A Benchmark for Fact Checking Algorithms Built on Knowledge Bases

Viet-Phi Huynh
viet-phi.huynh@eurecom.fr
Eurecom
France

Paolo Papotti
papotti@eurecom.fr
Eurecom
France

ABSTRACT

Fact checking is the task of determining if a given claim holds. Several algorithms have been developed to check claims with reference information in the form of facts in a knowledge base. While individual algorithms have been experimentally evaluated in the past, we provide the first comprehensive and publicly available benchmark infrastructure for evaluating methods across a wide range of assumptions about the claims and the reference information. We show how, by changing the *popularity*, *transparency*, *homogeneity*, and *functionality* properties of the facts in an experiment, it is possible to influence significantly the performance of the fact checking algorithms. We introduce a benchmark framework to systematically enforce such properties in training and testing datasets with fine tune control over their properties. We then use our benchmark to compare fact checking algorithms with one another, as well as with methods that can solve the *link prediction task* in knowledge bases. Our evaluation shows the impact of the four data properties on the qualitative performance of the fact checking solutions and reveals a number of new insights concerning their applicability and performance.

ACM Reference Format:
Viet-Phi Huynh and Paolo Papotti. 2019. A Benchmark for Fact Checking Algorithms Built on Knowledge Bases. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3358036

1 INTRODUCTION

Fact checking is the task of verification of textual content. While fact checking has historically been an activity for journalists, the increase of incorrect claims over the Web has motivated the study of computational methods to identify misleading claims. In fact, manual assessment of such claims (as done in websites such as politifact.com, factcheck.org, and snopes.com) cannot scale with the proliferation of sources spreading false information [29]. Different efforts tackle different types of claims and domains. We focus on algorithms that test textual claims, such as "Leo Tolstoy won the Nobel Prize", against trustful Knowledge Bases (KBs). KBs store information as triples, where a *predicate* expresses a binary relation between a *subject* and an *object*. KB triples, or facts, encode real-world entities and their relationships. Examples of KBs come from the academia [2, 24, 27] as well as the industry [6, 8, 9].

We assume that entities and predicates involved in "worth checking" claims have been identified [15, 17], and study the step estimating the *veracity* of a given claim (expressed as structured data) w.r.t. trusted reference data. The core issue in fact checking with KBs is that the reference information is incomplete, i.e., some true entities and relationships are missing. In particular, information is usually very sparse for topics in the long tail. For this reason, KBs are used under the Open World Assumption (OWA), i.e., a fact not in the KB can either be false or just missing [8, 12]. Fact checking algorithms therefore state if fact f is a valid missing fact in KB R, a task that can be seen as a special case of link prediction in graphs [5].

Algorithms to address these problems come from different intuitions. Some of them rely on paths and sub-graphs in the KB: they assume that training examples are available and learn models to label new facts to be tested [13, 14, 23]. Other approaches assume that constraints over the KB have been defined and can be exploited to validate a given fact [18]. Others rely on embeddings to model a candidate predicate between two entities as a translation in the corresponding low dimensional vector space [3, 28].

Due to the richness and diversity of algorithms, it is important to conduct fair experimental evaluations to assess the potential of each proposal. Thorough evaluation of fact checking algorithms requires systematic control over the training and test data, and the quality of reference information. To support rigorous empirical evaluations, a fact generation system must be able to produce multiple scenarios with low user effort and clear evaluation results.

This paper aims to take a major step in this direction. We present a new benchmark for fact checking algorithms covering a wide range of scenarios. Since the algorithms come from different research fields and rely on different hypothesis over the claims and the KBs, we introduce a benchmark that is centered on the properties of the data. Our scenario generator is the first tool conceived to support empirical evaluations of fact checking algorithms as per the requirements outlined above. It takes as input a predicate for a KB (e.g., *capital* for DBpedia) and creates scenarios of increasing complexity in terms of training, test, and reference data. We then analyze a variety of algorithms to answer the following questions:

- How do fact checking systems fare in absolute terms with existing KBs?
- Are there other approaches that can perform the same task and how do they compare to fact checking solutions?
- What are the properties of the data that most affect the quality of the results?

To answer these questions, we identify and study the impact of different aspects of the training and test data.

Example 1.1. Consider the example claims in Table 1. Predicate *capital* is a one-to-one (functional) predicate, which contains facts for cities and states as triples *capital(city,state)*. Intuitively, it is easier to recognize as false the claim that Sacramento (capital city of California) is not the capital of Arizona than the fact that Worcester (largest city in Massachusetts) is not the capital of Massachusetts. Similarly, a false claim that Paris is the capital of Japan is easier

	Scenario	Example Claims
Functional	1	capital(Phoenix, Arizona)
		capital(Sacramento, Arizona)
	2	capital(Boston,Massachusetts)
		capital(Worcester, Massachusetts)
	3	capital(Boston,Massachusetts)
		capital(Osaka, Japan)
Non funct.	4	award(J. Kittinger, War Prisoner Medal)
		award(J. Kittinger, Bronze Star Medal)
		award(J. L. Morgan, Bronze Star Medal)
		award(L. Linke, 2009 RTHK)

Table 1: Scenario examples, false claims in italic.

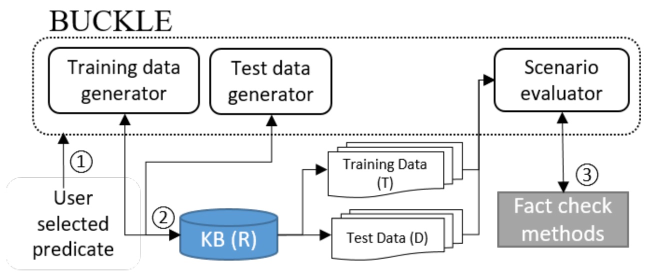


Figure 1: The benchmark architecture.

to verify than the claim that Osaka is the capital of Japan. Going beyond capitals, stating if someone is really the winner of an award is harder, in general, as many people can win the same award and the same person can win more than one.

The observations above are natural given our background knowledge as humans, but we need formal notions to capture the complexity of fact checking different claims w.r.t. a given KB. In this work, we define four properties of the facts to address this problem: *popularity*, *transparency*, *homogeneity*, and *functionality*. While we do not claim that this list is complete, every property radically affects the quality of the claim assessment. Such properties are therefore the fundamental building blocks for our generator of fact checking scenarios of different complexities.

We present next our testing infrastructure (Section 2). We describe the methods tested in our comparison (Section 3), including link prediction algorithms that are not specifically "branded" as fact checking methods. We introduce the data properties that model the complexity of our test scenarios and the algorithms to generate datasets that satisfy such properties (Section 4). We report the results of the experimental evaluation and the insights gained from it (Section 5). Finally, we discuss related work and open challenges that emerged from our study (Sections 6 and 7). Our infrastructure and code for scenario generation are available online (http://github.com/huynhvp/Benchmark_Fact_Checking).

2 THE BENCHMARK

We first give some background and fix the terminology. We then introduce the architecture of our benchmark platform.

Background. A *fact* is defined as a triple that has the form of ("subject" s, "predicate" p, "object" o). Natural language processing techniques are used to convert a textual claim into a structured format. Facts can be classified into categories, such as numerical, quote, and object property. We focus on object properties, which are facts stating a relationship between the subject and the object in a sentence, e.g., Sacramento is the capital of California.

A *Knowledge Base* (KB) is a direct graph where nodes correspond to entities (subject or object in a fact) and edges model binary predicates among entities. We focus on algorithms taking as input a KB and a fact that is not part of it. Such algorithms assess if the fact belongs to the missing part of the KB (therefore is "true") or no (is "false"). Most entities in a KB have a predicate defining their type (e.g., general as "thing", or specific as "person" or "company").

Fact checking scenario. Any benchmark has a set of standard application scenarios that can be tested against different systems sharing similar functionalities. This implies that every scenario is correctly interpreted by the systems. In our benchmark, a fact checking scenario is defined for a predicate of interest p with a triple (R, T, D), where R is the reference data (the KB), T is the training data (positive and negative p facts, sound w.r.t. R), and D is the test data: true and false p facts, or *claims*, missing from the KB.

Benchmark architecture. Our benchmark, namely BUCKLE, consists of three components. Two components create the data for the training (T) and test data (D) in a scenario, while the third combines and executes them over the fact checking systems, as depicted in Figure 1. A user interacts with BUCKLE by creating fact checking tasks for a predicate in a KB. For the input predicate, the system creates multiple scenarios, according to the configuration. Target algorithms are then executed and compared on such scenarios.

The role of data. Every data aspect plays a role in the creation of a fact checking scenario. Our goal is to support the automatic generation of scenarios of increasing complexity without assumptions on the target algorithm. This means that we introduce structural and semantic properties of R, T, and D, and propose algorithms to efficiently enforce such properties in the datasets.

3 FACT CHECKING ALGORITHMS

We classify different fact checking algorithms according to the methods they use to solve the problem. We include link prediction algorithms even if they were not designed for fact checking, as they can nonetheless perform the task. All algorithms assume that either training examples T (labelled facts) or reference information R (the KB) are available to build the models for checking claims in D. A common assumption is that the KB is trustable, and training examples are derived from it. In the following, we assume that the given test claims are missing from the KB.

Structure Based Algorithms. Given a claim $p(s, o)$, this group of algorithms makes a decision for it by exploiting the topological structures identified in the KB by the p triples. The KB triples are used to learn the alternative paths (different from p) between their subjects and objects. Properties of the paths are then modeled as features in a classifier that decides if predicate p holds for the given s and o. We consider four algorithms from this family.

Knowledge Linker (**KL**) builds an internal model based on a weighted adjacency matrix with edge weights computed as the in-degree of each node in the KB [5]. The model ignores the labels (semantics) of the predicates and evaluates the validity of an input fact based on the proximity between its subject and object. In our tests we use the *Metric closure* for computing the distance, as it performs best in practice [16]: every path connecting a given subject and object is mapped to a score computed on the frequency of the nodes in the KB. The more often the node occurs in KB, the less information it conveys.

Discriminate Predicate Path Mining (**KG-Miner**) exploits frequent anchored predicate paths between pair of entities in the KB [23]. Given a claim (*s, p, o*) and positive examples P that satisfy p in the KB, it collects the predicate paths for every node pair in P having subject with the same type of s (subject set $\varphi(s)$) and object with type of o (object set $\varphi(o)$). From each subject $u \in \varphi(s)$ and corresponding object $v \in \varphi(o)$, predicate paths that alternatively represent predicate p are extracted from the KB with a depth-first search (DFS) traversing the graph from u to v up to a length m. The information gain of paths and corresponding labels are computed based on their number of occurrences. It then selects the most discriminative paths and plugs them into training with positive and negative examples in a logistic regression model that optimizes the Area Under Receiver Operating Characteristic (AUROC). This model is then used to compute the likelihood of a claim.

Path Ranking Algorithm (**PRA**) is based on the same process of KG-Miner, but with important differences [14]. For extracting features from the (positive and negative) training set of triples, it uses a two-sided, unconstrained random walk starting at the source and corresponding target nodes to retrieve connected paths between them. Top k paths for each training instance are kept based on their number of occurrences and are collected into a feature matrix. A value in the matrix corresponding to a training instance (*s, p, o*) is the probability of arriving at the target node o by a walk starting at source node s and following a specific path among its top k paths. The feature matrix is then used with a classifier (trained on positive and negative examples) to validate the input claim.

Sub-graph Feature Extraction (**SFE**) extends PRA by extracting features from sub-graphs in the KB [13]. Given a parameter m, the *sub-graph of depth m* for each node n is the result of m breadth-first search steps starting at it. A sequence of predicates that connects a source node to target node is obtained by intersecting the sub-graphs of source and target. SFE uses such sequences to identify *binary features* that disregard the frequency (KG-Miner) or the probability (PRA) of feature paths. Features are then used in a classifier trained on positive and negative examples.

Embedding Based Algorithms. Embeddings encode entities and relationships in the KB into a low-dimensional vector space while preserving certain information of the graph and minimizing a margin-based ranking loss. A relationship in the graph is interpreted as a translation from subject entity to object entity in such space. To check a fact *p(s, o)*, these methods measure the relevance of the embedding representations **s** of s and **o** of o w.r.t. embedding representation **p** of predicate p though a specific score function *f(s, p, o)*. We consider two algorithms for this family.

Translating Embeddings (**TransE**) represents a predicate p from triple *p(s, o)* as a translation from subject s to object o on the same low-dimensional embedding space, that is $\mathbf{s} + \mathbf{p} \approx \mathbf{o}$ if (*s, p, o*) is true [3]. Its score function is defined as: $f(s,p,o) = \|\mathbf{s} + \mathbf{p} - \mathbf{o}\|$, where $\|.\|$ can be either L1 or L2 norm. TransE has drawbacks when dealing with non functional predicates. As it uses the same embedding space for both entities and predicates, a many-to-one predicate generates identical embedding representations for different subject entities in it. To address this issue, **TransH** enables an entity to have different embedding representations w.r.t. the different predicates it participates [28]. For each predicate p, it introduces a predicate-specific hyperplane w_p (normal vector) and defines an embedding vector **p** on this hyperplane.

There are other methods that can verify claims by exploiting constraints (logical rules) defined on the KB [1, 11, 18] Unfortunately, in general KBs do not come with a set of rules and it is not clear how to obtain a large number of high quality rules with existing mining methods [12, 21]. For this reason, we leave the evaluation of such rule-based algorithms to future work.

4 FACT SELECTION AND GENERATION

While the training T and test D datasets are produced separately (possibly with different input parameters), they are based on common properties of the data and can therefore be generated by one set of algorithms. We therefore blur the distinction between the training and test in the discussion of a unified dataset generator.

Each dataset includes true and false facts for a specific predicate, as in the examples in Table 1. We identify four independent properties for the datasets, and build scenarios that have different levels of difficulty based on how such properties are set and combined.

4.1 Popularity

Given any KB in our test, the true fact that Sacramento is the capital of California is easier to automatically check compared to the true fact that Kinshasa is the capital of Zaire. In fact, even if both facts are missing in the KB, the two entities in *capital(Sacramento,California)* are more "popular" in the reference data R than the African city and country. This can be naturally captured by distinguishing facts based on their amount of *context* information. In particular, we characterize the entities based on their structural properties. We define the *popularity* $\mathcal{G}(x)$ of an entity x as the number of incoming and outcoming edges for its node in the KB graph. For a given fact *p*(*s, o*), we compute the popularity of its entities (*s, o*) with the score:

$$\mathcal{G}(p,s,o) = min(\mathcal{G}(s), \mathcal{G}(o)) * (1 + \frac{max(\mathcal{G}(s), \mathcal{G}(o))}{\bar{\mathcal{G}}(p)}) \qquad (1)$$

where $\bar{\mathcal{G}}$(p) is the average popularity of all entities in the same predicate. The *min* operator avoids that a pair (*s, o*) get high popularity for one entity only. Higher popularity scores identify pairs with bigger context, which are easier to fact check.

For an example, consider facts f_1: *capital(Sacramento,California)*, f_2: *capital(Honolulu, Hawaii)*, and f_3: *capital(Kinshasa,Zaire)*. The popularity values in the KB DBpedia are 942 for Sacramento, 23298 for California, 1101 for Honolulu, 1144 for Hawaii, 205 for Kinshasa, and 328 for Zaire. With $\bar{\mathcal{G}}(capital)$ = 1452, these lead to popularity scores 16056, 1968, and 251 for f_1, f_2, f_3, respectively. More popular entities have less missing values in the KB and it is easier for the algorithms to create robust contexts for fact checking decisions.

4.2 Transparency

In fact checking, the false facts are the ones driving the evaluation of the performance of an algorithm. However, there are many ways to generate false facts. We first review existing methods for their selection and then motivate and introduce our approach.

Random generation. Consider a scenario with 50 US capitals as true facts taken from a trusted KB. From these 50 correct triples, we can generate 200 incorrect triples by random matching each capital to 4 other states for a total of 200 statements, as shown for the first scenario in Table 1. Since the *capital* predicate is functional, all the generated incorrect facts are truly false.

LCWA generation. To generate false facts that are truly incorrect also for non functional predicates, some ideas have been exploited in the literature. [8, 12, 21]. The *Local-Closed World Assumption* states that if a KB contains one or more object (resp. subject) values for a given subject (resp. object) and predicate, then it contains all possible occurrences for the two entities involved for that predicate [8, 12]. For example, if the KB states that "Homer" is *author* of "Odyssey", it implies that the KB contains all *author* facts for "Homer" and "Odyssey". This intuition suggests that for any person in the KB, any new pair combination for *author* with "Homer" as subject or "Odyssey" as object is a false fact.

Also, to ensure high accuracy for the generated negative facts, the types of the entities are required to be the same in a new fact. More precisely, given a function **T** returning the type, a true fact $p(x, y) \in R$, and a generated false fact $p(x, y') \notin R$, then it must be the case that $\mathbf{T}(y) = \mathbf{T}(y')$.

Finally, it has been noted that semantically connected negative examples lead to better rules in the context of rule mining [21]. As the KB also states that "Thomas Hobbes" has been a *translator* of "Odyssey", this is a valuable false fact for *author* in graph constraint mining. In fact, it can lead to a rule stating a contradiction between the birth year of the author and the release year of the book.

We denote with *LCWA generation* the approach based on these ideas. This false facts generation method correctly outputs facts that are false, but has some drawbacks that we discuss next.

Ambiguous false facts. Consider again the scenario with US capitals as true facts and false triples generated by random matching capitals and states. While the true facts are structurally connected in R (e.g., a capital always has more than one relationships with its state), the false facts have little in common in the KB. The facts are false, but they are not *ambiguous*, thus easy to identify as false by any algorithm. In other words, for any fact checking method, the true and the false facts have completely different internal representations that are easy to classify.

In Table 1, the second scenario contains false facts from another predicate between cities and states, *largestCity*. For each state in the correct capital-state triples, the scenario includes its largest city in the datasets as false facts. Since the large cities and the capital cities overlap and share properties in R, they are less "transparent" and harder to distinguish by the fact checking algorithms. For a given predicate p, this property is controlled by setting the percentage of false facts from pairs connected by *any* predicate different from p.

Issues with LCWA. It seems a valid idea to get meaningful false facts with the *LCWA generation*, where there always exists an alternative relationship between a subject and an object. Unfortunately, this approach introduces an important problem: many generated false facts resort to *very few direct predicates*. For example, given *nearestCity(Yosemite National Park, Mariposa California)* as a true fact, the LCWA-based generator extracts a false triple *location(Yosemite National Park, California)*. This is a false fact, but in practice this approach leads to predicate *location* accounting for ≈90% of the generated false facts for predicate *nearestCity*. A specific predicate appearing frequently among negative triples (such as *location*) leads to a strong evidence for the classification of false facts. For example, with methods that use logistic regression, predicate *location* is the feature with largest absolute weight.

While we need to avoid this drift, we want false facts coming from related predicates to create ambiguous, non transparent facts. To solve this problem, we give an algorithm for false fact generation.

Random walk generation. The first issue in the LCWA generation is considering only direct (1-hop) predicates between two entities for alternative paths, as this choice drastically reduces the number of alternatives. For a predicate *p*, a dataset should contain multiple false facts from distinct predicates that are semantically close to *p*. Given a true fact *p(s,o)*, we extend one-hop paths (direct predicates) to more general *predicate paths* between the given subject and object, i.e., $s \xrightarrow{p_1} \xrightarrow{p_2} \dots \xrightarrow{p_k} o$. Given the *nearestCity* example above, (Yosemite National Park, $location$, $isPartOf^{-1}$, Los Angeles California) is an ambiguous false fact based on a *nearestCity* triple and a 2-hop path.

To generate such triples, we introduce our *random walk generation of negative examples*, reported in Algorithm 1.

The algorithm starts with the generation of candidate facts $\mathcal{S}(x)$ (resp. $\mathcal{O}(y)$) by exploiting the LCWA assumption for subject x (resp. object y) in a true fact *p(x,y)*. For example, the candidate fact *location(Yosemite National Park, California)* for *nearestCity(Yosemite National Park, Mariposa California)*.

Then, the algorithm checks the types of the entities involved with a notion less strict than the equality. We say that type of y ($\mathbf{T}(\mathbf{y})$) is *consistent* to type of y' ($\mathbf{T}(\mathbf{y}')$) if they share path nodes in the type hierarchy of the KB. We define a function $\|\mathbf{T}(y)\|$ that returns the size of a type path, e.g., it returns 3 for type path "Thing/Place/ProtectedArea" obtained from y entity Yosemite National Park. This function allows us to test overlap of the path wrt a given threshold c. In the LCWA generation, this threshold coincides with the size of the type path, as it constraints all *subject* or *object* occurrences in a training or test dataset to have the same *type*. However, the higher the threshold c, the more difficult it is to find negative facts. To get more paths, we relax this condition with a value that is the minimum between a KB-depending threshold and the actual type path (for DBpedia, we use c=**4**).

If types are compatible, the algorithm extracts all possible paths with length smaller than a threshold t between the subject and object, e.g., $\mathbf{D}^3_{(Yosemite, California)}$ for *Yosemite National Park* and *California*.[1] For each path p in these possible paths, a random walk is initialized from the subject to each intermediate node that follows the predicates in the path. For example, for $p \in \mathbf{D}^3_{(Yosemite, California)}$

[1]We use t=3 in our experiments as it allows to efficiently compute rich paths. Triples for RDF/S predicates, such as Domain and Range, are ignored in the navigation.

```
input: $\mathcal{P}$ - reference data $kb$, true triples $\mathcal{P}$ for predicate
  $rel \in kb$, $n$ false triples for predicate $rel$
output: set of false triples $\mathcal{N}$
for $(x, rel, y)$ in $\mathcal{P}$ do
    $\mathcal{S}(x) = \{(x, rel', y') | (x, rel', y') \in kb \ \& \ (x, rel, y') \notin kb\}$;
    for $(x, rel', y')$ in $\mathcal{S}(x)$ do
        if $\|T(y) \cap T(y')\| \geq min(c, \|T(y)\|)$ then
            $D^t_{(x,y')}$ : predicate paths of maxlength $t$ from $x$ to $y'$
            for path in $D^t_{(x,y')}$ do
                o: node reached by a random walk starting
                 from $x$ and following path
                if $\|T(y) \cap T(o)\| \geq min(c, \|T(y)\|)$ then
                    if $(x, *, o) \notin \mathcal{N}$ & $(x, rel, o) \notin kb$ then
                        $\mathcal{N}$.append$(x, path, o)$
    $O(y) = \{(x', rel', y) | (x', rel', y) \in kb \ \& \ (x', rel, y) \notin kb\}$;
    for $(x', rel', y)$ in $O(y)$ do
        if $\|T(x) \cap T(x')\| \geq min(c, \|T(y)\|)$ then
            $D^t_{(y,x')}$ : predicate paths of maxlength $t$ from $y$ to $x'$
            for path in $D^t_{(y,x')}$ do
                s: node reached by a random walk starting
                 from $y$ and following path
                if $\|T(x) \cap T(s)\| \geq min(c, \|T(y)\|)$ then
                    if $(s, *, y) \notin \mathcal{N}$ & $(s, rel, y) \notin kb$ then
                        $\mathcal{N}$.append$(s, path, y)$
if $len(\mathcal{N}) < n$ then
    for $(x, rel, y)$ in $kb \backslash \mathcal{P}$ and $(x, rel, y) \sim \mathcal{P}$ do
        do same as above
        if $len(\mathcal{N}) == n$ then
            break
```

Algorithm 1: Random walk generation of negative facts.

with $p = location - isPartOf^{-1}$, a walk initialized at *Yosemite National Park* reaches *Catheys California*. If the identified triple satisfies the LCWA and the *type* consistency check, it is appended to the construction of the new false fact. In the example, triple *location(Yosemite National Park, Catheys California)* is kept as a false fact for the *nearestCity* predicate. Notice that the new false fact is semantically closer to the input true fact than the produced by LCWA-method as *Catheys* and *Mariposa* are at the same administrative level, while *California* is at a higher level.

We initialize every time only one random walk per path. This increases the likelihood of selecting diverse predicates in the alternative paths. Notice also that we can create multiple false facts for a single true fact. More precisely, we can add a new false fact for each path, which is useful if the number of require false triple n is larger than the true triples as input.

False facts generated by LCWA and random walk generators are reported in Table 2. In addition to facts created from 1-hop paths, our random walk generation produces ambiguous facts whose semantics is expressed by a path of predicates. Consider the false facts with *Will Smith*'s *spouses* in the second row. Since *Will Smith* has a child named *Jaden Smith* who associates with *Ta-Ku* in a music band, the random walk generation is able to conclude that (Will Smith, *spouse*, Ta-Ku) is a non transparent false fact. Also, starting from either the subject or the object, more false triples can be identified. The diverse semantics of false facts effectively increases the hardness of a scenario. Finally, as the random walk generator generalizes the LCWA approach, it is possible that some facts are produced by both, e.g., (Will Smith, *child*, Jaden Smith).

4.3 Homogeneity

We started with a scenario with *capital* triples for US states, as this is a setting studied in several fact checking papers. But, in such a well scoped scenario, all entities are semantically close to each other. That is, to check whether a US city in D is capital of a US state, the algorithms rely on the information from other US capitals and cities in T. In reality, data is heterogeneous, e.g., covering both US and European cities. To break homogeneity, the datasets should contain capitals of world countries, so that the facts model a more general concept of *capital*, as in the third scenario in Table 1. The fourth scenario of Table 1 is also not homogeneous, as people get awards in different domains (RTHK is a music award).

For a predicate p, this property can be controlled by clustering semantically the entities in the KB. We then determine the difficulty of the scenario based on the number of the selected clusters in the dataset: the smaller the number, the easier the scenario.

We employ bottom-up hierarchical clustering to partition pairs of entities (subject s, object o) of a given predicate p, denoted as $\{p(s,o)_i\}_{i=1..N}$, into multiple clusters in which pairs (s,o) belonging to same cluster have similar semantics. To calculate this likelihood, a measure of similarity between $\{(s,o)_i\}^p_{i=1..N}$ is required. For the sake of simplicity, we initially take the average of euclidean distances in the embedding space between subjects $d(s_i, s_j)$ and objects $d(o_i, o_j)$ [3]. However, if a predicate has its domain and range with same type, the subject of a triple can also be the object for another triple. For example, *child(Murray Deutch, Howard Deutch)* and *child(Howard Deutch, Zoey Deutch)*. To identify these cases (subject of a triple appearing as object of another triple), it is therefore necessary to take also a *cross-distance* $d(s_i, o_j)$ into account. Cross-distance helps to better discriminate these cases, thus leading to more accurate clusters. In addition, when dealing with predicates having different domains and range types, the cross-distance acts as a regulation term in the similarity calculation. Two pairs with large cross-distance will not belong to the same cluster. In summary, we use the following metric in our clustering:

$$d(i,j) = [d(s_i, s_j) + d(o_i, o_j) + d(s_i, o_j) + d(o_i, s_j)]/4 \qquad (2)$$

with $i, j = 1 \ldots N_p$.

With clustering methods, a challenging question is how to select a good number of clusters k that correctly represent the underlying structure. Intuitively, we can convert the problem of searching k into searching a threshold d (or distance). Here we use the Elbow method [25]. The number of clusters is selected by computing the gradient of the plot with the number of cluster vs. the threshold for d. The curve is expected to flatten out at the point giving maximum gradient of the plot, the "elbow", and we select the corresponding number of clusters as k. For example, in a plot for the *nearestCity* predicate, a plain plateau appears from the threshold d=1.2, and the corresponding number of clusters (k=115) is selected.

We report in Table 3 examples of homogeneous clusters obtained for four predicates. For *nearestCity*, our approach identifies one

Predicate	LCWA generator	Random walk generator
nearestCity	(Death Valley National Park, $location$, California)	(Death Valley National Park, $isPartOf^{-1}$, Indian Village, California)
		(Death Valley National Park, $location - isPartOf^{-1}$, Catheys, California)
	(Cleveland Forest, $location$, San Diego, California)	(RM National Park, $locatedInArea^{-1} - locatedInArea$, Larimer, Colorado)
		(Cleveland Forest, $sourceMountain^{-1}$, $region$, Orange, California)
spouse	(Will Smith, $child$, Jaden Smith)	(Will Smith, $child$, Jaden Smith)
		(Will Smith, $child - associatedMusicalArtist^{-1}$, Ta-Ku)
	(Julian Carroll, $successor$, John Brown Jr)	(Julian Carroll, $successor$, John Brown Jr)
		(John Brown Jr, $almaMater - almaMater^{-1}$, Karl Forester)
manufacturer	(SEAT 133, $assembly$, Fiat)	(Peugeot 306, $designer$, Pininfarina)
		(Alfa Romeo Giulietta, $assembly - foundationPlace^{-1}$, Fiat)
	(Ferrari 458, $designer$, Pininfarina)	(Ford Vedette, $relatedMeanOfTransportation - product^{-1}$, Simca)
		(GMC Typhoon, $manufacturer - division$, Chevrolet)
employer	(Yasheng Huang, $almaMater$, Harvard University)	(Yasheng Huang, $almaMater$, Harvard University)
		(Yasheng Huang, $employer - affiliation$, MIT)
	(Bob Jones III, $chancellor^{-1}$, BobJones University)	(Bob Jones III, $chancellor^{-1}$, BobJones University)
		(John Dickson Carr, $deathPlace - city^{-1}$, BobJones University)

Table 2: False facts generated by LCWA and random walk algorithms.

Predicate	Cluster
nearestCity	(Santa Barbara, California)
	(San Luis Obispo, California)
	(Los Angeles, California)
foundedBy	(Death Row Records, Dr. Dre)
	(Aftermath Entertainment, Dr. Dre)
	(Death Row Records, Dick Griffey)
manufacturer	(BMW 3 Series (F30), BMW)
	(Toyota Corolla (E120), Toyota)
	(Ford Sierra, Ford Motor Company)
employer	(Tim Berners-Lee, MIT)
	(William L.Langer, Harvard University)
	(William L.Langer, University of Chicago)

Table 3: Homogeneous clusters for four predicates based on embeddings and our measure of distance.

cluster containing locations related to USA (in the table); a second example contains Canada's national parks. For *manufacturer*, two identified domains are the car and the railways industries (car examples in the table). For predicates *foundedBy* and *employer*, music labels and scientists have been identified, respectively.

4.4 Functionality

The last property characterizes the **functionality** of a predicate. A functional or inverse functional predicate is easier to model than a one-to-many predicate, such as persons in the *author* predicate with one or more books, but each book has at most one author. The hardest case is with many-to-many predicates, such as *award*, i.e., persons who got a prize or a recognition. In the fourth scenario of Table 1, notice how the facts are not functional: the same person can win different awards and the same award can be given to multiple people. We profile predicates in the reference KBs to identify functional and not functional ones.

5 RESULTS

Our scenario generator exposes the four properties described above. In this section, we test how such properties affect the quality of results for several fact checking algorithms.

KB. Our algorithms work with any RDF KB, but for the sake of space we report the results for tests done on DBpedia, a KB with triples extracted from Wikipedia. From these triples, we construct a graph by assigning each unique entity to a graph node, and converting the triple into a directed edge with label "predicate" from the entity "subject" to entity "object". We obtain a directed graph with ≈4M nodes, ≈27M edges, and 671 predicates. We treat the KB as trusted (i.e., assumed correct) but incomplete (open world assumption).

Datasets. In our test cases for fact checking, we focus on predicates with different characteristics. We select functional and non functional predicates and identify three predicates that are *popular* (appear in more than 10K triples in the KB) and one that is representative of the long tail: manufacturer (non functional: many to many, 31514 triples), nearestCity (non functional: many to many, 15016 triples), foundedBy (mostly functional, 10444 triples), employer (non functional: many to many, 5337 triples). Tested predicates have entities as objects, because this kind is supported by most methods. Our techniques can generate training and test datasets also for properties involving literal values, e.g., *birthDate* or *salary*.

Each test case involves an unique training scenario (for KG-Miner and SFE) and multiple testing scenario with 300 examples and a positive/negative label ratio of 1.0. In every testing scenario, entities are picked randomly according to the values of the four properties; however, to allow a fair comparison between different experiments, we reuse as much as possible the true facts across multiple generations. By default, all triples are correct, i.e., true and false facts are labelled correctly. Triples in the testing datasets are removed from the KB in the experiments.

Metrics. For evaluation of the different algorithms, we use the Area Under the Receiver Operating Characteristic curve (AUROC) and execution times. We identify AUROC as the primary quality indicator because is a generic measure (independent of the threshold values) and its curve is effective for visualization.

Algorithms. For the sake of space and clarity, we report results for a subset of the algorithms discussed in Section 3. Algorithms PRA and TransH are not reported as their behaviour is similar to SFE and TransE, respectively, with the latter methods outperforming the former ones in all cases.

5.1 Popularity

We start analyzing how the popularity of a fact influences the output of the fact checking methods. In this experiment we fix transparency/homogeneity to 1/1 for the training data, and to 1/0.5 for the testing data. The training datasets are always created selecting triples at random, while for the testing datasets the selection strategy varies across top-50 popular pairs, bottom-50 pairs in popularity (non-popular), and pairs taken randomly.

Method	Predicate	Popular	Non-Popular	Random
KG-Miner	nearestCity	0.84	0.58	0.69
	foundedBy	0.80	0.63	0.81
	manufacturer	0.55	0.51	0.53
	employer	0.58	0.38	0.50
KL	nearestCity	0.87	0.66	0.76
	foundedBy	0.82	0.67	0.80
	manufacturer	0.90	0.85	0.92
	employer	0.69	0.43	0.66
SFE	nearestCity	0.72	0.60	0.68
	foundedBy	0.63	0.60	0.81
	manufacturer	0.54	0.53	0.55
	employer	0.66	0.50	0.62
TransE	nearestCity	0.49	0.40	0.43
	foundedBy	0.75	0.60	0.75
	manufacturer	0.72	0.47	0.60
	employer	0.62	0.46	0.48

Table 4: AUROC results. Training data T selected randomly (KG-Miner and SFE only), testing data D selected according to popularity.

As we can see from Table 4, all algorithms obtain worse results on testing sets with low popularity compared to popular and random ones. Specifically, *nearestCity* shows a clear trend: *Popular* triples are easier to fact check than the ones selected at *Random*, which are easier to check than the *Non-Popular* triples. This is due to a large difference in popularity values $\mathcal{G}$ among entities of this predicate, as we can see from the 3 examples f_1, f_2, f_3 discussed in Section 4.1.

We observe that for random triples KG-Miner and SFE do not obtain good results in the datasets. This is explained by the training data containing a mix of popular and non popular entities, but all involving few distinct paths. KL and (to a smaller extent) TransE are more robust to this issue. To better explain the results, we looked at some statistics for the datasets in this experiment, as reported in Table 5. The table reports for each predicate and each dataset the number of distinct paths identified over all triples, the average frequency of every path (e.g., a value of 2 means that all distinct paths appear twice over all triples), and the average popularity of each triple. We observe in Table 5 that for predicate *manufacturer* there is a large difference in the number of distinct paths and their average frequency between the training data and the test data. This is reflected by KG-Miner and SFE reporting significantly worse results in Table 4 for *manufacturer*.

Predicate	Dataset	# distinct paths	Avg. freq. of paths	Avg. popularity
nearestCity	Random Train	19	8.8	42
	Popular Test	91	7.6	41
	Random Test	42	10.2	38
	Non-Pop. Test	32	9.9	7
foundedBy	Random Train	72	18.2	120
	Popular Test	151	30.0	595
	Random Test	90	18.0	64
	Non-Pop. Test	58	7.3	10
manufact.	Random Train	**11**	**2.3**	42
	Popular Test	49	9.4	151
	Random Test	49	4.5	38
	Non-Pop. Test	51	4.3	16
employer	Random Train	31	4.4	35
	Popular Test	39	9.2	65
	Random Test	62	4.9	25
	Non-Pop. Test	50	4.9	7

Table 5: Structural properties for the datasets. Training datasets are used only with KG-Miner and SFE.

KL and TransE are also the ones showing biggest benefit with popular triples. This confirms that their algorithms are affected by the popularity of the entities involved, while the others rely more on the variety of the paths, e.g., KG-Miner and SFE work well for *foundedBy*, which has a large number of distinct paths. We also experimentally verified that if training is done with popular pairs only, the qualitative results improve for the Popular test dataset.

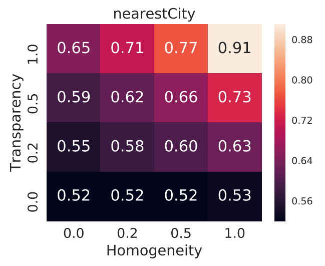
5.2 Transparency and Homogeneity

Transparency is used as a proportion of ambiguous false facts over non-ambiguous ones. Homogeneity is responsible for determining how many true and false facts are semantically close to each other, besides randomly-matched pairs. The two properties vary in the range [0..1].

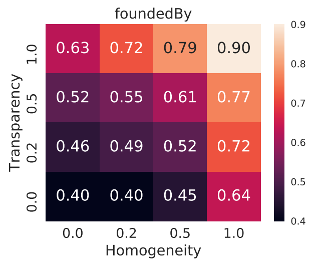
For transparency, value 1 is the case when all false facts are generated by random matching, while value 0 denotes a dataset with ambiguous, type-compatible instances only. For homogeneity, value 1 represents the case with all facts from the same cluster, while value 0 stands for a random selection that proportionally involves every clusters. The difficulty level of a scenario is directly proportional to transparency and homogeneity, and the dataset with high values for both is the easiest one.

We evaluate the impact of transparency and homogeneity properties by reporting in a two dimensional matrix the results from multiple experiments where we vary the values of the properties for the test datasets over values 0, 0.2, 0.5, and 1. In the training data we fix both transparency and homogeneity to 1.

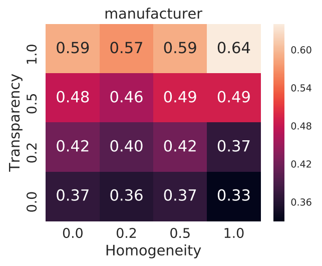
Qualitative results for KG-Miner and SFE are reported in Figure 2. As expected, the algorithms perform best on scenarios with high homogeneity and high transparency (top-right area of every matrix), and like a random classifier on scenarios with low value of two properties (bottom-left area). Interestingly, homogeneous *nearestCity* and *foundedBy* datasets of low transparency still give slightly good AUROC. Overall, the homogeneity property has strong impact



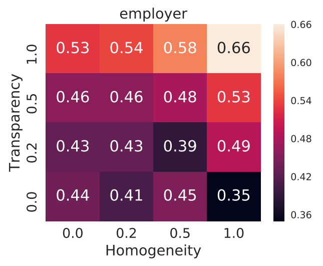
(a) nearestCity - KG-Miner



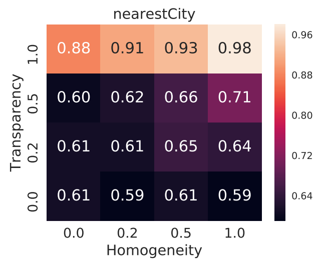
(b) foundedBy - KG-Miner



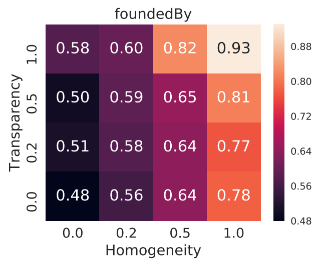
(c) manufacturer - KG-Miner



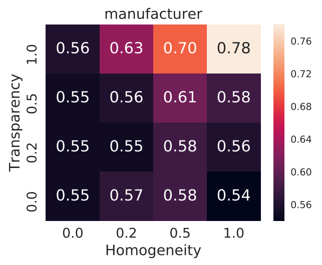
(d) employer - KG-Miner



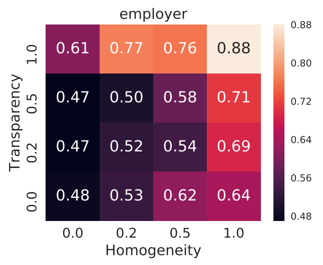
(e) nearestCity - SFE



(f) foundedBy - SFE



(g) manufacturer - SFE



(h) employer - SFE

Figure 2: AUROC matrices of KG-Miner and SFE in scenarios with different homogeneity and transparency.

and clear trend on the performance of unambiguous scenario (high transparency). For SFE, in most cases the performance is more proportional to transparency than homogeneity. These observations confirm our sense that these two properties play an important role on the behavior of path-based algorithms, such as KG-Miner and SFE. It may also be noted that SFE performs better than KG-Miner in most cases, particularly for homogeneous *nearestCity* and *employer*. After manually inspecting the paths and the sub-graphs, in these cases the two methods do not differ much. However, the difference in performance is explained by the different classifiers they use.

Results for TransE are shown in Figure 3. In general, scenarios with high transparency have good quality results. For example, in *foundedBy* with high homogeneity and transparency we observe results comparable to the other methods. Transparency also strongly affects the quality of TransE. The less transparent is at triple, the more difficult to fact check. Due to the embedding-based nature of TransE, the homogeneity has a more limited effect here. This is due to the nature of the clusters, which have been computed with embeddings, thus both entities in true and false facts are close to each other. Subject and object in false triples are semantically close to subject and object in the true facts, thus in some cases they have similar distances.

Figure 3 also shows that KL achieves very good results for scenarios with high transparency, regardless of the homogeneity level, and outperforms the other algorithms in most cases. KL directly relates its decision to the distance between subject and object in each triple in the test scenario. Hence, an unambiguous scenario polarizes these distances and is perfectly classified. Unsurprisingly, decreasing the transparency also degrades the performance of KL. For example, Worcester is not *capital* of Massachusetts, but it is one of its *largestCities*. Due to the lack of semantics, KL treats both Worcester and Boston as capital of Massachusetts because it finds good proximity between each city and the state in the graph. In terms of homogeneity, we observe a behavior similar to KG-Miner and SFE. With smaller and more homogeneous clusters (high homogeneity), it is easier for KL to identify better paths.

5.3 Functionality

Among the four predicates we tested in this evaluation, *foundedBy* is the simplest one as it is the most functional. In most cases, anything has been founded by only one person or company. As we can see from the results of all experiments on **Popularity**, **Homogeneity**, **Transparency**, all fact checking methods work better for the *foundedBy* scenario, rather than other predicates. The fact-checking task is clearly more robust for functional predicates than non-functional ones.

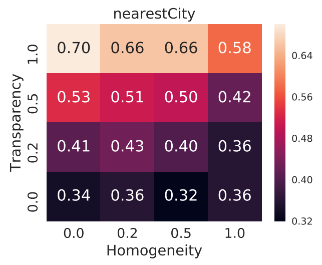
5.4 Execution times

We report in Table 6 the average execution times for every algorithm to fact check 300 triples in one scenario on a machine with a 8 core 1.6Ghz CPU and 30 GB RAM.

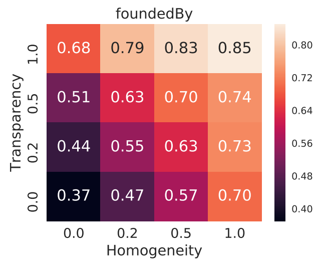
Algorithm	Time (seconds)
KL	1420
KG-Miner	30
SFE	20
TransE	5

Table 6: Average execution time (sec) for a scenario.

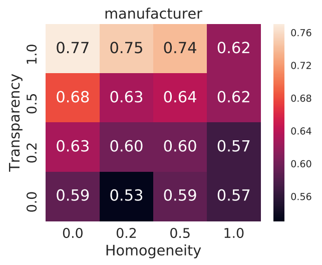
KL is computationally expensive, with more than 1400 seconds on average, and slower than path-based methods (KG-Miner, SFE),



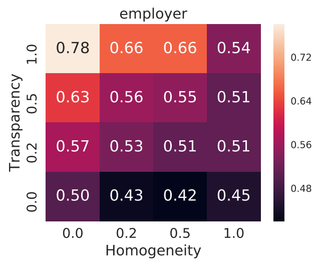
(a) nearestCity- TransE



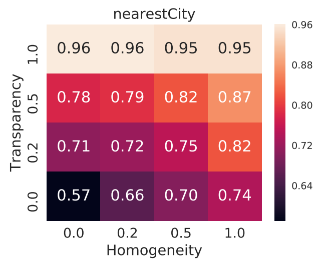
(b) foundedBy - TransE



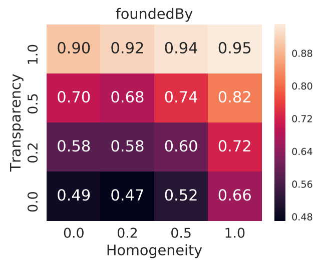
(c) manufacturer - TransE



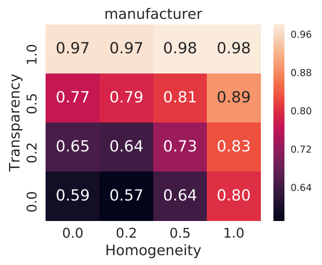
(d) employer - TransE



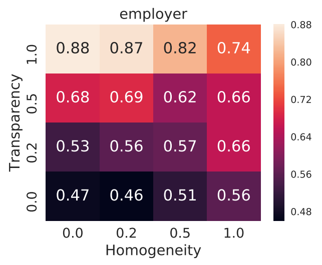
(e) nearestCity - KL



(f) foundedBy - KL



(g) manufacturer - KL



(h) employer - KL

Figure 3: AUROC matrices of TransE and KL in scenarios with different homogeneity and transparency.

which takes 20 to 30 seconds, and embedding-based algorithms (TransE), up to 5 seconds. However, for TransE, the computational cost is moved from the testing to the training step, where it takes more than 4 hours to learn the model. The training step for KG-Miner and SFE runs as quickly as their testing step and time to build an adjacency matrix in KL is much smaller (≈ 600s) than TransE's learning time. For our algorithms to enforce data properties (Section 4), execution times are always in seconds, assuming embeddings for the KBs have been pre-computed.

5.5 Take away messages

We draw some conclusions about the four data properties from the experimental evaluation.

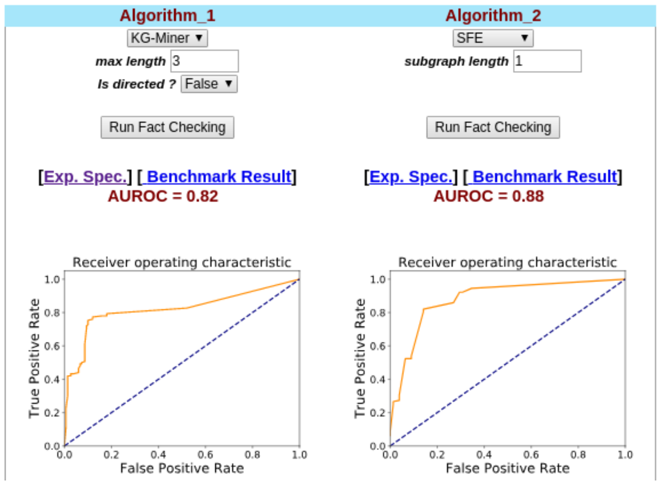
(1) More popular triples are easier to fact check for all algorithms (Table 4).

(2) More complex data stuctures do not necessarily lead to better results. In some cases (Table 4), a simple path (KG-Miner, KL) does better than a sub-graph (SFE). However, for more challenging setting, the richer information provided by sub-graphs leads indeed to better results (Figure 2).

(3) Whereas KG-Miner and SFE significantly depend on the structural properties of the entities in the KB, TransE is more stable and KL benefits from its independence from the predicates labels in the paths. On the other hand, labelled output paths and graphs can be used as an explanation for a fact checking decision.

(4) All methods perform better with functional predicates than non functional ones.

(5) Finally, KL and TransE do not need labelled training data, a crucial property in many real use cases.



Algorithm_1 | Algorithm_2

KG-Miner | SFE

max length 3 | subgraph length 1

Is directed ? False

[Exp. Spec.] [Benchmark Result] AUROC = 0.82

[Exp. Spec.] [Benchmark Result] AUROC = 0.88

Figure 4: Qualitative comparison for two algorithms executed on the same fact checking scenario.

The experiments show that our benchmark enables the effective comparison of different algorithms over the same fact checking scenario. We built a GUI to let users do the analysis with an interactive process. In the system, users select a KB, a predicate to test, and the desired values for the four data properties. Once a scenario has been generated, users select and tune the algorithms to be executed. For example, by selecting KL, the choice between *metric* and *ultra-metric* closure appears as an option. After the execution, qualitative results are visualized with the receiver operating characteristic curve, as shown in Figure 4. With the benchmarking tool, qualitative differences in the results of all algorithms become evident with datasets involving non functional predicates, more ambiguous datasets, and less popular entities. The tool also enables to replicate experimental settings from papers in the literature.

6 RELATED WORK

We focus on fact checking methods based on reference information in the forms of knowledge graphs. This is different from fact checking approaches that exploit documents on the Web [19, 26], databases of previously checked claims (https://fullfact.org), and relational datasets [4], but also from the study of trust and misinformation spread in social networks [10, 22].

Qualitative experimental evaluations of fact checking systems have already been conducted [5, 11, 13, 14, 18, 23]. We reused systems from these earlier studies, but our work differs in several ways. First, previous evaluations have involved a limited number of systems, typically one or two, while this is the first benchmark that supports qualitative comparison for a large number of different systems. Second, earlier studies have considered mostly simple scenarios that did not fully stress the functionalities of the tested systems. We generate scenarios of increasing complexity, ultimately highlighting that several cases are still far from satisfactory qualitative results. Finally, datasets and code for this work is available online. We believe this is crucial towards advancing reproducibility and the experimental culture of the community.

In the context of fact checking, there have been proposals for the comparison of verification tools on a *fixed* set of claims (e.g., https://herox.com/factcheck). Related efforts in creating generic benchmarks have focused on the collection of *suites* of scenarios [7, 20] (https://docs.openml.org/benchmark/) or on the execution time to train models of increasing size (https://mlperf.org/). In contrast, we are closer in spirit to benchmark approaches where scenarios of increasing complexity are *generated from existing data collections*, i.e., the KBs of interest. Other works have studied how sampling and class imbalance affect link prediction methods [30], but they do not study the properties we introduce in this work.

7 CONCLUSIONS

Our effort should not be mistaken as aiming primarily for a performance competition. The benchmark aims at covering a range of scenarios to answer the questions we posed in Section 1. Specifically, we can now observe a diverse set of systems and inputs that allow us to derive conclusions about the different fact checking algorithms. We showed that a single data property can have a bigger impact on the qualitative results than algorithmic choices. While all methods perform well in simple scenarios, qualitative differences in the results become evident with more complex datasets in terms of the properties we have introduced. Experimental results vary in most cases from around 0.5 to 0.95 AUROC for a fixed algorithm, fixed KB predicate, and fixed ratio of training/test data.

Based on the lessons that we have learned, one future direction clearly stands out. As different models shine in different data situations, we envision a system that can take the best from all of them by picking the right algorithm at runtime, based on our data properties. We plan to extend our work in this direction and identify more data properties to better assess fact checking algorithms.

REFERENCES

[1] Naser Ahmadi, Joohyung Lee, Paolo Papotti, and Mohammed Saeed. 2019. Explainable Fact Checking with Probabilistic Answer Set Programming. In *Truth and Trust Online, TTO.*

[2] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD.* 1247–1250.

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NIPS.*

[4] Tien Duc Cao, Ioana Manolescu, and Xavier Tannier. 2018. Searching for Truth in a Database of Statistics. In *WebDB.* 4:1–4:6.

[5] Giovanni Luca Ciampaglia, Prashant Shiralkar, Luis M Rocha, Johan Bollen, Filippo Menczer, and Alessandro Flammini. 2015. Computational fact checking from knowledge networks. *PloS one* 10, 6 (2015), e0128193.

[6] Omkar Deshpande, Digvijay S Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. 2013. Building, maintaining, and using knowledge bases: a report from the trenches. In *SIGMOD.*

[7] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. (2017). http://archive.ics.uci.edu/ml

[8] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *KDD.*

[9] Xin Luna Dong. 2018. Challenges and innovations in building a product knowledge graph: extended abstract. In *(GRADES-NDA).* 1:1.

[10] Emilio Ferrara, Onur Varol, Clayton A. Davis, Filippo Menczer, and Alessandro Flammini. 2016. The rise of social bots. *Commun. ACM* 59, 7 (2016), 96–104.

[11] Mohamed H. Gad-Elrab, Daria Stepanova, Jacopo Urbani, and Gerhard Weikum. 2019. ExFaKT: A Framework for Explaining Facts over Knowledge Graphs and Text. In *WSDM.* 87–95.

[12] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW.* ACM.

[13] Matt Gardner and Tom M Mitchell. 2015. Efficient and Expressive Knowledge Base Completion Using Subgraph Feature Extraction.. In *EMNLP.* 1488–1498.

[14] Matt Gardner, Partha Pratim Talukdar, Jayant Krishnamurthy, and Tom Mitchell. 2014. Incorporating vector space similarity in random walk inference over knowledge bases. (2014).

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

[16] Viet-Phi Huynh and Paolo Papotti. 2018. Towards a Benchmark for Fact Checking with Knowledge Bases. In *Companion of the The Web Conference.* 1595–1598.

[17] Israa Jaradat, Pepa Gencheva, Alberto Barrón-Cedeño, Lluís Màrquez, and Preslav Nakov. 2018. ClaimRank: Detecting Check-Worthy Claims in Arabic and English. In *NAACL-HTL.* 26–30.

[18] Julien Leblay. 2017. A Declarative Approach to Data-Driven Fact Checking. In *AAAI.* 147–153.

[19] Tsvetomila Mihaylova, Preslav Nakov, Lluís Màrquez, Alberto Barrón-Cedeño, Mitra Mohtarami, Georgi Karadzhov, and James R. Glass. 2018. Fact Checking in Community Forums. In *AAAI.* 5309–5316.

[20] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining* 10, 1 (2017), 36.

[21] Stefano Ortona, Vamsi Meduri, and Paolo Papotti. 2018. Robust Discovery of Positive and Negative Rules in Knowledge-Bases. In *ICDE.* 1168–1179.

[22] Wanita Sherchan, Surya Nepal, and Cecile Paris. 2013. A Survey of Trust in Social Networks. *ACM Comput. Surv.* 45, 4, Article 47 (2013), 33 pages.

[23] Baoxu Shi and Tim Weninger. 2016. Discriminative predicate path mining for fact checking in knowledge graphs. *Knowledge-Based Systems* 104 (2016), 123–133.

[24] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW.* 697–706.

[25] Catherine A Sugar and Gareth M James. 2003. Finding the number of clusters in a dataset: An information-theoretic approach. *J. Amer. Statist. Assoc.* 98, 463 (2003), 750–763.

[26] James Thorne and Andreas Vlachos. 2018. Automated Fact Checking: Task Formulations, Methods and Future Directions. In *COLING.* 3346–3359.

[27] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[28] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes.. In *AAAI.*

[29] Claire Wardle. 2019. Enlisting the Public to Build a Healthier Web Information Commons. In *TheWebConf (WWW).* 3.

[30] Yang Yang, Ryan N. Lichtenwalter, and Nitesh V. Chawla. 2015. Evaluating link prediction methods. *Knowl. Inf. Syst.* 45, 3 (2015), 751–782.