Fact Checking via Evidence Patterns

Valeria Fionda[1], **Giuseppe Pirrò**[2],
[1] DeMaCS, University of Calabria, Rende, Italy
[2] ICAR-CNR, Rende, Italy
fionda@mat.unical.it, pirro@icar.cnr.it

Abstract

We tackle fact checking using Knowledge Graphs (KGs) as a source of background knowledge. Our approach leverages the KG schema to generate candidate evidence patterns, that is, schema-level paths that capture the semantics of a target fact in alternative ways. Patterns verified in the data are used to both assemble semantic evidence for a fact and provide a numerical assessment of its truthfulness. We present efficient algorithms to generate and verify evidence patterns, and assemble evidence. We also provide a translation of the core of our algorithms into the SPARQL query language. Not only our approach is faster than the state of the art and offers comparable accuracy, but it can also use any SPARQL-enabled KG.

1 Introduction

We live in a digital era where both false and true rumors spread at an unprecedented speed. Having a way to quickly assess the reliability of claims becomes crucial. Most of existing computational approaches (e.g., [Zubiaga *et al.*, 2017; Hassan *et al.*, 2017]) detect facts in large corpora (e.g., the Web, Twitter) and leverage indicators (e.g., tweet popularity) and other (mostly syntactic) features to measure their credibility. The success of these approaches depends on the accuracy of fact spotting, the capability to assemble background knowledge (e.g., related facts), and the response time.

The broad goal of this paper is to tackle fact checking from a semantic point of view by using Knowledge Graphs (KGs) like DBpedia and Yago that provide semantically-rich and automatically processable knowledge. KGs maintain atomic facts of the form (Dune, director, D. Lynch) structured thanks to a schema, which allows to understand the constituent parts of a fact (e.g., the fact is about a Film directed by a Person).

We present an approach that can contextualize facts by identifying semantically related facts (e.g., facts about Actors of a Film). Differently from previous work, which starts from the data to surface evidence for a fact, we leverage the schema and a predicate relatedness measure to generate candidate evidence patterns, that is, schema-compliant paths that capture the semantics of a fact in alternative ways. Semantic evidence is assembled from evidence patterns that are verified in the data. Interestingly, verified evidence patterns can be reused for facts of the same kind. Our approach is scalable thanks to efficient algorithms for candidate generation, verification and evidence building. We implemented our algorithms both in memory and by a translation into SPARQL [Harris *et al.*, 2013]. This latter aspect allows to use any SPARQL-enabled KG. Not only our approach offers an off-the-shelf fact checking solution, but it can also support existing approaches to seamlessly incorporate structured knowledge into the loop.

1.1 Related Work

Fact checking has been studied from different perspectives. [Gupta *et al.*, 2014; Ratkiewicz *et al.*, 2011] tackle misinformation spread in social networks; while considering contextual/user information (e.g., credibility) these approaches do not analyze the semantics of (related) facts. [Nickel *et al.*, 2016; Bordes *et al.*, 2013; Lin *et al.*, 2015] leverage embeddings for entities/predicates. Both the computation of embeddings and the fact checking process are not scalable and depend on the embedding. [Wu *et al.*, 2014] focus on specific types of facts only (i.e., window aggregates). ClaimBuster [Hassan *et al.*, 2017] detects claims in political discourses; neither it does consider structured knowledge nor it allows to build (reusable) semantic evidence. Logic-based approaches [Leblay, 2017] focus on how to represent/query facts and capture incompleteness/uncertainty; in [Leblay, 2017] no experimental evaluation is discussed.

Our work differs from rule-learning techniques (e.g., [Galárraga *et al.*, 2015; Gad-Elrab *et al.*, 2016]); we focus on fact checking for which we can quickly generate semantic evidence and provide a numerical assessment by proceeding from the schema (and not from the data) and only considering semantically related facts. Our approach differs from other KGs-based approaches (e.g., [Shiralkar *et al.*, 2017; Ciampaglia *et al.*, 2015; Shi and Weninger, 2016]) since: (i) instead of starting from the data to surface evidence, we start from the KG-schema, which allows to assemble semantic evidence both more quickly and precisely, by focusing first on evidence patterns that are semantically related with the target fact; (ii) our approach can provide verdicts in a few seconds, instead of hundreds of seconds like, for instance, [Shiralkar *et al.*, 2017]; (iii) our approach can scale to very large (online available) KGs thanks to the SPARQL-based implementation.

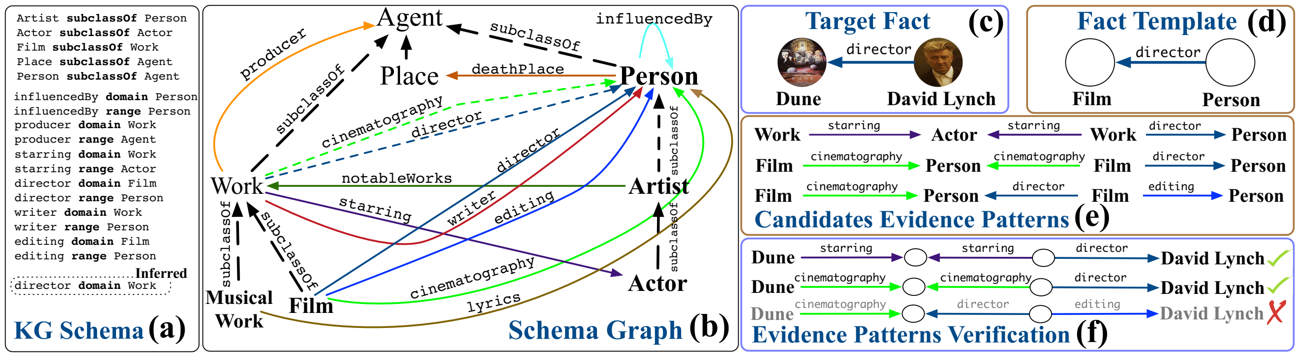

Figure 1: Given the KG schema (**a**), we build the schema graph (**b**). Given a fact (**c**), we generate a fact template by using type information (**d**) and then generate candidate evidence patterns using the schema graph (**e**) that will be verified in the data (**f**) to collect semantic evidence.

1.2 Running Example

Consider the fact (Dune, director, D. Lynch) show in Fig. 1 (c) and the DBpedia KG as source of background knowledge.

Schema Graph. Our approach leverages the class hierarchy, domain and range of predicates defined in the KG schema along with RDFS inference rules [Munoz *et al.*, 2009; Franconi *et al.*, 2013], to build a schema graph where nodes are classes (entity types) and edges predicates. Fig. 1 (a) shows an excerpt of the DBpedia schema. As an example, the predicate director has Film as one of its domains and Person as range; moreover, Film is subclassOf of Work. Fig. 1 (b) shows an excerpt of the corresponding schema graph; there are two edges labeled as director; the first: (Film, director, Person) directly derived from domain and range, while the second one: (Work, director, Person) obtained by deriving a new domain (i.e., Work) for the predicate director using RDFS inference.

Candidate Evidence Patterns. Our approach leverages the schema graph to generate candidate evidence patterns, that is, schema-compliant paths that provide an alternative way for expressing a target fact template (Fig. 1 (d)). As the number of potential (schema-compliant) paths can be large we adopt two pruning strategies. The first bounds pattern length to avoid the inclusion of pieces of knowledge that may hinder the contextualization/understanding of the fact. The second one restricts the types of predicates considered during the traversal to only those semantically related (see Section 3.1) to the target predicate. For instance, when checking a fact about director predicates like population can be misleading.

Fig. 1 (e) shows some evidence patterns of length 3 when considering the top-3 most predicates related to director (e.g, director, starring, cinematography,). The first pattern in Fig. 1 (e) shows an alternative way of expressing the fact template in Fig. 1 (d): a Film (the superclass Work in this case) can have an outgoing starring edge that links it to entities of type Actor in the underlying KG, which in turns have incoming edges from entities of type Work via starring edges; these are finally linked to entities of type Person via director edges. Candidate evidence pattern are generated by traversing the schema graph starting from the source entity types toward the target entity types; in our example from {Film, Work} (for Dune) to Person (for D. Lynch).

Evidence Patterns Verification. Candidate evidence patterns tell us how the subject and object entity types of a fact could be connected in the data; however, not all of them are verified (i.e., have bindings) in the data. Candidate patterns verification is performed by: (i) instantiating the pattern endpoints with the subject and object of the target statement (Fig. 1 (f)) and; (ii) traversing the data graph to check whether the subject can reach the object according to the sequence of predicates in the pattern. To verify the first evidence pattern, our verification algorithm starts from Dune in the KG and by traversing the sequence of edges (according to their directions) starring←starring→director checks if D. Lynch can be reached. By traversing starring edges from Dune it is possible to reach entities like J. Nance; from here, by traversing starring edges backward, nodes like Twin Peaks can be reached. From this latter set of nodes, director edges are traversed forward toward D. Lynch. In this example, only the first two patterns are verified in the KG data. Evidence (paths in the KG) from verified patterns allows to build an evidence graph.

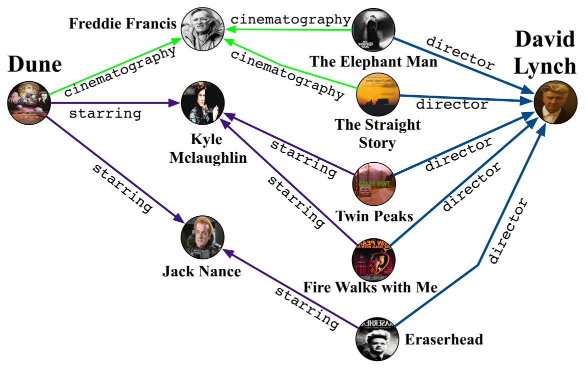

Figure 2: Evidence supporting the fact (Dune, director, D. Lynch).

Fig. 2 shows an excerpt of the evidence graph built from the first two patterns in Fig. 1 (e). Actors like K. Mclaughlin and J. Nance starred Dune and other movies directed by D. Lynch; F. Francis was responsible for the cinematography of Dune and Elephant Man directed by D. Lynch. These chains of facts support the (true) fact that D. Lynch directed Dune.

Evidence score. Evidence as shown in Fig. 2 takes some human effort to parse and understand. To fully automate the fact checking process and numerically distill evidence we devised the evidence score. Given a pattern, the score takes into account both in *how many ways* the subject to the object are linked, and *how specific* are the object and the subject. As an example, the first and the second evidence patterns link the subject and the object in 3 and 2 different ways, respectively (see Fig. 2). Moreover, the first evidence pattern is more specific than the second wrt the subject; this is because if instead of D.Lynch there were an arbitrary Person then the first pattern and second pattern would have 488 and 26 counts, respectively. Frequency and specificity can be combined according to different similarity measures (e.g., DICE, Tversky) to provide an evidence score between 0 and 1. In our example, for the first evidence pattern in Fig. 1 (e) the evidence score is 0.22 by using the Tversky's measure. A final evidence score can be assembled by averaging the scores from individual (verified) patterns.

Reusing Patterns. Consider the fact (Kill Bill, director, Q. Tarantino). To check this fact we can reuse the candidate evidence patterns generated in the previous example because the fact template (Fig. 1 (d)) is the same. Even in this case the first pattern is verified in the data as S. Jackson also acted in Kill Bill 2 and Jango Unchained that were directed by Q. Tarantino. False facts like (Dune, director, A. Jodorowski) or (The Godfather Part II, director, Michael Mann) do not verify any candidate pattern in Fig. 1 (e).

1.3 Contributions and Outline

We tackle the problem of fact checking by a novel approach that generates evidence patterns from the KG schema and use them to assemble semantic evidence from the data. We contribute: **(i)** a fact checking approach based on the idea of evidence patterns; **(ii)** efficient algorithms to generate and verify evidence patterns, and assemble semantic evidence; **(iii)** the evidence score as a final distillation of the fact checking process; **(iv)** two implementations (one in-memory and another SPARQL-based), an experimental evaluation, and comparison with related research.

The remainder of the paper is organized as follows. We introduce some background in Section 2. Section 3 describes our approach. The implementation and experimental evaluation is discussed in Section 4. We conclude in Section 5.

2 Preliminaries

A Knowledge Graph (KG) is a directed node and edge labeled multi-graph $G{=}(V, E, T)$ where V is a set of uniquely identified vertices representing entities (e.g., D. Lynch, France), E a set of predicates (e.g., director, nationality) and T a set of statements of the form (s, p, o) representing directed labeled edges, where s, o $\in V$ and p $\in E$. To structure knowledge, KGs resort to an underlying schema. In this paper we focus on the portion of the schema that concerns entity *types* (and their hierarchy) and *domain* and *range* of predicates. Our approach leverages RDFS inference rules to construct the RDFS-closure of the schema [Munoz *et al.*, 2009; Franconi *et al.*, 2013]. From the closure of the schema we build the corresponding schema graph closure.

Definition 1. *(Schema Graph Closure). Given a KG schema, its closure is defined as $G_S{=}(V_s, E_s, T_s)$, where each $v_i \in V_s$ is a node denoting an entity type, each $p_i \in E_s$ denotes a predicate and each $(v_s, p_i, v_t) \in T_s$ is a triple (asserted or inferred via RDFS reasoning) where v_i (resp., v_t) is the domain (resp., range) of the predicate p_i.*

Fig. 1 (a) shows an excerpt of the DBpedia schema while Fig. 1 (b) of the schema closure; here, dashed lines represent inferred triples. For instance, the triple (Work, director, Person) is inferred from the triples (Film, subclassOf, Work) and (Film, director, Person). Given a predicate p, $Dom(p)$ (resp., $Range(p)$) denotes the set of nodes (i.e., entity types) in G_S having an outgoing (resp., incoming) edge labeled with p. We refer to the schema graph closure simply as the schema graph.

Definition 2. *(Evidence Pattern). Given a schema graph $G_S{=}(V_s, E_s, T_s)$, a target predicate p^* and an integer d, an evidence pattern is a sequence $\Pi(p^*) = \pi_1 \cdot \pi_2, ... \cdot \pi_{l\text{-}1} \cdot \pi_l$, with $l \leq d$, such that $\pi_i{=}(t_i, \otimes p_i, t_{i+1})$ for $i \in \{1, ..., l\}$, $\otimes \in \{\rightarrow, \leftarrow\}$, $t_i \in V_s$ for $i \in \{1, ...l\}$, $p_j \in E_s$ for $j \in \{1, ...l\text{-}1\}$, $t_1 \in Dom(p^*)$, $t_l \in Range(p^*)$ and $\cdot$ denotes the composition (i.e., join) operator.*

The operator $\otimes \in \{\rightarrow, \leftarrow\}$ determines the direction of an edge in the pattern. By looking at Fig. 1 (b), an example of evidence pattern is Π(director)=$\pi_1 \cdot \pi_2 \cdot \pi_3$ where π_1=(Work, →starring, blueActor), π_2=(blueActor, ←starring, brownWork), π_3=(brownWork ←director, Person).

3 Fact Checking via Evidence Patterns

To verify a fact $(\mathsf{s}, p^*, \mathsf{o})$ our approach proceeds as follows: (i) generate candidate evidence patterns (if not already available) using the schema graph and predicate relatedness; (ii) verify evidence patterns and build an evidence graph; (iii) optionally, output an evidence score.

3.1 Predicate Relatedness

The first ingredient of our approach is a predicate relatedness measure, which allows to isolate the portion of the schema (and thus of the underlying data) that is of most interest for a particular fact. Its goal is to rank predicates in a graph $G = (V, E, T)$ wrt p^*. Given $(p_i, p_j){\in}E$, the relatedness measure takes into account the co-occurrence of p_i and p_j in the set of triples T and weights it by the predicate popularity. This approach resembles the TF-ITF scheme used in information retrieval [Shiralkar *et al.*, 2017; Pirrò, 2012]. We now introduce its building blocks.

$$TF(p_i, p_j) = \log(1 + C_{i,j}) \tag{1}$$

where $C_{i,j}$ counts the number of s and o in triples of the form (s, p_i, o) and (s, p_j, o) or (o, p_i, s) and (o, p_j, s).
ITF is defined as follows:

$$ITF(p_j, E) = \log \frac{|E|}{|\{p_i : C_{i,j} > 0\}|} \tag{2}$$

Having co-occurrences for each pair (p_i, p_j) weighted by the ITF allows to build a co-occurrence matrix:

$$w_{i,j}(p_i, p_j, E) = TF(p_i, p_j) \times ITF(p_j, E) \tag{3}$$

The relatedness between p_i and p_j is then computed as:

$$Rel(p_i, p_j) = Cosine(W_i, W_j) \tag{4}$$

where W_i (resp., W_j) is the row of p_i (resp., p_j).

3.2 Finding Evidence Patterns

We now describe the candidate evidence patterns generation algorithm (Algorithm 1). It takes as input a schema graph, a predicate p^* (the predicate in the target fact), an integer d to bound the length of the pattern, an integer k to pick the top-k most related predicates to p^*, and uses a priority queue to store candidate patterns ranked by their relatedness wrt p^*.

Algorithm 1: `findEvidencePatterns`

```
Input: A schema graph $G_S$=$(V_s, E_s, T_s)$, a target predicate
       $p^*$, maximum depth $d$, number of related predicates $k$
Output: Evidence Patterns $\mathcal{P}$

1  $\mathcal{P} = \emptyset$; /* priority queue based on average relatedness to $p^*$ */
2  $R$= getRelatedPredicates($p^*$,$k$)
3  Parallel execution:
4  for each predicate $p_i \in R$ do
5      Let $\Delta_1 = \emptyset$
6      for $(t_r, p_i, t_s) \in Ts$ do
7          if $t_r \in sourceNodes(p^*)$ then
8              $\Delta_1$=$\Delta_1 \cup \{(t_r, \rightarrow p_i, t_s)\}$
9              if $t_s \in targetNodes(p^*)$ then
10                 $\mathcal{P} = \mathcal{P} \cup \{(t_r, \rightarrow p_i, t_s)\}$
11         if $t_s \in sourceNodes(p^*)$ then
12             $\Delta_1$=$\Delta_1 \cup \{(t_r, \leftarrow p_i, t_s)\}$
13             if $t_r \in targetNodes(p^*)$ then
14                 $\mathcal{P} = \mathcal{P} \cup \{(t_r, \leftarrow p_i, t_s)\}$
15     for $j = 2, ..., d$ do
16         Let $\Delta_j = \emptyset$;
17         for each evidence pattern $\pi \in \Delta_{j-1}$ do
18             for each entity type $n_s \in outNodes(\pi)$ do
19                 for each triple $(n_s, p, n_t) \in Ts$ s.t. $p \in R$ do
20                     $\Delta_j = \Delta_j \cup \{\pi \cdot (n_s, \rightarrow p, n_t)\}$
21                 for each triple $(n_t, p, n_s) \in Ts$ s.t. $p \in R$ do
22                     $\Delta_j = \Delta_j \cup \{\pi \cdot (n_s, \leftarrow p, n_t)\}$
23         for each evidence pattern $\Pi_j \in \Delta_j$ do
24             if (checkTypes($\pi_j, p^*$) = true) then
25                 $\mathcal{P} = \mathcal{P} \cup \{\Pi_j\}$
26 return $\mathcal{P}$
```

The algorithm for each of the most related predicates (line 4) in parallel: performs a traversal of the schema graph by checking that predicate p_i and predicate p^* both have as source node t_r (lines 7) (resp., t_s (line 11)); this allows to build length-1 evidence patterns that are added to the results if they allow to reach the same node t_s (line 9) (resp., t_r (line 13)). Length-1 evidence patterns are expanded (line 11-25). The algorithm takes a q-length (partial) pattern (with $q<d$) and expands it by only considering the top-k most related predicates to p^* (lines 19 and 21). The expansion is done both in forward and reverse direction since the schema graph is treated as undirected. Evidence patterns of at most length d are added to the results (line 25) after checking that the pattern starts from one of the types in $Dom(p^*)$ and ends in one of the types in $Range(p^*)$ (line 24).

Theorem 1. *Algorithm 1 runs in time $O(d \times |E_s|)$.*

Proof sketch. Algorithm 1 runs (in parallel) a d-step traversal of the graph and at each step at most $|E_s|$ edges (note that the algorithm in practice considers a subset R of all predicates) can be visited. `checkTypes` can be done in constant type by hashing domain(s) and range(s) of predicates. □

3.3 Verifying Evidence Patterns

Candidate evidence patterns found via Algorithm 1 represent schema-compliant ways in which the endpoints of the target predicate *could* be connected in the underlying data. To understand which evidence patterns actually contribute to build semantic evidence for $(\mathtt{s}, p^*, \mathtt{o})$, there is the need to verify them. A pattern is verified if starting from s it is possible to reach o by traversing the sequence of edges in it. When instantiating the endpoints of the first evidence pattern in Fig. 1 (e), we obtain the evidence pattern in Fig. 1 (f). This pattern is actually verified as shown in Fig. 2. On the other hand, the last evidence pattern in Fig. 1 (e) is not verified. Algorithm 2 handles evidence pattern verification and builds an evidence graph for the target fact $(\mathtt{s}, p^*, \mathtt{o})$.

Algorithm 2: `buildEvidenceGraph`

```
Input: A knowledge graph $G$=$(V, E, T)$, a fact $(\mathtt{s}, p^*, \mathtt{o})$,
       candiate evidence patterns $\mathcal{P}$, number of evidence
       patterns $h$
Output: Evidence graph $G_e = (V_e, E_e)$ for $(\mathtt{s}, p^*, \mathtt{o})$
1  $\mathcal{P}'$=filterPatterns($\mathcal{P}$, $h$)
2  $e$ = getQuery($\mathcal{P}'$) /* query used in the verification */
3  build the automaton $\mathcal{A}_e$ associated to $e$
4  $G \times \mathcal{A}_e$ = build($G, \mathcal{A}_e$, s) /* build the product automaton */
5  $G_e = \emptyset$
6  reached $= \emptyset$
7  toVisit $= \bigcup_{(\mathtt{o}, q_f) \in F} \{(\mathtt{o}, q_f)\}$ s.t. $q_f \in F$ /* backwards */
8  while toVisit $\neq \emptyset$ do
9      $(n, q)$ = removeFirst(toVisit)
10     reached = reached $\cup \{(n, q)\}$
11     for transition $\delta((n', q'), x) \in G \times \mathcal{A}_e$ s.t.
       $(n, q) \in \delta((n', q'), x)$ do
12         if $(n', q') \notin$ reached then
13             toVisit = toVisit $\cup \{(n', q')\}$
14             if $x \in E$ then
15                 add $(n, x, n')$ to $G_e$
16             else
17                 add $(n', x, n)$ to $G_e$
18 return $G_e$
```

The algorithm allows to select a subset of candidate evidence patterns (line 1) ranked according to the average relatedness (of their predicates) wrt p^*. Given an evidence pattern Π_i=$\pi_1 \cdot ... \cdot \pi_l$, with π_i=$(t_i, \otimes p_i, t_{i+1})$ for $i \in \{1, ..., l\}$, by concatenating the sequence of predicates $p_i \in \pi_i$, with $i \in \{1, ..., l\}$, we can build the expression e_i=$\bar{p}_1 \cdot ... \cdot \bar{p}_l$ where $\bar{p}_i$=p_i if $\otimes$=$\rightarrow$ and $\bar{p}_i$=p_i^- if $\otimes$=$\leftarrow$. The expression e_i is a *navigational query* (e.g., SPARQL property paths [Fionda *et al.*, 2015a]) that can express in a succinct way (viz. without the usage of intermediate variables) a graph traversal. Given a set of candidate patterns $\mathcal{P}'$=$\{\Pi_1, \Pi_2, ..., \Pi_n\}$ our algorithm builds (line 2) the navigational query e=$(e_1 \mid e_2 \mid .. \mid e_n)$, where $\mid$ is the disjunction operator and e_i is the query associated to Π_i, for $i \in \{1, ..., n\}$. Moreover, the size of e is $|e| = O(||\mathcal{P}'||)$ with $||\mathcal{P}'||$=$|\Pi_1|$+...+$|\Pi_n|$. As an example, consider the first two evidence patterns in Fig. 1 (e), the corresponding navigational query is: e=($\mathsf{starring} \cdot \mathsf{starring}^- \cdot \mathsf{director} \mid \mathsf{cinematography} \cdot \mathsf{cinematography}^- \cdot \mathsf{director}$).

To verify e we use automata theory. In polynomial time, it is possible to translate e into a Nondeterministic Finite-state Automaton (NFA) $\mathcal{A}_e$=$(Q, \Sigma_e, \delta, q_0, F)$ (line 3) where: Q is the set of states of $\mathcal{A}_e$, $\delta : Q \times (\Sigma_e \cup \epsilon) \rightarrow 2^Q$ the transition function, $q_0 \in Q$ the initial state and $F \subseteq Q$ the set of accepting (final) states. $\mathcal{A}_e$ is used to recognize strings (sequences of edge labels in G) spelled by e [Hopcroft *et al.*, 2006]. To verify e over a KG G, we can build the product $G \times \mathcal{A}_e$ [Pérez *et al.*, 2010; Fionda *et al.*, 2015b] in time $O(|G|{\times}|e|)$=$O(|G|{\times}||\mathcal{P}'||)$, where $|G| = |V| + |E|$. The product is built starting from s and navigating *in parallel* the automaton (which is a graph) and the data graph and marking for each state $q \in Q$ entities in G that have been reached at that state. If o is reached at some final state $q_f \in F$, then e is satisfied and thus some $\Pi_i \in \mathcal{P}'$ is verified. The evidence graph G_e is built according to Algorithm 2. The idea is to start with an empty G_e (line 5) and navigate the product automaton *backward*, from the final states $(\mathsf{o}, q_f) \in F$ and adding to G_e (lines 15 and 17) only nodes/edges that lead to the initial state $(\mathsf{s}, q_0) \in Q$.

Theorem 2. *Algorithm 2 runs in time* $O(|G|{\times}||\mathcal{P}'||)$.

Proof sketch. Each state and each transition (in the backward direction) is visited at most once with cost $O(|G|{\times}|\mathcal{A}|_e) = O(|G|{\times}||\mathcal{P}'||)$. The total cost, when also considering the cost of building the product automaton, is $O(|G|{\times}||\mathcal{P}'||)$. □

3.4 Evidence Score

While an evidence graph can be useful to visually inspect evidence that supports or disprove a fact, its parsing and understanding requires human effort. To fully automatize the fact checking process and distillate evidence collected for a fact $(\mathsf{s}, p^*, \mathsf{o})$ we devised the evidence score. It builds upon three main ingredients. The first concerns *how often* s and o are linked according to an evidence pattern Π.

$$Fr(\Pi, \mathsf{s}, \mathsf{o}) = count(\mathsf{s}, \mathsf{o}, \Pi) \tag{5}$$

The rationale is that the more s and o are connected the stronger is the support (in terms of evidence) of Π. As an example, the first and second evidence patterns in Fig. 1 (b) allow to link Dune to $\mathsf{D.\ Lynch}$ in 3 and 2 different ways, respectively. The second component takes into account the specificity of Π wrt s:

$$Spec_s(\Pi, \mathsf{s}, \mathbf{x}) = count(\mathsf{s}, \mathbf{x}, \Pi) \tag{6}$$

where $\mathbf{x}$ plays the role of a variable (i.e., the object is not fixed). $Spec_s$ counts how many other entities $\mathbf{x}$ are linked to s via Π. Likewise we can define the specificity for o.

$$Spec_o(\Pi, \mathbf{x}, \mathsf{o}) = count(\mathbf{x}, \mathsf{o}, \Pi) \tag{7}$$

For $(\mathsf{Dune}, \mathsf{director}, \mathsf{D.\ Lynch})$ and the first evidence pattern Π_1 in Fig. 1 (f) then $Spec_s(\Pi_1, \mathsf{Dune}, \mathbf{x})$)=488 and $Spec_o(\Pi_1, \mathbf{x}, \mathsf{D.\ Lynch})$ =1657. The evidence score ($\mathcal{E}$) is obtained via a similarity function f (e.g., Tversky, DICE).

$$\mathcal{E}(\mathsf{s}, \mathsf{o}, \Pi) = f(Fr, Spec_s, Spec_o) \tag{8}$$

As an example when f=Tversky we have:

$$\mathcal{E}(\mathsf{s}, \mathsf{o}, \Pi) = \frac{Fr}{Fr + Spec_s + Spec_o} \tag{9}$$

Given a set of patterns $\mathcal{P}$, the overall evidence score is:

$$\mathcal{E}(\mathsf{s}, \mathsf{o}, \mathcal{P}) = \mathcal{E}(\mathsf{s}, \mathsf{o}, \Pi_i)/|\mathcal{E}(\mathsf{s}, \mathsf{o}, \mathcal{P})|,\ \Pi_i \in \mathcal{P},\ i \in [1, |\mathcal{P}|] \tag{10}$$

4 Implementation and Evaluation

We implemented our algorithms in the `CHEEP` (`CH`eck via `E`videnc`E` `P`tterns) system using: (i) Java with the KG loaded in memory (`CHEEP`-M); and, (ii) by accessing the KG and verifying patterns via SPARQL (`CHEEP`-Q). Predicate relatedness values have been precomputed. Moreover, the evidence is calculated via SPARQL (count) queries. The implementations are available online[1].

Evaluation methodology. We compared our approach against the following systems by using available implementations [KStream, 2018]: Knowledge Linker (`KL`) [Ciampaglia *et al.*, 2015], PredPath (`PP`) [Shi and Weninger, 2016] and the best configuration of KStream (`KS`) reported in [Shiralkar *et al.*, 2017]. As for `CHEEP`, the notation `CHEEP`(#Predicates,#Patterns) represents a configuration for a given number of predicates related to p^* and evidence patterns to be verified. We considered `CHEEP`(10,5), and `CHEEP`(10,10). We used the evidence score in eq. (9) and d =3 in Algorithm 1. Experiments have been performed on a MacBook Pro 2.6 GHz Intel Core i5 with 8Gbs memory.

To test the performance, we used the area under the Receiver Operating Characteristic Curve (AUC) as also done in previous work [Shiralkar *et al.*, 2017]. AUC allows to compare the accuracy across datasets with different ratios of true and false facts. Each method gives as output a numeric score for each target fact, and the AUC is useful to express the probability that a true fact receives a higher score than a false one.

Knowledge Graphs and Datasets. We considered the DBpedia dataset used by [Shiralkar *et al.*, 2017] (24M triples, and 663 predicates). We also tested `CHEEP`-Q using the whole DBpedia dataset (438M triples and 663 predicates) by accessing DBpedia via its endpoint[2]. For the evaluation we considered 10 datasets used in Shiralkar et al. [Shiralkar *et al.*, 2017] and for which the ground truth is available [KStream, 2018]. The first 5 are real-world datasets derived from Google Relation Extraction Corpora and WSDM Cup Triple Scoring challenge while the last 5 synthetic datasets mix a priori known true and false facts.

Results. Table 1 summarizes the results in terms of average AUC (real world datasets are reported on the left). For `CHEEP` we report two scores, referring to `CHEEP`-M (on the small DBpedia) and `CHEEP`-Q (on the full DBpedia), respectively. Results show that our approach is competitive wrt the state of the art. In particular, `CHEEP` performs slightly better than `KS` (the most direct competitor) in almost all real-world datasets. Despite the similar performance, there are crucial

[1]`https://factcheckingkgs.wordpress.com`
[2]`http://dbpedia.org/sparql`

Approach	**birthPlace** (273/1092)	**deathPlace** (126/504)	**almaMater** (1546/6184)	**nationality** (50/200)	**profession** (110/440)	**author** (93/558)	**team** (41/164)	**director** (78/4680)	**keyPerson** (201/1208)	**spouse** (16/256)
CHEEP-M/Q (10,5)	.84/.88	.85/.87	.75/.75	.83/.84	.97/.98	.91/.93	89/.90	.98/.99	.81/.82	.95/.95
CHEEP-M/Q (10,10)	.89/.91	.86/.87	.76/.77	.83/.85	.97/.98	90/.91	.89/91	.99/.99	.81/.82	.96/.96
KStream (KS)	.82	.84	.75	.93	.93	.92	.99	.83	.80	.86
KLinker (KL)	.91	.87	.78	.86	.93	.96	.92	.88	.83	.91
PredPath (PP)	.86	.76	.83	.95	.92	.99	.92	.84	.88	.87

Table 1: Performance (average AUC) on both real-world (left) and synthetic (right) datasets (average of 4 runs). The number of true facts over all facts is reported below the predicate name. For CHEEP, the two scores refer to CHEEP-M/CHEEP-Q using the same configuration.

differences between these approaches: KS (in its best configuration [Shiralkar *et al.*, 2017]) includes information from the two paths that maximize the flow between the subject and the object. However, interpreting the flow value as an indicator of the truthfulness of a fact is difficult. On the other hand, CHEEP computes the evidence score between 0 and 1 thus offering a more direct interpretation. Another crucial difference is the fact that CHEEP can control the amount of evidence that will be used to determine the score by deciding to include/remove candidate evidence patterns. In this respect, we noted that increasing the number of evidence patterns in most of the cases brings slightly better performance with the exception of author; we can explain this behavior by the fact that including more patterns that give a low evidence score may decrease the final (average) score. When not bounding the number of paths considered by KS, its performance degrades [Shiralkar *et al.*, 2017]. CHEEP behaves similarly to PP but has a different departure point; PP considers paths in the data graph that better discriminate the subject and the object of the target statement while CHEEP starts from the schema and only verifies promising (i.e., semantically related to p^*) patterns. Moreover, differently from the evidence score produced by CHEEP, the output of PP is not easily interpretable as a final distillation of the truthfulness of the statement. We observe that when using the whole DBpedia dataset performance obtains a further increase. This may be explained by the fact that more data allows to discover additional ways (i.e., paths in the data) that can verify patterns thus contributing to increase the evidence score. Note that CHEEP-Q (the variant based on SPARQL) is the only system that can handle the whole DBpedia dataset by accessing it via its (remote) SPARQL endpoint.

Running time. Table 2 reports the average runtime over all datasets. We can notice that CHEEP-M obtains performance comparable to KL and PP while is much faster than KS. This can be explained by the fact that our algorithms for both candidate evidence patterns generation and verification run in polynomial time. On the other hand, KS runs in (pseudo polynomial) time $O(\gamma|E|log|V|)$, where γ (the maximum flow) is a parameter that has to be determined by the algorithm.

The variant of CHEEP using SPARQL queries (CHEEP-Q) offers lower running times while being able to seamlessly handle a much larger amount of data. The good performance can be explained by the fact that SPARQL processors rely on efficient optimizers and index structures while the other algorithms work in main memory (with no indexing and query optimization strategies). Overall, using CHEEP-Q appears to be the best option both for the low running times and the possibility to use it in any (remote) SPARQL-enabled KG.

Method	**DBpedia-small**	**DBpedia-full**
KS	280s	-
KL	32s	-
PP	43s	-
CHEEP-M	32s	-
CHEEP-Q	5s	7s

Table 2: Average running times (seconds).

Predicate	**Evidence Pattern**
nationality	Person $\xrightarrow{\text{residence}} \cdot \xleftarrow{\text{deathPlace}} \cdot \xrightarrow{\text{residence}}$ Place
	Person $\xrightarrow{\text{birthPlace}} \cdot \xleftarrow{\text{residence}} \cdot \xrightarrow{\text{ethnicity}}$ Place
birthPlace	Person $\xrightarrow{\text{birthPlace}} \cdot \xleftarrow{\text{residence}} \cdot \xrightarrow{\text{stateOfOrigin}}$ Place
	Person $\xrightarrow{\text{birthPlace}} \cdot \xleftarrow{\text{residence}} \cdot \xrightarrow{\text{ethnicity}}$ Place
keyPerson	Company $\xleftarrow{\text{parentCompany}} \cdot \xrightarrow{\text{keyPerson}}$ Person
	Company $\xleftarrow{\text{owningCompany}} \cdot \xrightarrow{\text{keyPerson}}$ Person
spouse	Person $\xrightarrow{\text{child}} \cdot \xleftarrow{\text{child}}$ Person
	Person $\xleftarrow{\text{parent}} \cdot \xrightarrow{\text{parent}}$ Person

Table 3: Some evidence patterns found by CHEEP.

Evidence patterns. Table 3 shows some evidence patterns found by CHEEP. We can note, for instance, that spouse is captured by the fact that two people have the same child(ren) or by a child (or children) having the same parents; moreover, when someone is a keyPerson of a Company it is also the keyPerson of a parentCompany. We want to mention that the decoupling of evidence pattern generation and verification in CHEEP and its SPARQL-based implementation allows to: (i) select patterns to be verified based on the relevance (and semantic relatedness) wrt a target predicate beforehand; (ii) discover meaningful patterns verified in the data; (iii) seamlessly work with any SPARQL-enabled (remote) KG. Finally, patterns discovered via CHEEP form an interesting body of knowledge as they can be used in applications like information extraction or KG completion [Galárraga *et al.*, 2015].

5 Concluding Remarks and Future Work

We described an approach for fact checking based on KGs. Differently from previous work, it makes usage of the KG-schema to find possible forms of evidence for a fact by only focusing on the portion of the graph that is most semantically related to it. This design choice guarantees lower running times and the possibility to reuse evidence for facts of the same kind. We devised polynomial algorithms for both candidate generation and verification. A further advantage and novelty of our approach is the translation of the core of our algorithms into SPARQL queries. Not only this allows to verify facts in a few seconds, but also to apply our approach in any existing SPARQL-enabled KG. In the future we plan to combine knowledge from multiple KGs and include negative information in the verification process.

References

[Bordes *et al.*, 2013] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[Ciampaglia *et al.*, 2015] Giovanni Luca Ciampaglia, Prashant Shiralkar, Luis M Rocha, Johan Bollen, Filippo Menczer, and Alessandro Flammini. Computational Fact Checking from Knowledge Networks. *PloS one*, 10(6), 2015.

[Fionda *et al.*, 2015a] Valeria Fionda, Giuseppe Pirrò, and Mariano P. Consens. Extended Property Paths: Writing more SPARQL Queries in a Succinct Way. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI)*, pp. 102–108, 2015.

[Fionda *et al.*, 2015b] Valeria Fionda, Giuseppe Pirrò, and Claudio Gutiérrez. NautiLOD: A Formal Language for the Web of Data Graph. *ACM Transactions on the Web*, 9(1), art. 5, pp. 1:43, 2015.

[Franconi *et al.*, 2013] Enrico Franconi, Claudio Gutierrez, Alessandro Mosca, Giuseppe Pirrò, and Riccardo Rosati. The Logic of Extensional RDFS. In *Proceedings of the 12th International Semantic Web Conference*, pp. 101–116, 2013.

[Gad-Elrab *et al.*, 2016] Mohamed H Gad-Elrab, Daria Stepanova, Jacopo Urbani, and Gerhard Weikum. Exception-enriched Rule Learning from Knowledge Graphs. In *Proceedings of the 15th International Semantic Web Conference*, pp. 234–251, 2016.

[Galárraga *et al.*, 2015] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *The VLDB Journal*, 24(6):707–730, 2015.

[Gupta *et al.*, 2014] Aditi Gupta, Ponnurangam Kumaraguru, Carlos Castillo, and Patrick Meier. Tweetcred: Real-time Credibility Assessment of Content on Twitter. In *International Conference on Social Informatics*, pages 228–243. Springer, 2014.

[Harris *et al.*, 2013] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 Query Language. *W3C recommendation*, 21(10), 2013.

[Hassan *et al.*, 2017] Naeemul Hassan, Fatma Arslan, Chengkai Li, and Mark Tremayne. Toward Automated Fact-Checking: Detecting Check-worthy Factual Claims by ClaimBuster. In *KDD*, pages 1803–1812, 2017.

[Hopcroft *et al.*, 2006] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation, Third Edition*. Addison-Wesley, 2006.

[KStream, 2018] KStream. https://github.com/shiralkarprashant/knowledgestream, 2018.

[Leblay, 2017] Julien Leblay. A Declarative Approach to Data-driven Fact Checking. In *Proceedings of the 31th Conference on Artificial Intelligence (AAAI)*, pp. 147–153, 2017.

[Lin *et al.*, 2015] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI)*, pp. 2181–2187, 2015.

[Munoz *et al.*, 2009] Sergio Munoz, Jorge Pérez, and Claudio Gutierrez. Simple and Efficient Minimal RDFS. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):220–234, 2009.

[Nickel *et al.*, 2016] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[Pérez *et al.*, 2010] Jorge Pérez, Marcelo Arenas, and Claudio Gutiérrez. nsparql: A Navigational Language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[Pirrò, 2012] Giuseppe Pirrò. REWOrD: Semantic Relatedness in the Web of Data. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI)*, pp. 129–135, 2012.

[Ratkiewicz *et al.*, 2011] Jacob Ratkiewicz, Michael Conover, Mark Meiss, Bruno Gonçalves, Snehal Patil, Alessandro Flammini, and Filippo Menczer. Truthy: mapping the Spread of Astroturf in Microblog Streams. In *Proceedings of the 20th international conference companion on World wide web*, pp. 249–252. ACM, 2011.

[Shi and Weninger, 2016] Baoxu Shi and Tim Weninger. Discriminative Predicate Path Mining for Fact Checking in Knowledge Graphs. *Knowledge-Based Systems*, 104:123–133, 2016.

[Shiralkar *et al.*, 2017] Prashant Shiralkar, Alessandro Flammini, Filippo Menczer, and Giovanni Luca Ciampaglia. Finding Streams in Knowledge Graphs to Support Fact Checking. In *ICDM*, pages 859–864, 2017.

[Wu *et al.*, 2014] You Wu, Pankaj K Agarwal, Chengkai Li, Jun Yang, and Cong Yu. Toward Computational Fact-Checking. *Proceedings of the VLDB Endowment*, 7(7):589–600, 2014.

[Zubiaga *et al.*, 2017] Arkaitz Zubiaga, Ahmet Aker, Kalina Bontcheva, Maria Liakata, and Rob Procter. Detection and Resolution of Rumours in Social Media: A Survey. *ACM Computing Surveys*, 51(2), 2018.