each of the $25 \times 3 = 75$ executions, we timed the algorithms, and computed the size of the output graphs. The experiment was performed first for random signatures, then for seed signatures.

In early small trials with the shrinking algorithm, it tended to be very slow. We therefore only ran the algorithms for 25 signatures, instead of 100, and used a smaller graph than in Section 6.1.1. In this experiment we used graph with size $|G| = 14077$. The results from the experiment are shown in Table C.1 and Table C.2.

Algorithm	Materialization			Shrinking			Materialization		
Input graph	G			G			G_s		
$\|\Sigma\|$	1	5	10	1	5	10	1	5	10
min	0.17	0.31	0.42	69.57	71.30	71.36	0.16	0.28	0.39
max	0.42	0.72	0.98	94.52	82.26	81.22	0.35	0.46	0.91
mean	0.28	0.41	0.55	79.45	76.60	75.93	0.22	0.34	0.50

Algorithm	Materialization			Shrinking			Materialization		
Input graph	G			G			G_s		
$\|\Sigma\|$	1	5	10	1	5	10	1	5	10
min	0.20	0.29	0.36	71.05	68.40	70.43	0.17	0.27	0.36
max	0.42	0.55	1.16	81.15	95.17	87.68	0.38	0.68	1.00
mean	0.26	0.37	0.53	76.27	79.17	74.94	0.22	0.39	0.52

Table C.1: Running time (s) for the materialization algorithm and the shrinking algorithm for random signature (top) and seed signature (bottom).

Algorithm	Materialization			Shrinking			Materialization		
Input graph	G			G			G_s		
$\|\Sigma\|$	1	5	10	1	5	10	1	5	10
min	0	+2	+3	-901	-901	-901	0	+2	+3
max	+3	+6	+25	-901	-901	-901	+3	+6	+25
mean	+0.60	+7.96	+15.56	-901	-901	-901	+0.60	+7.96	+15.56

Algorithm	Materialization			Shrinking			Materialization		
Input graph	G			G			G_s		
$\|\Sigma\|$	1	5	10	1	5	10	1	5	10
min	0	+1	+2	-901	-901	-901	0	+1	+2
max	+3	+734	+739	-901	-855	-855	+3	+734	+739
mean	+0.60	+80.56	+137.76	-901	-899.16	-895.48	+0.60	+80.56	+137.76

Table C.2: Size increase (+) and size reduction (-) between the input and the output graph for the materialization algorithm and the shrinking algorithm, for random signature (top) and seed signature (bottom).

Table C.1 presents the running time for the materialization algorithm with the original graph G as input and the shrunken graph G_s as input. It also presents the running time of the shrinking algorithm. The materialization algorithm is faster with G_s as input, than G as input, but only with a few centiseconds. $|\Sigma|$ has a much larger effect on the materialization time than using the shrinking algorithm first has.

Looking at the mean running time for the shrinking algorithm for both random and seed signature, the values ranges between 74.94 and 79.45. Mean running time for the materialization algorithm is less than one second for all cases presented in the table. Therefore, even though materialization with a shrunken graph is faster, the full RBSG approach will be considerably slower if the shrinking algorithm is included. Also, the largest running time for

materializing G was 1.16 seconds, so the materialization algorithm is fairly fast on its own.

Furthermore, Table C.2 presents how many new triples are generated from materializing G and G_s, that is $|G'| - |G|$ and $|G'_s| - |G_s|$, respectively. Additionally it presents the number of triples that are removed using the shrinking algorithm, that is $|G| - |G_s|$. First, note that $|G'| - |G| = |G'_s| - |G_s|$, proving that using the shrinking algorithm first, does not exclude any important triples from materialization. Second, for a random signature, 901 triples are removed from G to make G_s, regardless of the signature size. This is only 6.4% of the triples in G. At the same time, the running time for shrinking G and then materializing it is between 10000% and 30000% longer than just materializing G. Therefore using the shrinking algorithm produces only a small reduction of triples in G, but a huge increase in the running time of RBSG, which is not ideal.

The results does not show any signs that using a seed signature affects the running time of the shrinking algorithm, compared to using a random signature. The maximal size of G_s for $|\Sigma| = 5$ and $|\Sigma| = 10$ is larger when using a seed signature than using a random signature (855 triples removed compared to 901 triples removed), but it is only a 46 triples difference. Therefore the shrinking algorithm is not greatly affected by the signature type. In total the experiments show that the shrinking algorithm is to slow to be helpful, compared to how many triples where removed from the original KG. Therefore it only increased the total time spent on the whole approach, and hence was not used any further. The results also show that the materialization algorithm is fairly fast, and can be used without shrinking the graph first. Table 6.2 showed that the longest running time for the materialization algorithm with $|G| = 46187$ was 9.96 seconds (seed signature, $|\Sigma| = 10$), which is still significantly faster than shrinking a graph that is only 1/3 of the size.

Bibliography

[BBP14] Benedetti, F., Bergamasch, S. and Po, L. 'A Visual Summary for Linked Open Data Sources'. In: *Proceedings of the 2014 International Conference on Posters & Demonstrations Track - Volume 1272*. ISWC-PD'14. Riva del Garda, Italy: CEUR-WS.org, 2014, pp. 173–176.

[Bor+13] Bordes, A. et al. 'Translating Embeddings for Modeling Multi-Relational Data'. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 2787–2795.

[Bos+22] Boschin, A. et al. 'Combining Embeddings and Rules for Fact Prediction'. In: *Summer School on Artificial Intelligence in Bergen (AIB)*. 2022.

[Che+17] Cheng, G. et al. 'Generating Illustrative Snippets for Open Data on the Web'. In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM '17. Cambridge, United Kingdom: Association for Computing Machinery, 2017, pp. 151–159.

[Che+19] Chen, J. et al. 'Towards More Usable Dataset Search: From Query Characterization to Snippet Generation'. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. Beijing, China: Association for Computing Machinery, 2019, pp. 2445–2448.

[CTQ11] Cheng, G., Tran, T. and Qu, Y. 'RELIN: Relatedness and Informativeness-Based Centrality for Entity Summarization'. In: *The Semantic Web – ISWC 2011*. Ed. by Aroyo, L. et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 114–129.

[CXQ15a] Cheng, G., Xu, D. and Qu, Y. 'C3d+ p: A summarization method for interactive entity resolution'. In: *Journal of Web Semantics* vol. 35 (2015), pp. 203–213.

[CXQ15b] Cheng, G., Xu, D. and Qu, Y. 'Summarizing entity descriptions for effective and efficient human-centered entity linking'. In: *Proceedings of the 24th International Conference on World Wide Web*. 2015, pp. 184–194.

[Dol+07] Dolby, J. et al. 'Scalable Semantic Retrieval through Summarization and Refinement'. In: *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 1*. AAAI'07. Vancouver, British Columbia, Canada: AAAI Press, 2007, pp. 299–304.

[GM10] Goodman, E. L. and Mizell, D. 'Scalable in-memory RDFS closure on billions of triples'. In: *The 6th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2010)*. 2010, p. 17.

[GTS15] Gunaratna, K., Thirunarayan, K. and Sheth, A. 'FACES: Diversity-Aware Entity Summarization Using Incremental Hierarchical Conceptual Clustering'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* vol. 29, no. 1 (Feb. 2015).

[Gun+17] Gunaratna, K. et al. 'Relatedness-based multi-entity summarization'. In: *IJCAI: proceedings of the conference*. Vol. 2017. NIH Public Access. 2017, p. 1060.

[Hog+21] Hogan, A. et al. 'Knowledge Graphs'. In: *ACM Comput. Surv.* vol. 54, no. 4 (July 2021).

[KNB18] Kroll, H., Nagel, D. and Balke, W.-T. 'BAFREC: Balancing Frequency and Rarity for Entity Characterization in Open Linked Data'. In: *Proceedings of 1st International Workshop on Entity Retrieval, co-located with CIKM 2018 (EYRE)*. 2018.

[KP18] Kazemi, S. M. and Poole, D. 'SimplE Embedding for Link Prediction in Knowledge Graphs'. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 4289–4300.

[KW17] Kipf, T. N. and Welling, M. 'Semi-Supervised Classification with Graph Convolutional Networks'. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[Liu+20] Liu, Q. et al. 'ESBM: An Entity Summarization BenchMark'. In: *The Semantic Web*. Ed. by Harth, A. et al. Cham: Springer International Publishing, 2020, pp. 548–564.

[Liu+21] Liu, Q. et al. 'Entity Summarization: State of the Art and Future Challenges'. In: *Journal of Web Semantics* vol. 69 (2021), p. 100647.

[Mik+13] Mikolov, T. et al. 'Efficient Estimation of Word Representations in Vector Space'. In: *International Conference on Learning Representations*. 2013.

[NTK11] Nickel, M., Tresp, V. and Kriegel, H.-P. 'A Three-Way Model for Collective Learning on Multi-Relational Data'. In: *International Conference on Machine Learning*. 2011.

[Pau23] Paulheim, H. *RDF2vec.org*. Last accessed 25 October 2023. 2023.

[Rie+14] Rietveld, L. et al. 'Structural Properties as Proxy for Semantic Relevance in RDF Graph Sampling'. In: *The Semantic Web – ISWC 2014: 13th International Semantic Web Conference, Riva Del Garda, Italy, October 19-23, 2014. Proceedings, Part II*. Riva del Garda, Italy: Springer-Verlag, 2014, pp. 81–96.

[Ris+19] Ristoski, P. et al. 'RDF2Vec: RDF graph embeddings and their applications'. In: *Semantic Web* vol. 10 (2019), pp. 721–752.

[RP16] Ristoski, P. and Paulheim, H. 'RDF2Vec: RDF Graph Embeddings for Data Mining'. In: *The Semantic Web – ISWC 2016*. Ed. by Groth, P. et al. Cham: Springer International Publishing, 2016, pp. 498–514.

[Sch+18] Schlichtkrull, M. et al. 'Modeling Relational Data with Graph Convolutional Networks'. In: *The Semantic Web*. Cham: Springer International Publishing, 2018, pp. 593–607.

[Soc+13] Socher, R. et al. 'Reasoning with Neural Tensor Networks for Knowledge Base Completion'. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 926–934.

[Ste+23] Steenwinckel, B. et al. 'pyRDF2Vec: A Python Implementation and Extension of RDF2Vec'. In: *European Semantic Web Conference*. Springer Nature Switzerland, 2023, pp. 471–483.

[Sub+16] Subercaze, J. et al. 'Inferray: Fast in-Memory RDF Inference'. In: *Proceedings of the VLDB Endowment* vol. 9, no. 6 (Jan. 2016), pp. 468–479.

[Suc+19] Suchanek, F. M. et al. 'Knowledge Representation and Rule Mining in Entity-Centric Knowledge Bases'. In: *Reasoning Web*. 2019.

[Sun+19] Sun, Z. et al. 'RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space'. In: *International Conference on Learning Representations*. 2019.

[Tro+16] Trouillon, T. et al. 'Complex Embeddings for Simple Link Prediction'. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 2071–2080.

[Tsa+21] Tsamoura, E. et al. 'Materializing Knowledge Bases via Trigger Graphs'. In: *Proceedings of the VLDB Endowment* (2021), pp. 943–956.

[Urb+10] Urbani, J. et al. 'OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples'. In: *The Semantic Web: Research and Applications*. Ed. by Aroyo, L. et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 213–227.

[Wan+14] Wang, Z. et al. 'Knowledge Graph Embedding by Translating on Hyperplanes'. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI'14. Québec City, Québec, Canada: AAAI Press, 2014, pp. 1112–1119.

[WCK19] Wang, X., Cheng, G. and Kharlamov, E. 'Towards Multi-Facet Snippets for Dataset Search'. In: *Joint Proceedings of PROFILES 2019 and SEMEX 2019, The 6th International Workshop on Dataset PROFlLing and Search (PROFILES 2019), co-located with the 18th International Semantic Web Conference (ISWC '19)*. PROFILES-SEMEX 2019. CEUR-WS. Auckland, New Zealand, 2019, pp. 1–6.

[Yan+15] Yang, B. et al. 'Embedding Entities and Relations for Learning and Inference in Knowledge Bases'. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Bengio, Y. and LeCun, Y. 2015.

[ZZC12] Zhang, L., Zhang, Y. and Chen, Y. 'Summarizing Highly Structured Documents for Effective Search Interaction'. In: *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '12. Portland, Oregon, USA: Association for Computing Machinery, 2012, pp. 145–154.