Collaborative Project

FROCKG - Fact Checking for Large Enterprise Knowledge Graphs

Project Number: E! 113314 Start Date of Project: 2020/01/01 Duration: 36 months

Deliverable 2.2: Final version of the fact checking algorithms including benchmarking results

Dissemination Level	Public
Due Date of Deliverable	Month 32, 2022/08/31
Actual Submission Date	Month 32, 2022/08/31
Work Package	WP2
Tasks	T2.1, T2.2
Type	Report
Approval Status	Final
Version	1.0

Abstract:
This report presents the final prototype of the Fact Checking components of the FROCKG platform. This includes approaches for checking a single fact as well as an approach to check a complete knowledge graph. The report covers a description of the approaches as well as an evaluation.

Project by Eurostars.

History

Version	Date	Reason	Revised by
0.1	2022/04/04	Drafted content	Michael Röder, Farshad Afshari
0.2	2022/08/19	First draft finished	Michael Röder, Farshad Afshari
1.0	2022/08/31	Final version	Michael Röder, Farshad Afshari

Author List

Organization	Name	Contact Information
Paderborn University	Michael Röder	michael.roeder@upb.de
Paderborn University	Farshad Afshari	farshad.afshari@uni-paderborn.de

Contents

Introduction

The main goals of this work package are twofold. First, we develop an approach to compute the veracity of single facts based on both textual evidence and corroborating facts found in a reference knowledge graph. To achieve this goal, the consortium develops novel solutions based on supervised explainable machine learning algorithms (T2.1).
The second goal is to check not only single facts but a set of facts (ranging from small sub-graphs to large knowledge bases) for their veracity. A manual check of every single fact in a knowledge base is infeasible for large knowledge bases. Hence, an automatic approach is needed that reduces the number of triples that may have to be checked by a human expert [21] (T2.2).
This deliverable describes the final prototypes that were implemented to achieve the aforementioned goals. The University Paderborn (UPB) leads the development. Note that these prototypes are extensions of the intermediate prototypes described in D2.1 [20] and implement the requirements defined in D1.2 [14] and D1.3 [15].
The next section describes the developments that have been done within T2.1. This includes a description of the different Fact Checking services and their evaluation. The section after that describes the knowledge graph veracity framework that has been developed within T2.2, and its evaluation. The last section concludes the deliverable.

T2.1 Fact Checking single Facts

We define the task of checking a single fact as follows: “Given a fact, compute the likelihood that the given fact is true" [11]. In the context of the FROCKG project, the facts originate from a knowledge base, i.e., they can be represented as a triple (s, p, o) with a subject s, a predicate p and an object o. The triple represents a relation between s and o. The type of relation is expressed by p. For example, the triple (“Barack Obama”, “nationality”, “United States of America”) represents the fact that Barack Obama is a citizen of the United States of America. We will use this example throughout this section.
Fact Checking approaches for single facts of the form (s, p, o) can be separated into two groups—text-based approaches and knowledge-graph-based approaches. The FROCKG project makes use of both approaches. Text-based approaches rely on a reference corpus and use it to identify evidence for the given fact. Knowledge-graph-based approaches use knowledge graphs to derive knowledge that either supports or refutes the given fact. The techniques used for that can rely on the identification of paths between the given subject and object that support or refute the triple.[1] Other approaches mine graph patterns which are used as evidence. However, it has been shown that path-based approaches outperform the pattern-based approaches [11]. Hence, we prefer the path-based approaches when we work with knowledge-graph-based approaches.
Within the FROCKG project, both groups of approaches are utilized. The interfaces of the FROCKG platform described in D1.2 [14] have been defined in a way that one or several

[1] Note that refuting a triple is not easy since the Open World Assumption has to be taken into account. Hence, it is always possible that a triple that seems to be unlikely is still possible to exist and that our reference knowledge is simply incomplete.

fact checking algorithms can be used. These algorithms are implemented as individual services and are hidden behind a facade.
Within T2.1, two services—a text-based and a knowledge-graph-based approach, respectively—are developed. Both are integrated into the facade service which will be the main service used by other components of the FROCKG platform.

Text-based Fact Checking

A text-based Fact Checking approach relies on a reference corpus to search for pieces of evidence that support the given fact. For the FROCKG platform, we decided to use the state-of-the-art approach FactCheck [1].
FactCheck uses a local corpus by utilizing Elasticsearch[2] to index the documents of the corpus. When it is used to check a given fact, the fact is transformed into keyword queries. For the running example, the following queries could be generated:

"Barack Obama nationality United States of America"
"Barack Hussein Obama II nationality U.S.A."
"Barack Obama born in United States"
...

For generating queries, information about the three parts of a fact are combined. This includes the different labels of the subject and object as well as different formulations for the predicate. In the example above, FactCheck already knows that the nationality property is closely bound to the place of birth. Hence, the generated queries include this information.
These queries are used to search for relevant documents in the reference corpus. The retrieved documents are further analyzed in two ways. First, parts of the text in which the searched keywords occur are extracted. In our running example, these pieces of text could look as follows:

"During Obama's terms in office, the United States' reputation abroad, as well as the American economy, significantly improved."

"Obama was born in Honolulu (U.S.A.)."

Each extracted piece of text is deeply analyzed and rated by a machine learning algorithm with respect to its quality. This includes the distance between the subject and the object of the given fact within the extracted text. In contrast to previous approaches (e.g., Gerber et al. [12]), FactCheck additionally makes use of the sentence structure instead of relying on simple string matching algorithms. In the example above, the analysis would reveal that Obama and the United States are not well connected within the first text although their distance is small. In the second piece of text, the two terms are connected via a verb that represents the predicate FactCheck searched for. Hence, the second piece of text would receive a better score by the machine learning model.
A second analysis that is run by FactCheck is the measurement of trustworthiness of the

[2] https://www.elastic.co/

identified documents. To this end, FactCheck relies on the trust model presented by Nakamura et al. [13]. It uses articles that describe the subject and object (Nakamura et al. [13] suggest Wikipedia articles) and compares the content of the articles with the content of the retrieved document. This ensures that documents with a high topical overlap with the subject or object article get a higher trust score than documents with a different topic.
In its final step, FactCheck relies on a trained classifier that uses all identified pieces of evidence to decide whether the given fact is true or false. The result is a truth value that is returned as result.
We adapted the open-source implementation to the needs of the FROCKG platform. This included several changes:[3]

- FactCheck has been mainly used for checking facts of the DBpedia. We removed this dependency and generalized its workflow to work with facts of any knowledge base.
- When used for a new set of properties on a new knowledge base, FactCheck has to be trained. We made the configuration and execution of this training easier.
- FactCheck solely returned the truth value. However, since the explanation of the Fact Checking process is of central importance for the FROCKG platform, we extended the service API of FactCheck according to D1.2 [14] to return the identified pieces of evidence and their detailed scores. This additional information is crucial for the implementation of the generation of explanations in WP3 of the FROCKG project.
- We created a Docker image to integrate the service into the FROCKG platform.
- We set up an environment that eases the evaluation of FactCheck's performance based on a given benchmark dataset. This allows an easier adaptation of FactCheck's configuration based on a given dataset.
- We extended a module within FactCheck that allows an easier configuration of additional property verbalizations. FactCheck mainly relied on pregenerated patterns to generate more diverse queries (e.g., "nationality" led to the usage of "born in" in the example above). Since these patterns won't be available for all RDF properties that will be used within the FROCKG project, we assured that manually created patterns can be added easily.
- Within a master thesis, which was supported by the FROCKG project, we explored the possibility to extend FactCheck to search for refuting evidence. A first attempt was made with an approach that executes a search not only with wildcard queries. These queries do not search for the given triple but for combinations of the given subject and predicate. After that, the algorithm extracts the object. This gives additional information about the subject and evidence, e.g., in the example above, it could identify other countries if Barack Obama had their nationality.

Knowledge-Graph-based Fact Checking

A knowledge-graph-based Fact Checking approach makes use of structured information of a reference knowledge graph to identify pieces of evidence that support or refute the given fact. To enhance the performance of the Fact Checking service implemented in the FROCKG platform, we decided to offer the usage of such a Fact Checking algorithm in addition to the text-based Fact Checking. To this end, we utilized the open-source approach

[3] The changes are open-source and can be found at https://github.com/dice-group/FactCheck

COPAAL [11] and extended it to implement a reliable service that can be used in the platform. In addition, we extended its path search approach by implementing ESTHER—a search for corroborative paths that relies on knowledge graph embedding spaces. Both are described in more detail in the following.

COPAAL

COPAAL [11] is an open-source knowledge-graph-based Fact Checking algorithm. For a given fact, COPAAL searches for paths between the subject and object of the fact within the reference knowledge graph. Figure 1 shows some paths between the resources representing Barack Obama and the United States of America for our example. It can be seen that the paths can have different lengths (2 or 3 triples in the figure).

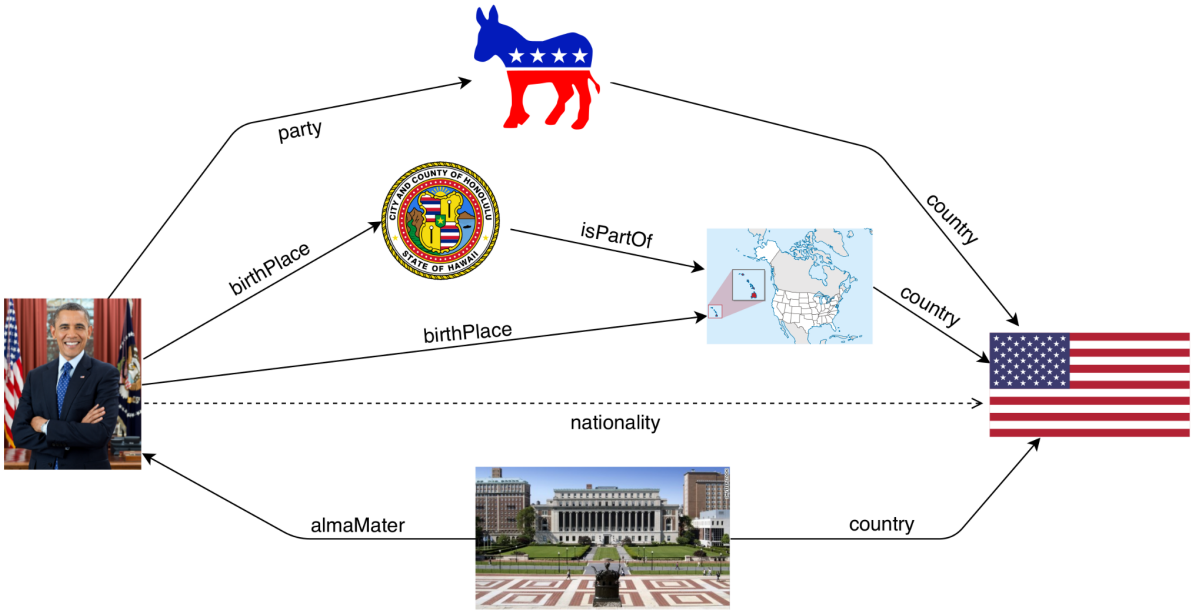

Figure 1: The fact that should be checked (the dotted line) and additional paths between the resources Barack Obama and the United States of America extracted from the DBpedia.

These paths are scored whether they corroborate the existence of the fact using an NPMI-based heuristic. To make COPAAL applicable for the FROCKG platform, we extended the existing implementation in the following ways:[4]

- We identified several problems in the existing prototypical implementation of COPAAL. We changed the structure of the project and separated it into different modules. These modules make COPAAL extendible and adaptable to different use cases. We will give more details further below.
- Several flaws have been fixed, e.g., the new implementation is able to rely on paths with an arbitrary size while the original implementation was limited to paths of the size 2. Similarly, the summarization of the scores for the paths to form a final truth value for the given fact has been improved.
- We added a database that can be used to store intermediate results for later reuse.
- We created a Docker image to integrate the service into the FROCKG platform.

[4] The changes are open-source and can be found at https://github.com/dice-group/COPAAL

ESTHER

We extended the concept of the search for corroborative paths, i.e., the search implemented in COPAAL, to knowledge graph embedding spaces. Instead of the costly search within the knowledge graph based on individual given facts, ESTHER searches for corroborative paths within a knowledge graph embedding space. For a given property, ESTHER applies an A* search algorithm to generate candidates for corroborative paths. Identified paths are scored similar to the paths identified by COPAAL. Paths with a high score are stored for later reusage.
An advantage of ESTHER is that the search for corroborative paths can be applied as a preprocessing step. For checking a given fact, the algorithm is reduced to check the existence of the previously identified corroborative paths between the fact's subject and object. This reduces the runtime to check single facts.
ESTHER is integrated into the FROCKG platform by extending the COPAAL service to make use of both path search approaches—the original COPAAL search algorithm and the ESTHER algorithm—to identify corroborative paths.[5] This approach has been published at the ISWC 2021 [22].

Modularization of COPAAL

Based on its internal workflow, we separated COPAAL into three modules that implement the following three steps:

1. Path search,
2. Path scoring, and
3. Claim validation.

Note that the first and second step do not rely on the fact that COPAAL should check. Instead, they only rely on the property that such a fact could have as predicate. Hence, they can either be executed at runtime, i.e., with the fact that should be checked as input, or they can be executed as a preprocessing step for each property that COPAAL should be able to process. In the latter case, only the third step has to be executed at runtime.

Path Search

The path search is the first module in COPAAL's workflow. It receives a property as input and searches for paths that are candidates for corroborating the existence of a triple with the given property. There are three different variants of this search implemented within FROCKG.
The first variant relies on a number of example triples that have the given property as predicate. For each of these triples, all paths between the triple's subject and object are retrieved. This variant can be run with a single triple, a number of sampled triples from the knowledge graph or all triples that are known to have the given property.
The second variant searches in a meta space utilizing the ESTHER approach described above. The third variant solely relies on the ontology of the reference knowledge graph. It generates all possible paths that could exist between a subject and an object of the given property.

[5] The source code is open-source and available at https://github.com/dice-group/esther

All three variants have strengths and weaknesses. The first variant will always lead to reliable results, if the number of sampled triples is high enough. However, it has the disadvantage that it can lead to a very long runtime. In cases, in which the knowledge graph is large but comes with rich domain and range information for its properties, the third variant might be faster since it does not have to actively search within the graph. The second variant has the advantage that it can be used to identify longer paths but it relies on knowledge graph embeddings which have to be available.
Please note that COPAAL is not bound to the three searches described above. The new modular architecture allows various implementations. For example, a module could make use of existing efficient graph path search implementations, e.g., the graph path search implemented within GraphDB by the FROCKG project partner Ontotext.

Path Scoring

The path scoring is the second module of COPAAL. It receives a property and a list of paths that are candidates for corroborating the existence of a triple with said property. To this end, it relies on a score that measures the co-occurrence of a path and triples with the given property. The score is based on counts that are derived from the reference knowledge graph. The module can be configured to use different endpoint implementations and scoring strategies:

- While the prototypical implementation of COPAAL is based on SPARQL, we faced several problems when using other SPARQL stores than OpenLink Virtuoso.[6] We tested several other endpoints and extended the SPARQL client in COPAAL to make use of different request types and response formats to support these stores.[7] Especially the integration of the Tentris triple store [23] is an important change. This in-memory store showed a good performance when deriving occurrence counts within the graph but supported only a subset of the SPARQL query standard. Hence, we adapted the queries within COPAAL to be compatible with Tentris.
- COPAAL makes use of domain and range information of the given fact's property. However, such information is not available for all RDF properties. To this end, we introduced the concept of "virtual types". A property without a domain is assigned a "virtual" subject type and all subjects of triples with this property get this type assigned. The same can be done for the objects of these triples, if the range information is missing.
- We added the possibility to rely on exact counts instead of the heuristics suggested by Syed et al. [11]. This can be very helpful in scenarios in which the heuristic leads to bad results and the size of the graph allows the determination of the exact number of connected pairs.

Claim Validation

The third step in the workflow receives a triple which should be checked and relies on the scored corroborative paths that have been found in the previous step for the given triple's

[6] https://virtuoso.openlinksw.com/
[7] We also used GraphDB (https://www.ontotext.com/products/graphdb/) in some experiments. While we did not encounter problems with GraphDB, we found that SPARQL queries optimized for OpenLink Virtuoso did not work well on GraphDB. This showed that using different strategies to generate SPARQL queries depending on the SPARQL store used can further optimize the runtime of COPAAL.

predicate. This step checks whether the subject and object of the given triple are connected by any of the given paths. If an instance of these paths exists between them, the score is added to a list of scores. Finally, the list is summarized to form a final, single veracity value for the given triple.
Our experiments revealed that the summarization of the scores described in the publication of Syed et al. [11] is not able to handle negative scores in a correct way. To this end, we evaluated several different summarizations and kept the best performing for our experiments.

Fact Checking Facade

Following D1.2 [14], the FROCKG platform provides a Fact Checking facade service. This facade ensures that the usage of the Fact Checking components is eased by providing a single service that encapsulates all functionalities. This facade service includes the following functionalities:[8]

- Given a fact to check, it calls one or several Fact Checking services. Both types of Fact Checking services are integrated, i.e., text-based as well as knowledge-graph-based Fact Checking services.
- If multiple services are available, their results are merged. In the current version, the maximum score is used, i.e., if one of the services returns a high truth value for the given fact this value will be returned.
- The facade handles the pieces of evidence returned by the services and forwards them to the integrated explanation component which is developed within WP3. The result (i.e., the explanation of the Fact Checking) is added to the Fact Checking result that is sent to the user.
- Since the facade can be used with any Fact Checking service, it offers two ways to request the check of a triple. The first variant keeps the HTTP connection of the request open until the result is available. Then, the result is sent as response to the request. However, in cases in which the Fact Checking services may need more time than some seconds, this becomes impractical. Hence, we implemented a second variant, in which the facade answers with an ID to the request. The client can use this ID to regularly check the status of the Fact Checking process.
- In addition to an API to check the fact, it contains an API endpoint that returns the human-readable result of the fact and the explanation.

In the most recent version of the facade service, the merging of results prefers the highest veracity score over all other scores. However, the facade itself is independent of the merging strategy and other approaches can be implemented as well. For example, the FROCKG project supported the development of a hybrid Fact Checking approach that was developed within the KnowGraphs project.[9] This hybrid approach combines the usage of textual reference data, knowledge graph embeddings and a path-based approach using an artificial neural network [25]. At the time of writing, the scientific article describing this approach has been accepted at the International Semantic Web Conference 2022 [25].

[8] The source code is open-source and available at https://github.com/dice-group/FROCKG
[9] https://knowgraphs.eu/

Evaluation

For our evaluation we focus on the evaluation of FactCheck, COPAAL and the facade service. In a second experiment, we evaluate ESTHER. The third experiment compares the runtimes of COPAAL with and without pre-processed data.

Datasets

For the first experiment, we use the FactBench dataset created by Gerber et al. [12]. We use the property subset, since it is known to be the hardest dataset within FactBench. The dataset comprises 750 true and 750 false facts. The true facts are gathered from DBpedia while the false facts are generated. The generation is done by randomly choosing a triple (s,p,o) and replacing the property with p' to form the false fact (s,p',o). During that, it is ensured that the domain and range restrictions of p' are fulfilled [12].
For the second experiment, we use the datasets FB15k-237 and WN18RR. Both datasets come with a small knowledge graph and a set of true facts. The main features of the datasets are listed in Table 1. We randomly generated wrong facts as suggested by Gerber et al. [12]. We configure ESTHER to work with three different embedding algorithms: TransE [17], RotatE [18] and DensE [19]. In the experiment, we compare ESTHER to 10 other graph-based approaches. In the first run, we benchmark all 11 approaches on their own. In the second run, we combine each approach with ESTHER using a decision tree. Since this transforms the approach into a supervised approach, we use a 10 fold cross validation.
Our third experiment relies on the DBpedia dump from March 2021. Its features are shown in Table 1. We use the following five example facts to measure the time needed to check a single fact:

- dbr:Jack_Welch, dbo:birthPlace, dbr:Peabody,_Massachusetts
- dbr:Laura_Ingalls_Wilder, dbo:deathPlace, dbr:Mansfield,_Missouri
- dbr:John_of_Saxony, dbo:spouse, dbr:Amalie_Auguste_of_Bavaria
- dbr:Port_Sinister, dbo:starring, dbr:Lynne_Roberts
- dbr:Thomas_Howard,_4th_Duke_of_Norfolk, dbo:child, dbr:Thomas_Howard,_1st_Earl_of_Suffolk

We measure the runtime of COPAAL with and without pre-processing. When we use pre-processing, we rely on the generation of all possible property combinations that could create paths between the given subject and object. COPAAL without the pre-processing is used in the setup described by Syed et al. [11]. As SPARQL endpoint, we rely on Tentris to gather counts.[10]

	FB15k-237	WN18RR	DBpedia 03/2021
Entities	14,541	40,943	9,383,908
Relations	237	11	14,016

[10] The experiment is executed on a VM with an Intel Xeon E5-2695 v4 and 125 GB RAM.

Triples	289,650	89,869	375,900,462
True facts	750	750	—

Table 1: Features of the datasets used in the second and third experiment.

Results

We measure the performance of a Fact Checking approach using the area under the receiver operating characteristic curve (AUC-ROC) [1]. We report the results of experiment 1 in Table 2.[11] It can be seen that different configurations of COPAAL lead to a different performance. This shows that for the use cases within WP6 of the FROCKG project different configurations of COPAAL should be tested as a single configuration may not always perform best. The best performing configuration for FactBench's property subset defines the maximum path length to be 3 and uses virtual types.

FactCheck achieves a lower AUC-ROC value than the best COPAAL configuration. However, the results of the Facade are all better than their corresponding COPAAL instances. Hence, we can conclude that FactCheck and COPAAL have different views on the given facts and that their combination leads to better results.

Approach	AUC-ROC
COPAAL (length 2)	0.6331
COPAAL (length 3)	0.4806
COPAAL (length 2, vt)	0.6112
COPAAL (length 3, vt)	0.6441
FactCheck	0.5872
Facade (length 2)	0.6896
Facade (length 3)	0.5702

[11] The experiments have been run on GERBIL and are available at http://w3id.org/gerbil/kbc/experiment?id=202106040005, http://w3id.org/gerbil/kbc/experiment?id=202106040001, http://w3id.org/gerbil/kbc/experiment?id=202106040002, http://w3id.org/gerbil/kbc/experiment?id=202106040003, http://w3id.org/gerbil/kbc/experiment?id=202106100001, http://w3id.org/gerbil/kbc/experiment?id=202106040009, http://w3id.org/gerbil/kbc/experiment?id=202106040007, http://w3id.org/gerbil/kbc/experiment?id=202106040010, http://w3id.org/gerbil/kbc/experiment?id=202106100003 .

Facade (length 2, vt)	0.6706
Facade (length 3, vt)	0.6988

Table 2: AUC-ROC results for COPAAL, FactCheck and the Facade on FactBench's property subset. Higher values are better.

We tested different configurations of ESTHER. The approach achieved an AUC-ROC of 83.07 for FB15k-237 when using RotatE embeddings and 77.55 on WN18RR with TransE embeddings. Table 3 shows that ESTHER achieves a better performance than most other approaches. However, some approaches (especially KS) perform better. However, ESTHER was able to improve the performance of all other approaches by 20% on average as shown in Table 3.

Approach	Approach only		With ESTHER	
	FB15k-237	WN18RR	FB15k-237	WN18RR
COPAAL [1]	77.42	68.11	87.12 (+9.70)	79.38 (+11.27)
KS [2]	87.59	86.44	89.97 (+2.38)	94.92 (+8.48)
Katz [3]	82.8	69.96	86.30 (+3.50)	86.22 (+16.26)
Pathent [4]	73.46	79.98	84.75 (+11.29)	86.94 (+6.96)
Simrank [5]	40.07	44.15	81.60 (+41.53)	82.47 (+38.32)
Adamic Adar [6]	72.12	59.86	85.36 (+13.24)	84.22 (+24.36)
Jaccard [7]	38.56	42.34	82.91 (+44.35)	87.18 (+44.84)
Degree Product [8]	77.11	65.57	83.28 (+6.17)	87.43 (+21.86)
PredPath [9]	69.87	80.2	83.76 (+13.89)	82.20 (+2.00)
PRA [10]	8.53	71.8	97.44 (+88.91)	75.35 (+3.55)

Table 3: AUC-ROC values achieved by different approaches with and without ESTHER. The values in parentheses show the difference. Higher values are better.

We evaluate the influence of the pre-processing on the runtime of COPAAL. Table 4 shows the time that the path scoring module takes for each of the five properties. While the generation of candidates for corroborative paths is very fast and takes only some seconds, collecting the counts from the SPARQL endpoint takes several hours. Especially the time

needed for birth and death place properties is high because they connect the two classes dbo:Person and dbo:Place. The instances of these classes are the most connected entities within the DBpedia which leads to a large number of edges that have to be taken into account when counting paths.

Property	Time for co-occurrences	Time for path instances	Total time
dbo:birthPlace	50 h	141 h	191 h
dbo:deathPlace	70 h	240 h	310 h
dbo:spouse	22 h	15 h	37 h
dbo:starring	2 h	3 h	5 h

Table 4: Time needed to count co-occurrences and path instances within the pre-processing.

Property	With Pre-processing		Without Pre-processing	
	Average	Standard Deviation	Average	Standard Deviation
dbo:birthPlace	4.85	0.62	2323.60	175.70
dbo:deathPlace	2.50	0.35	26.60	12.60
dbo:spouse	0.52	0.06	105.00	60.57
dbo:starring	0.66	0.23	127.20	11.76

Table 5: Comparison of the runtime of the fact validation with and without preprocessing (in seconds).

Table 5 compares the time for the fact validation. This is the time a user would have to wait for the system to answer a request. The results show that the usage of pre-processing can speed up the claim validation up by a factor of 480.[12]
We showed that the preprocessing can reduce the runtime of the validation of a single fact. However, collecting counts in a preprocessing might not be a good approach for dynamic knowledge graphs, i.e., graphs that are updated very often. Hence, the FROCKG implementation supports both COPAAL variants, i.e., with and without preprocessing.

T2.2 Knowledge Graph Veracity

Given the size of knowledge graphs, it is impracticable to check every single fact. Hence, developing an algorithm that approximates the overall veracity of a knowledge graph by

[12] Note that the measured times heavily depend on the SPARQL store used within the experiment and the optimization of the queries with respect to that store. We make use of Tentris since it gives us the best results. The Open Link Virtuoso instance of the DBpedia live endpoint can lead to faster runtimes but aborts the query processing after a maximum runtime, which leads to faulty results.

checking only a small portion of the graph is needed. The goal of the T2.2 is the development of an algorithm that can ease the determination of the veracity of a given knowledge graph. Previous approaches try to reduce the number of facts that have to be checked to estimate the overall veracity [24]. However, these approaches still rely mainly on the manual evaluation of triples instead of making use of automatic approaches.

Approach

Our approach is based on the idea to select a subset of facts from the knowledge graph and evaluate them instead of the complete knowledge graph. We defined an adaptable framework that comprises the following steps:

1. Fact selection
2. Unsupervised Fact Checking
3. Supervised Fact Checking
4. Summary generation

The first step selects the subset of facts. The second applies unsupervised Fact Checking algorithms to determine triples that are likely to be true. Based on these triples and automatically generated false triples, we apply supervised Fact Checking approaches in the third step. The fourth step summarizes the results. In the following, the single steps will be explained in more detail.[13]

Fact Selection

The goal of this first step is to select a subset of triples. This is necessary since not all triples of a graph can be checked within a reasonable amount of time. The set of triples should be representative for the graph. There are several methods to select such a subset. We follow the suggestion of Ojha et al. [24] and exclude triples that can be derived from other triples. For example, the majority of the rdf:type triples can be derived from other triples of the same resource. An RDF resource which is subject in a triple with the dbo:birthPlace property, has to have the type dbo:Animal by virtue of RDFS semantics.[14] In a similar way, the resource in the object position of such a triple has to have the type dbo:Place. Hence, rdf:type triples for these resources that express exactly this relation do not have to be checked if the dbo:birthPlace triple has been proven to be correct.

However, we are not only bound to RDFS semantics to derive triples. As pointed out by Ojha et al. [24], additional dependencies can be derived from the graph using rule-mining algorithms.

When finally sampling the triples, Ojha et al. [24] follow an approach that is focused on humans that perform a manual evaluation of triples. Hence, they propose a cost function that relies on the number of triples and the number of distinct subjects within the sampled set. In contrast, we focus on the selection of triples for a small number of properties. As shown in the evaluation of COPAAL with respect to its runtime, the costs of COPAAL mainly occur in a preprocessing phase. The validation of a single triple is fast if the property of this triple is already known by the algorithm. A similar observation can be done for FactCheck. In this

[13] The implementation of the framework is open-source and can be found at https://github.com/dice-group/KGV

[14] See https://dbpedia.org/ontology/birthPlace for details about this property.

supervised algorithm, the costs for a new property lead to the need of a new trained model for this property. Hence, our approach focuses on central properties of the knowledge graph and prefers the check of triples of these properties.
Note that the sampling process can be adapted, e.g., an extension of the cost function is possible. For example, in cases in which a manual check should be done for facts that couldn't be clearly identified as true or false by one of the Fact Checking approaches, it would be possible to include the cost function suggested by Ojha et al. [24].

Unsupervised Fact Checking

The second step of our approach is the application of unsupervised Fact Checking approaches. In our current implementation, we rely on COPAAL. However, note that any other unsupervised approach or even a combination of multiple approaches can be used for this step.
Since most unsupervised approaches rely on a reference knowledge graph, there are two types of reference graphs that can be used. Either the given graph that should be checked or another, linked knowledge graph is used. The first variant is easier to apply in practice since no additional data is needed. However, this means that the Fact Checking approach mainly checks whether the graph in itself is consistent. If the graph has been generated automatically, this might be the case even if the automatic creation is erroneous and led to a faulty graph. For the second variant, another, trustworthy graph that is linked to the given graph is needed. In this case, the properties of the graph have to be mapped to each other, or the properties that have to be checked are introduced in the reference graph while gathering statistics.
The result of this step are veracity scores for the subset of triples selected in the previous step. These results are further enhanced in the next step.

Supervised Fact Checking

In the third step, a supervised Fact Checking approach is trained based on the previous results of the unsupervised approach. We use Fact Check for this step. However, any other supervised approach is applicable as well.
The results of the evaluation within Task 2.1 showed us that COPAAL tends to underestimate the correctness of triples, i.e., the number of triples that are falsely marked as false triples can be high while the number of triples that are falsely marked as correct is typically low. Hence, the triples that have got a high veracity score in the previous step are very likely to be true. We use these triples as known positive examples for the training of the unsupervised approach. We generate the same number of negative examples based on the suggestion of Gerber et al. [12].
Depending on the chosen supervised approach, additional data might be necessary. For the application of FactCheck, a reference corpus is needed. This might differ when other approaches are used.
The trained supervised approach is used to check the veracity of the chosen triples that have not got a clear result within the previous step.

Summary Generation

The last step comprises the generation of a summary that describes the overall veracity of the knowledge base. To this end, we assign a veracity score to all chosen triples in the same way as the facade service is implemented within Task 2.1. Based on these scores, a final score is generated based on the calculated veracity scores. In our current implementation, we rely on the accuracy of the knowledge graph, i.e., the percentage of true facts. To this end, we define a threshold θ and count the number of facts that have a veracity value that is larger than θ.
Note that similar to Ojha et al. [24], it would be possible to extend the framework with an additional manual check. Triples that have a score close to the threshold could be given to human volunteers for a final check. However, this goes beyond the scope of our work.

Evaluation

We evaluate our approach with a similar experiment setup as Ojha et al. [24]. We use the same sample of YAGO 2.0 as they did and rely on their identification of dependencies between single triples.

Datasets

We reuse the YAGO data provided by Ojha et al. [24]. This dataset is a randomly sampled subset of YAGO2. The authors derived dependencies between the triples in the dataset using AMIE [26]. For a subset of 1386 triples that are not dependent on any other triple, the authors created a ground truth, which shows that 99.2% of these triples are correct. The selected subset contains 17 properties listed in Table 6.
We use DBpedia as a reference dataset and Wikipedia as a reference corpus. For the fact conversion, we created a mapping from Yago properties to DBpedia properties. The entities are already linked with owl:sameAs links.

Results

We created a mapping between YAGO and DBpedia properties. If several properties in the DBpedia could be used as equivalent for the YAGO property, we choose a single one.[15] In some cases, no equivalent property existed, which means that COPAAL cannot be applied in these cases.

YAGO2 Property	DBpedia Property
yago:actedIn	http://dbpedia.org/ontology/starring
yago:created **	—
yago:diedIn	http://dbpedia.org/ontology/deathPlace

[15] We solely relied on the http://dbpedia.org/ontology namespace within the DBpedia.

yago:directed	http://dbpedia.org/ontology/director
yago:hasAcademicAdvisor *	http://dbpedia.org/ontology/doctoralAdvisor
yago:hasChild	http://dbpedia.org/ontology/child
yago:hasOfficialLanguage	http://dbpedia.org/ontology/officialLanguage
yago:isCitizenOf **	—
yago:isKnownFor	http://dbpedia.org/ontology/knownFor
yago:isLeaderOf *	https://dbpedia.org/ontology/governor
yago:isLocatedIn	http://dbpedia.org/ontology/location
yago:isMarriedTo	http://dbpedia.org/ontology/spouse
yago:isPoliticianOf **	—
yago:livesIn	http://dbpedia.org/ontology/residence
yago:produced	http://dbpedia.org/ontology/producer
yago:wasBornIn	http://dbpedia.org/ontology/birthPlace

Table 6: The properties in the YAGO2 subset and their equivalents in DBpedia. * There are multiple options as equivalent property. ** There is no equivalent property in DBpedia.

Table 7 shows the number of triples that have been identified as correct by our knowledge graph veracity pipeline. As in previous experiments, we observe that COPAAL and FactCheck are able to identify different triples as correct, i.e., they complement each other. Another insight is that even if COPAAL cannot identify a large number of correct triples for one of the properties (e.g., yago:produced), the training of FactCheck enables it to derive evidence for triples with these properties.

Property	Triples	COPAAL	FactCheck	Facade	Veracity
actedIn	4949	1894	127	2021	41.09%
diedIn	759	416	37	453	59.68%
directed	226	102	11	113	50.00%
hasAcademic Advisor	132	111	0	111	84.09%
hasChild	2736	1870	45	1915	69.99%
hasOfficialLanguage	311	260	26	286	91.96%
isKnownFor	266	150	23	173	65.04%

isLeaderOf	176	9	27	36	16.27%
isLocatedIn	679	168	203	371	54.64%
isMarriedTo	1706	399	242	641	37.57%
livesIn	473	292	7	299	63.21%
produced	616	0	25	25	4.06%
wasBornIn	1826	1066	99	1165	63.80%
Sum	14855	6737	872	7609	51.22%

Table 7: Number of triples that have been proven to be correct by our knowledge graph veracity pipeline.

The last line of Table 7 shows that for more than 50% of the triples, our pipeline is able to confirm the correctness of these triples. Hence, these triples do not have to be checked manually, as suggested by Ojha et al. [24]. Instead, a human expert could check a subset of triples with properties for which our pipeline reports a low veracity, e.g., yago:produced or yago:isLeaderOf. This leads to similar results as the algorithm of Ojha et al. [24] but saves half of the manual validations that would have to be done by humans.
Note that our pipeline does not mark these triples as wrong but as triples that should be further checked. This is caused by the Open World Assumption which leads to the situation that it is not easy to identify a given triple as wrong since it might be true and our reference knowledge might be incomplete.

Summary

With the work described in this report, WP2 fulfilled all its goals.
The evaluation results of T2.1 show that the single services with their different approaches to verify a given fact work within the FROCKG platform. We also showed that we improved the runtime of COPAAL to enable fast processing of user requests.
For FactCheck, we saw a high impact of patterns that are used to generate queries to get relevant documents. This is an important insight for WP6 as we will have to come up with patterns for the properties that we want to use in the single use cases.
Finally, our results underline that the combination of several Fact Checking approaches leads to an improvement. Hence, the development of the facade as an extendable component was a good choice and allows the further addition of other Fact Checking approaches if necessary.
The evaluation results of T2.2 show the effectiveness of the knowledge graph veracity framework. Compared to a state-of-the-art approach [24], it saves 50% of the manual work that would have to be done by domain experts.

References

[1] Zafar Habeeb Syed, Michael Röder, and Axel-Cyrille Ngonga Ngomo: “FactCheck: Validating RDF Triples using Textual Evidence”. In Proceedings of the International Conference on Information and Knowledge Management (CIKM), 2018.

[2] Shiralkar, P., Flammini, A., Menczer, F., Ciampaglia, G.L.: “Finding streams in knowledge graphs to support fact checking”. In: 2017 IEEE International Conference on Data Mining (ICDM). pp. 859–864. IEEE (2017)

[3] Katz, L.: “A new status index derived from sociometric analysis”. Psychometrika 18 (1953)

[4] Xu, Z., Pu, C., Yang, J.: “Link prediction based on path entropy”. Physica A: Statistical Mechanics and its Applications 456, 294–301 (2016)

[5] Jeh, G., Widom, J.: Simrank: “A measure of structural-context similarity”. In: Proceedings of the Eighth ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining (2002)

[6] Adamic, L.A., Adar, E.: “Friends and neighbors on the web”. Social Networks 25(3) (2003)

[7] Liben-Nowell, D., Kleinberg, J.: “The link prediction problem for social networks”. In: Proceedings of the Twelfth Intern. Conf. on Information and Knowledge Management (2003)

[8] Shi, B., Weninger, T.: “Fact checking in large knowledge graphs - A discriminative predicate path mining approach”. CoRR abs/1510.05911 (2015)

[9] Shi, B., Weninger, T.: “Discriminative predicate path mining for fact checking in knowledge graphs”. Knowledge-based systems 104, 123–133 (2016)

[10] Lao, N., Cohen, W.W.: “Relational retrieval using a combination of path-constrained random walks”. Machine learning 81(1), 53–67 (2010)

[11] Zafar Habeeb Syed, Michael Röder, and Axel-Cyrille Ngonga Ngomo: “Unsupervised Discovery of Corroborative Paths for Fact Validation”. The Semantic Web – ISWC 2019 , page 630–646. Cham, Springer International Publishing, (2019)

[12] D. Gerber, D. Esteves, J. Lehmann, L. Bühmann, R. Usbeck, A. Ngonga Ngomo, and R. Speck: “DeFacto - Temporal and Multilingual Deep Fact Validation”. Web Semantics: Science, Services and Agents on the World Wide Web (2015)

[13] Satoshi Nakamura, Shinji Konishi, Adam Jatowt, Hiroaki Ohshima, Hiroyuki Kondo, Taro Tezuka, Satoshi Oyama, and Katsumi Tanaka: “Trustworthiness analysis of web search results”. Research and advanced technology for digital libraries (2007), 38–49.

[14] Deliverable 1.2: Component architecture and interfaces. FROCKG project. (2020)

[15] Deliverable 1.3: Requirements Specification. FROCKG project. (2020)

[16] Farzaneh Mahdisoltani, Joanna Biega, Fabian M. Suchanek: “YAGO3: A Knowledge Base from Multilingual Wikipedias”. Conference on Innovative Data Systems Research (2015).

[17] Bordes, Antoine and Usunier, Nicolas and Garcia-Duran, Alberto and Weston, Jason and Yakhnenko, Oksana: “Translating embeddings for modeling multi-relational data”. In Advances in neural information processing systems. 2787–2795 (2013).

[18] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie and Jian Tang. “RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space”. In CoRR, abs/1902.10197 (2019).

[19] Haonan Lu and Hailin Hu: “DensE: An Enhanced Non-Abelian Group Representation for Knowledge Graph Embedding”. In arXic, cs.AI, 2008.04548 (2020).

[20] Deliverable 2.1: First version of the fact checking algorithms including initial benchmarking results. FROCKG project. (2020)

[21] Ojha, Prakhar, and Talukdar, Partha: “KGEval: Accuracy estimation of automatically constructed knowledge graphs.” Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. (2017)

[22] Ana Alexandra Morim da Silva, Michael Röder and Axel-Cyrille Ngonga Ngomo: “Using Compositional Embeddings for Fact Checking”. In The Semantic Web – ISWC 2021. (2021)

[23] Alexander Bigerl, Felix Conrads, Charlotte Behning, Mohamed Ahmed Sherif, Muhammad Saleem and Axel-Cyrille Ngonga Ngomo: “Tentris – A Tensor Based Triple Store”. In The Semantic Web – ISWC 2020. (2020)

[24] Prakhar Ojha and Partha Talukdar: “KGEval: Accuracy Estimation of Automatically Constructed Knowledge Graphs”. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. (2017)

[25] Umair Qudus, Michael Röder, Muhammad Saleem, Axel-Cyrille Ngonga Ngomo: “HybridFC: A Hybrid Fact-Checking Approach for Knowledge Graphs”. To appear in The Semantic Web – ISWC 2022. (2022)

[26] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose and Fabian M. Suchanek: "AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases". In Proceedings of the 22nd international conference on World Wide Web. (2013)