Randomized Generation of Adversary-Aware Fake Knowledge Graphs to Combat Intellectual Property Theft

Snow Kang,[1] **Cristian Molinaro,**[2] **Andrea Pugliese,**[2] **V.S. Subrahmanian**[1]
[1] Dartmouth College, USA
[2] University of Calabria, Italy
Snow.Kang.21@dartmouth.edu, cristian.molinaro@unical.it, andrea.pugliese@unical.it, vs@dartmouth.edu

Abstract

Knowledge Graphs (KGs) can be used to store information about software design, biomedical designs, and financial information—all domains where intellectual property and/or specialized knowledge must be kept confidential. Moreover, KGs can also be used to represent the content of technical documents. In order to deter theft of intellectual property via cyber-attacks, we consider the following problem: given a KG K_0 (e.g., representing a software or biomedical device design or the content of a technical document), can we automatically generate a set of KGs that are similar enough to K_0 (so they are hard to discern as synthetic) but sufficiently different (so as to be wrong)? If this is possible, then we will be one step closer to automatically generating fake KGs that an adversary has difficulty distinguishing from the original. We will also be closer to automatically generating documents corresponding to fake KGs so that an adversary who steals such documents has difficulty distinguishing the real from the fakes. We formally define this problem and prove that it is NP-hard. We show that obvious approaches to solving this problem do not satisfy a novel concept of "adversary-awareness" that we define. We provide a graph-theoretic characterization of the problem and leverage it to devise an "adversary-aware" algorithm. We validate the efficacy of our algorithm on 3 diverse real-world datasets, showing that it achieves high levels of deception.

Introduction

Theft of intellectual property (IP) is a growing problem. According to a 2017 report by the US Intellectual Property Commission, theft of US IP alone is estimated to range from 0.87 to 2.61 percent of annual US GDP: a huge loss exceeding that of the 1991 Gulf War.[1] While one may dispute the precise numbers claimed in various news and other reports, the fact is that most IP producing companies have grave concerns about IP theft.

Recent efforts (Chakraborty et al. 2019; Abdibayev et al. 2021) have proposed generating fake versions of technical documents by replacing concepts in an original document with different concepts (e.g., in a sentence such as "In the iron-based powder mixture for use, it was confirmed that component segregation was greatly reduced"[2], the word "iron" might be replaced by "zinc" which has similar malleability properties but is more reactive). However, these past proposals have several limitations: (i) they do not consider the knowledge encoded within a document, and (ii) they do not account for the adversary who might both know the algorithm for generating fakes and analyze relationships between the resulting set (original + fakes) and thus discern the original one.

In this paper, we consider an "original" knowledge graph K_0 that captures some intellectual property. For instance, KGs can be used to capture information about software designs and flows via UML (Berardi, Calvanese, and De Giacomo 2005), biomedical designs (Alshahrani et al. 2017; Alshahrani 2019), financial information (Pujara 2017), corporate structure (Atzeni et al. 2020), and much more. Such KGs may capture not only intellectual property, but also valuable proprietary or secret corporate information. In addition, there are numerous methods to represent the knowledge within a document as a knowledge graph (Ji et al. 2020; Hogan et al. 2020). Such knowledge graphs can be readily extracted using standard methods in NLP (Luan et al. 2018). In the case of KGs representing text, we assume assume that given a knowledge graph, we can generate text from it—either via strategic replacements as in (Chakraborty et al. 2019) or via a generative model as in (Koncel-Kedziorski et al. 2019).[3]

Thus, this paper focuses on the central problem of generating fake versions of a given KG K_0. We show how to automatically generate a set $\mathcal{K} = \{K_0, K_1, \ldots, K_n\}$ having n fake documents (plus the original) so as to make it very hard for an adversary to guess which document in $\mathcal{K}$ is correct. In particular, the fakes are constrained to be "similar enough" (according to a distance metric) to the original (to be believable), but sufficiently dissimilar to be likely wrong. We make the following contributions: (i) We formally define the Fake Knowledge Graph problem FAKEKG and show that solving it is NP-hard. (ii) We show that even solutions to FAKEKG can be "cracked" by an intelligent adversary. (iii) In order to

[1] https://www.nationalreview.com/2020/07/china-intellectual-property-theft-counting-costs-united-states/

[2] Japanese patent JP5112828B2, https://patents.google.com/patent/JP5112828B2/en?q=metallurgy&oq=metallurgy

[3] Due to space reasons, we do not show how to generate KGs from documents or how to generate fake versions of an original document from fake KGs.

avoid this, we formally define the notion of an "adversary-aware" algorithm: even if the adversary knows the algorithm we use and also knows all inputs except for the original KG, s/he does not gain knowledge from them to identify the original KG. (iv) We show that solving the FAKEKG problem is closely linked to a graph theory problem and propose an adversary-aware algorithm, CLIQUE-FAKEKG, for this problem. (v) We run experiments on 3 knowledge graph datasets showing that CLIQUE-FAKEKG achieves good results in deceiving adversaries.

Preliminaries

We assume two disjoint finite sets E of *entities* and R of *relations*. A *knowledge graph* (KG) is a subset K of $\mathsf{E} \times \mathsf{R} \times \mathsf{E}$. Thus, K is a finite set of triples of the form $\langle s, r, o \rangle$, whose meaning is that the "subject" entity s is related to the "object" entity o via relation r. An alternative way of looking at KGs is as directed graphs where vertices are entities and edges are labeled with relations.

In addition to KGs which are directed and labeled, we use undirected graphs in our graph-theoretic characterization of the problem and to develop our algorithm. An *undirected graph* G is a pair $\langle V, E \rangle$, where V is a finite set whose elements are called *vertices*, and $E \subseteq V \times V$ is a set of unordered pairs (of vertices) called *edges*. We use "graphs" to refer to undirected graphs.

A *clique* of G is a subset C of V such that for every pair of distinct vertices v and v' in C, $(v, v') \in E$. We say that C is a *maximum* clique of G iff there is no clique C' of G such that $|C| < |C'|$. A clique of G with cardinality k is also called a k-clique of G.

Given a set of vertices $V' \subseteq V$, the *subgraph of G induced by V'*, denoted $G[V']$, is the graph $\langle V', E' \rangle$ where $E' = \{(v', v'') \mid v', v'' \in V' \text{ and } (v', v'') \in E\}$.

For an arbitrary set S, we will use 2^S to denote the powerset of S, that is, the set of all subsets of S.

The Fake KG Problem

In this section, we formally define the problem of generating fake KGs, study its computational complexity, and introduce the notion of an *adversary-aware* algorithm.

Our goal is to generate fake yet highly believable KGs from an original one, so that after putting them all together, an adversary has no clue which is the original KG.

Before providing the formal definition of our problem, we introduce distance functions for KGs. Given a set $\mathcal{U}$ of KGs, a *distance function* d for $\mathcal{U}$ is a function $d : \mathcal{U} \times \mathcal{U} \to [0, 1]$ such that for all $K, K', K'' \in \mathcal{U}$, (i) $d(K, K') = 0$ iff $K = K'$, (ii) $d(K, K') = d(K', K)$, and (iii) $d(K, K') \leq d(K, K'') + d(K'', K')$. As a simple example, a distance function can be the *Jaccard distance*, that is, $d(K, K') = 1 - \frac{|K \cap K'|}{|K \cup K'|}$.

The problem we address, named FAKEKG, is defined as follows.

Definition 1 (FAKEKG problem)**.** Given a set $\mathcal{U}$ of KGs, a distance function d for $\mathcal{U}$, a knowledge graph $K_0 \in \mathcal{U}$, an integer $n \geq 1$, and an interval $\tau = [\ell, u] \subseteq [0, 1]$, find a set $\mathcal{K} = \{K_0, K_1, \ldots, K_n\} \subseteq \mathcal{U}$ of $n + 1$ distinct KGs s.t. $\ell \leq d(K_i, K_j) \leq u$ for every $0 \leq i \neq j \leq n$.

Thus, an instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of FAKEKG includes

- a set $\mathcal{U}$ of KGs, which are all the KGs of interest for the application at hand;
- a function d measuring the distance between KGs in $\mathcal{U}$;
- the original KG K_0 for which we want to generate fakes;
- the number n of fake KGs we want to generate;
- an interval τ that bounds the distance between any two distinct KGs in a solution.

Notice that $\mathcal{U}$ is an arbitrary set of KGs whose role is to specify the set of KGs from which fake ones are taken. In fact, in many cases it is desirable for users to express which KGs are of interest or deemed to be "admissible" for the application at hand. For instance, $\mathcal{U}$ might only include KGs that satisfy a given set of semantic constraints which capture whether the KG makes sense or not. It is also very important to note that d can be *any arbitrary* distance function, which can therefore incorporate structural and/or semantic aspects of KGs (e.g., the Jaccard distance only looks at the structure of a KG). We do not impose any restriction on d—our framework and algorithms work for any distance function one wants to employ. Thus, both structural and semantic aspects can be taken into account when defining the set of admissible KGs (namely, $\mathcal{U}$) and their distance (namely, d).

A solution for I is a set $\mathcal{K} = \{K_0, K_1, \ldots, K_n\}$ of $n + 1$ distinct KGs such that their pairwise distance lies in the interval τ. Notice that $\mathcal{K}$ includes the original knowledge graph K_0 along with n additional KGs. The requirement $\ell \leq d(K_i, K_j)$ allows users to set a minimum desired distance between every pair of distinct KGs in $\mathcal{K}$. This also means that fake KGs must be "far enough" from the original one, that is, their distance from K_0 must be at least ℓ. The requirement $d(K_i, K_j) \leq u$ allows users to set a maximum desired distance between every pair of distinct KGs in $\mathcal{K}$. This also means that fake KGs must differ from the original at most of u, so users can state that fake KGs must not be too far from the original one (e.g., to keep them "believable enough"). We point out that d can also be used to express strict requirements that should hold between any pair of KGs, and in particular between the original and fakes. So, for instance, d can be defined in such a way that if two KGs have the same structural property of interest then their distance is measured according to some criterion, while if they do not have the same structural property their distance is set to 1. Notice that an instance of FAKEKG might not admit a solution.

The following theorem states NP-hardness of FAKEKG. The proof relies on a reduction from the NP-complete CLIQUE problem, that is, given an undirected graph G and an integer k, decide whether G has a k-clique. Specifically, we reduce the CLIQUE problem to the problem of deciding whether an instance of FAKEKG admits a solution, which in turn implies NP-hardness of FAKEKG.

Theorem 2. *The* FAKEKG *problem is NP-hard.*

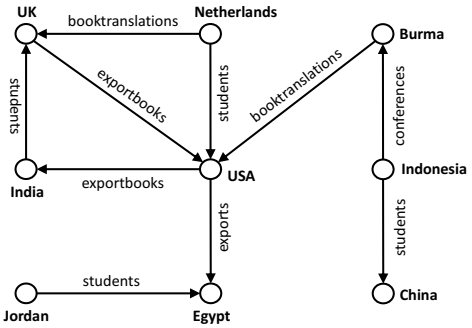


Figure 1: Knowledge graph of Example 3.

Our goal is to design algorithms to solve FAKEKG. Clearly, a good algorithm must make it difficult for an adversary to single out the original KG from the output of the algorithm. This comes with a caveat. As customary in information security (e.g., in cryptography), the algorithm itself is public, and this knowledge must not allow an adversary to make a better guess of which KG is the original one. Indeed, we place ourselves in a very hostile setting, assuming that an adversary knows the algorithm A as well as $\mathcal{U}$, d, n, and τ (and $\mathcal{K}$, of course). Thus, an adversary knows everything except which KG is the original one in $\mathcal{K}$.

In the following example, we report a (not so good) algorithm to solve FAKEKG and show why an algorithm for this purpose must be carefully designed.

Example 3. Consider the simple KG K_0 reported in Figure 1, which is an excerpt taken from the Nation dataset (Kim, Xie, and Ong 2016), one of the datasets we used in our experimental evaluation. Vertices represent countries, while a directed edge labeled R that goes from vertex A to vertex B represents country A having some relationship R with country B, with the meaning of edge labels being as follows:

- *conferences*: country A attends conferences in country B;
- *exportbooks*: country A exports books to country B;
- *booktranslations*: country A has translations of books from country B;
- *students*: country A has students from country B;
- *exports*: country A exports to country B.

Suppose $\mathcal{U}$ contains all the KGs that can be built using the entities (i.e., vertex labels) and relations (i.e., edge labels) appearing in K_0. Let the distance function for $\mathcal{U}$ be the Jaccard distance. Finally, suppose we want to generate $n = 2$ fake KGs for K_0 with $\tau = [0, 1]$. Consider now a simple algorithm A that generates fake KGs either by adding exactly one (randomly chosen) triple to the original KG or by deleting exactly one (randomly chosen) triple from the original KG (with the resulting KG being in $\mathcal{U}$). Suppose A is called with $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ as defined above, and A returns $\mathcal{K} = \{K_0, K_1, K_2\}$, where $K_1 = K_0 \setminus \{\langle USA, exports, Egypt \rangle\}$ (i.e., K_1 has been derived from K_0 by deleting $\langle USA, exports, Egypt \rangle$) and $K_2 = K_0 \cup \{\langle Jordan, students, India \rangle\}$ (i.e., K_2 has been derived from K_0 by adding $\langle Jordan, students, India \rangle$). Notice that $d(K_i, K_j) \in \tau$ for $0 \leq i \neq j \leq 2$. An adversary looking at $\mathcal{K}$, knowing algorithm A and its input other than the original KG, can nonetheless identify K_0 as the original KG with certainty. In fact, $A(\langle \mathcal{U}, d, K_1, n, \tau \rangle)$ can never output $\mathcal{K}$, as K_2 has two more triples than K_1, and thus K_2 cannot have been generated from K_1. Likewise, $A(\langle \mathcal{U}, d, K_2, n, \tau \rangle)$ can never output $\mathcal{K}$, as K_2 has two more triples than K_1, and thus K_1 cannot have been generated from K_2. Thus, it has to be the case that A was called giving K_0 as input, which discloses the actual original KG.

The previous example suggests that an algorithm solving FAKEKG must be carefully designed, in order to prevent an adversary from taking advantage of knowing the algorithm and some of the input elements to single out the original KG. We introduce a property that formalizes when an algorithm is "adversary-aware", that is, the knowledge of everything but K_0 does not provide any additional clue on which KG in $\mathcal{K}$ is the original one.

We consider randomized algorithms (deterministic algorithms are a special case of randomized ones). A randomized algorithm A might return different outputs when it is called multiple times with the same input. Thus, for a given input, rather than having a single output that is uniquely determined by the input, the output of A is a random variable. More formally, consider a randomized algorithm A to solve FAKEKG. For an instance I of FAKEKG (which is then an input of A), we use $A(I)$ to denote the random variable defined as the output of A when the input is I. Then, $A(I)$ takes values in $2^{\mathcal{U}}$ (i.e., each value is a set of KGs) and it is associated with a probability distribution $\mathsf{Pr} : 2^{\mathcal{U}} \to [0, 1]$ with $\sum_{\mathcal{K} \in 2^{\mathcal{U}}} \mathsf{Pr}[A(I) = \mathcal{K}] = 1$.

We now introduce the aforementioned notion of an *adversary-aware* algorithm, which is a conservative definition of security for an algorithm, as it assumes that an adversary knows the algorithm, $\mathcal{U}$, d, n, τ, and $\mathcal{K}$.

Property 1 (Adversary-aware algorithm)**.** *A (randomized) algorithm A to solve* FAKEKG *is* adversary-aware *iff it satisfies the following property: For every instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of* FAKEKG*, if $\mathcal{K} = \{K_0, K_1, \ldots, K_n\}$ is a possible output of $A(I)$, that is*

$$\mathsf{Pr}[A(I) = \mathcal{K}] > 0,$$

then

$$\mathsf{Pr}[A(\langle \mathcal{U}, d, K_i, n, \tau \rangle) = \mathcal{K}] = \mathsf{Pr}[A(\langle \mathcal{U}, d, K_j, n, \tau \rangle) = \mathcal{K}]$$

for every $0 \leq i \neq j \leq n$.

The rationale behind the previous property is as follows. Suppose we give as input to A an arbitrary instance $\langle \mathcal{U}, d, K_0, n, \tau \rangle$ of FAKEKG, and the returned output is $\mathcal{K} = \{K_0, K_1, \ldots, K_n\}$. Then, in order for A to be adversary-aware, it has to be the case that all the K_i's in $\mathcal{K}$ have the same probability of being the original KG. That is, for every $K_i \in \mathcal{K}$, the probability of $\mathcal{K}$ being the output when K_i is the input remains the same. If this is the case, then the knowledge of A, $\mathcal{U}$, d, n, τ, and $\mathcal{K}$ does not allow an adversary to make a better guess than picking one KG (uniformly at random) as the original KG.

Example 4. The algorithm A of Example 3 is not adversary-aware, since for the output $\mathcal{K} = \{K_0, K_1, K_2\}$ of Example 3 we have $\mathsf{Pr}[A(\mathcal{U}, d, K_0, n, \tau) = \mathcal{K}] > 0$, $\mathsf{Pr}[A(\mathcal{U}, d, K_1, n, \tau) = \mathcal{K}] = 0$, $\mathsf{Pr}[A(\mathcal{U}, d, K_2, n, \tau) = \mathcal{K}] = 0$. Consider an adversary looking at $\mathcal{K}$ and trying to figure out which KG was the original one. By knowing A as well as $\mathcal{U}$, d, n, and τ, the adversary can compute the probabilities above and figure out that K_0 is certainly the original KG (as K_1 and K_2 cannot yield $\mathcal{K}$ as result).

Notice that a deterministic algorithm A (i.e., A always returns the same output when it is called multiple times with the same input) can be seen as a particular randomized algorithm where, for every instance I of FAKEKG, $A(I)$ is such that $\mathsf{Pr}[A(I) = \mathcal{K}] = 1$ for some $\mathcal{K} \in 2^{\mathcal{U}}$. In such a case, we write $A(I) = \mathcal{K}$ to denote that $\mathcal{K}$ is the (only) output of A when it is called with input I. The following proposition provides an equivalent characterization of adversary-awareness for deterministic algorithms.

Proposition 5. *A deterministic algorithm A is* adversary-aware *iff it satisfies the following property: For every instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of* FAKEKG*, if*

$$A(I) = \mathcal{K} = \{K_0, K_1, \ldots, K_n\},$$

then

$$A(\langle \mathcal{U}, d, K_i, n, \tau \rangle) = \mathcal{K}$$

for every $1 \leq i \leq n$.

Intuitively, the previous proposition says that for an adversary-aware deterministic algorithm A, if $\mathcal{K}$ is the output of A for some arbitrary input $\langle \mathcal{U}, d, K_0, n, \tau \rangle$, then $\mathcal{K}$ must be the output of A also when the input is K_i (together with $\mathcal{U}$, d, n, and τ), for every $1 \leq i \leq n$. That is, all the K_i's yield $\mathcal{K}$ as output. This does not allow an adversary to rule out any of the K_i's as the candidate original KG.

Our goal in the next section is to develop an adversary-aware algorithm for the FAKEKG problem.

A Graph-theoretic Approach

In this section, we first provide a graph-theoretic characterization of the FAKEKG problem and then leverage it to design an adversary-aware algorithm.

Consider an instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of FAKEKG. Recall that a solution is a set $\{K_0, K_1, \ldots, K_n\} \subseteq \mathcal{U}$ of distinct KGs such that $d(K_i, K_j) \in \tau$ for every $0 \leq i \neq j \leq n$. We define the *distance graph* of I as the undirected graph $G_I = (V_I, E_I)$ such that $V_I = \mathcal{U}$ and $(K_i, K_j) \in E_I$ iff $d(K_i, K_j) \in \tau$. Then, we can show the following property, which allows us to look at solutions of I from a graph-theoretic perspective.

Proposition 6. *Consider an instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of* FAKEKG*. Then, $\mathcal{K} \in 2^{\mathcal{U}}$ is a solution for I iff $\mathcal{K}$ is an $(n+1)$-clique of G_I containing K_0.*

In light of Proposition 6, one natural approach to solve FAKEKG is to look for an $(n+1)$-clique containing K_0 in the distance graph. Algorithm 1, called NAIVECLIQUE-FAKEKG, is a simple randomized algorithm leveraging this property. In particular, it chooses an $(n+1)$-clique containing K_0 from the set of all $(n+1)$-cliques containing K_0. Attempting to be adversary-aware, Algorithm 1 picks the clique uniformly at random. However, as shown below, NAIVECLIQUE-FAKEKG is not adversary-aware.

Algorithm 1 NAIVECLIQUE-FAKEKG

Input: An instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of FAKEKG.
Output: A solution for I or $\emptyset$.
1: Let $\mathcal{C}$ be the set of all $(n+1)$-cliques of G_I containing K_0.
2: **if** $\mathcal{C} \neq \emptyset$ **then**
3: Pick an element C from $\mathcal{C}$ uniformly at random.
4: **return** C.
5: **else**
6: **return** $\emptyset$.

Fact 7. NAIVECLIQUE-FAKEKG *is not adversary-aware.*

To prove Fact 7, it suffices to show an instance of the FAKEKG problem for which NAIVECLIQUE-FAKEKG does not satisfy the condition of Property 1, which we report in the following example.

Example 8. Consider an instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of FAKEKG, where $\mathcal{U} = \{K_a, K_b, K_c, K_d, K_e, K_f, K_g\}$, $K_0 = K_a$, $n = 2$, and d and τ are such that the distance graph G_I in Figure 2 is derived. Suppose we call NAIVECLIQUE-FAKEKG with input I. The possible solutions for I are the 3-cliques of G_I including K_a, namely $C_1^a = \{K_a, K_b, K_c\}$, $C_2^a = \{K_a, K_b, K_d\}$, and $C_3^a = \{K_a, K_c, K_d\}$. Consider now $\mathcal{K} = C_2^a = \{K_a, K_b, K_d\}$, which is a possible output of NAIVECLIQUE-FAKEKG. We show that $\mathcal{K}$ does not satisfy the condition stated in Property 1. That is, we show that $\mathsf{Pr}[A(\langle \mathcal{U}, d, K_i, n, \tau \rangle) = \mathcal{K}]$ is not always the same for $K_i \in \{K_a, K_b, K_d\}$, where A is NAIVECLIQUE-FAKEKG. This means that, to single out the original KG, an adversary can make a better guess than picking one KG in $\mathcal{K}$ uniformly at random. $\mathcal{K}$ is one out of three possible cliques containing K_a (see C_1^a, C_2^a, and C_3^a above), so we have $\mathsf{Pr}[A(\langle \mathcal{U}, d, K_a, n, \tau \rangle) = \mathcal{K}] = 1/3$. Let us consider now $\mathsf{Pr}[A(\langle \mathcal{U}, d, K_b, n, \tau \rangle) = \mathcal{K}]$. The 3-cliques of G_I including K_b are $C_1^b = \{K_b, K_a, K_c\}$, $C_2^b = \{K_b, K_a, K_d\}$, and $C_3^b = \{K_b, K_c, K_d\}$. Thus, supposing that the original KG was K_b, the probability that A would have been returned $\mathcal{K}$ is $\mathsf{Pr}[A(\langle \mathcal{U}, d, K_b, n, \tau \rangle) = \mathcal{K}] = 1/3$. Finally, let us consider $\mathsf{Pr}[A(\langle \mathcal{U}, d, K_d, n, \tau \rangle) = \mathcal{K}]$. The 3-cliques of G_I including K_d are $C_1^d = \{K_d, K_a, K_b\}$, $C_2^d = \{K_d, K_a, K_c\}$, $C_3^d = \{K_d, K_b, K_c\}$, and $C_4^d = \{K_d, K_e, K_f\}$. Thus, supposing that the original KG was K_d, the probability that A would have been returned $\mathcal{K}$ is $\mathsf{Pr}[A(\langle \mathcal{U}, d, K_d, n, \tau \rangle) = \mathcal{K}] = 1/4$. As a consequence,

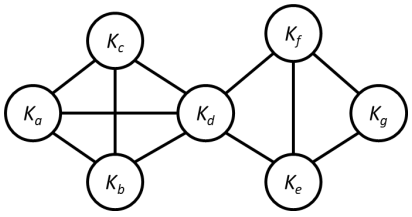


Figure 2: Distance graph for the scenario of Example 8.

NAIVECLIQUE-FAKEKG is not adversary-aware.

In the preceding example, we note that an adversary who knows the NAIVECLIQUE-FAKEKG algorithm, has the output $\mathcal{K} = \{K_a, K_b, K_c\}$, as well as the input parameters $\mathcal{U}$, d, n, and τ (but not which KG in $\mathcal{K}$ is the original one) can try to identify the original KG by computing the above probabilities. S/he thus gains important information on K_a and K_b being more likely than K_d to be the original KG (which was indeed K_a).

Before presenting an adversary-aware algorithm, we point out one issue which makes NAIVECLIQUE-FAKEKG non-adversary-aware. As shown in Example 8, for the output $\{K_a, K_b, K_d\}$, K_a (resp., K_b) belongs to three 3-cliques, which are all possible solutions for K_a (resp., K_b), while K_d belongs to four 3-cliques, which are all possible solutions for K_d. This makes the probabilities of K_a, K_b, and K_d of being the original KG different.

To overcome this issue, we devise a new (adversary-aware) algorithm whose main ideas are as follows. Pairwise disjoint cliques of the distance graphs are computed, with each clique being of size at least $n + 1$—indeed, as discussed in the following, our algorithm is optimized so as to iteratively compute cliques on subgraphs of decreasing size, and it performs pruning that avoids the computation of all cliques. For each vertex (i.e., KG) K of a clique C, if K is the original KG for which we want to generate fakes, the possible solutions are all subsets $\mathcal{K} \subseteq C$ including K with $|\mathcal{K}| = n + 1$. One of such solutions is picked uniformly at random. This behavior ensures that all K_i's in $\mathcal{K}$ have the same number of solutions, which in turn makes the algorithm adversary-aware, as we will show later.

Let us clarify how cliques are computed. In order for a clique to accommodate as many KGs as possible, maximum cliques are computed. More specifically, the algorithm starts by finding a maximum clique C of the distance graph with $|C| \geq n + 1$. If K_0 belongs to C, one solution is picked uniformly at random from the set of all subsets $\mathcal{K} = \{K_0, K_1, \ldots, K_n\}$ of C and it is returned. If K_0 does not belong to C, then the (smaller) subgraph $G[V \setminus C]$ is considered (i.e., the subgraph of the distance graph induced by $V \setminus C$) and the same process described above is applied.

We now go into the details of our algorithm, called CLIQUE-FAKEKG (cf. Algorithm 3), which in turn relies on algorithm CLIQUECOMPUTATION (cf. Algorithm 2).

We start by explaining CLIQUECOMPUTATION, which takes as input an instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of the FAKEKG problem, and it gives as output a set of cliques of the distance graph of I, with each clique having size at least $n + 1$. On line 1, the distance graph G_I is defined. Then, line 2 defines a graph G, which is set equal to G_I. Graph G is the graph where a maximum clique is sought: initially, it is the entire distance graph G_I; then, it will be a smaller induced subgraph (indeed, it gets smaller and smaller as the algorithm proceeds). On line 3, the set $\mathcal{C}$ of all cliques that have been found so far is initialized to the empty set. The **while** loop on lines 4–10 looks for (maximum) cliques of size at least $n + 1$ containing K_0, and it stops when such a clique has been found or there is no clique of size at least $n+1$. On line 5, a maximum clique C of G is found (notice that $|C| \geq n + 1$ because of the **while** condition). We assume an arbitrary but fixed total order over sets of vertices, so when there are multiple maximum cliques, one is deterministically chosen according to such order (this is needed to ensure the algorithm is adversary-aware). On line 6, C is added to $\mathcal{C}$. If C contains K_0, then $\mathcal{C}$ is returned (lines 7–8). Otherwise, C is deleted from G (lines 9–10), and the process discussed so far is repeated. Eventually, the algorithm returns $\mathcal{C}$.

CLIQUE-FAKEKG takes as input an instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of the FAKEKG problem, and it gives as output a solution for I or the empty set (meaning that no solution has been found). First of all, a set $\mathcal{C}$ of cliques is computed by algorithm CLIQUECOMPUTATION (line 1). If $\mathcal{C}$ includes a clique C containing K_0 (line 2), which is guaranteed to have size at least $n+1$ by CLIQUECOMPUTATION, then, among all subsets of C containing n elements different from K_0, one is picked uniformly at random, denoted $\mathcal{S}$ (line 3). Then, K_0 is added to $\mathcal{S}$, and the resulting set is returned (line 4). Otherwise, the empty set is returned (line 6).

As mentioned before, the algorithm offers several computational benefits. First, the set $\mathcal{C}$ of cliques can be stored and be used for subsequent calls of CLIQUE-FAKEKG, as long as $\mathcal{U}$, d, and τ remain the same. The latter parameters are indeed less likely to be modified for a fixed application. If the original KG belongs to a clique in $\mathcal{C}$, no additional computation is needed. If the original KG does not belong to any clique in $\mathcal{C}$, the computation can be resumed starting from the stored $\mathcal{C}$. Notice that this approach can be used even with different values of n.

Another computational benefit of the algorithm is that it performs the computation of maximum cliques of graphs of decreasing size at each iteration, making the computation less demanding as the algorithm proceeds. Indeed, different scalable algorithms for computing maximum cliques have been developed and can be employed off-the-shelf, e.g. (Cheng et al. 2012).

Algorithmic Complexity. Overall, the worst-case time complexity of CLIQUE-FAKEKG is given by the worst-case time complexities of building the distance graph and extracting cliques. The former is $O(T \cdot |\mathcal{U}|^2)$, where T is the worst-case time complexity to evaluate d. As for the latter, extracting cliques is a well-known problem, for which several algorithms have been developed, and the time complexity depends on the algorithm that is used, e.g., see (Cheng et al. 2012).

We illustrate CLIQUE-FAKEKG, as well as CLIQUECOMPUTATION, in the following example.

Example 9. Consider again the FAKEKG instance I of Example 8, whose distance graph G_I is reported in Figure 2. Recall that $K_0 = K_a$ and $n = 2$. First of all, CLIQUECOMPUTATION is invoked and looks for a maximum clique in G_I with size at least 3. The clique is $C = \{K_a, K_b, K_c, K_d\}$. Then, C is added to $\mathcal{C}$. Since C contains K_a, CLIQUECOMPUTATION returns $\mathcal{C}$ to CLIQUE-FAKEKG. Since $\mathcal{C}$ includes clique C, which in turn contains K_a, among all subsets of C containing two KGs different from K_a, one is picked uniformly at random, say it $\mathcal{S}$. Thus, $\mathcal{S}$ might be one of $\{K_b, K_c\}$, $\{K_b, K_d\}$, and $\{K_c, K_d\}$. Finally, $\mathcal{S} \cup \{K_a\}$ is

Algorithm 2 CLIQUECOMPUTATION

Input: An instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of FAKEKG.
Output: A set of k-cliques of G_I with $k \geq n + 1$.

Let $G_I = (V_I, E_I)$.
$G = (V, E) = (V_I, E_I)$.
$\mathcal{C} = \emptyset$.
while G has a clique of size at least $n + 1$ **do**
  Let C be a maximum clique of G.
  Add C to $\mathcal{C}$.
  if C contains K_0 **then**
    return $\mathcal{C}$.
  $G' = G[V \setminus C]$.
  $G = G'$.
return $\mathcal{C}$.

Algorithm 3 CLIQUE-FAKEKG

Input: An instance $I = \langle \mathcal{U}, d, K_0, n, \tau \rangle$ of FAKEKG.
Output: A solution for I or $\emptyset$.

$\mathcal{C} =$ CLIQUECOMPUTATION(I).
if $\mathcal{C}$ includes a clique C containing K_0 **then**
  Pick a set $\mathcal{S}$ of n elements from $C \setminus \{K_0\}$ uniformly at random.
  return $\mathcal{S} \cup \{K_0\}$.
else
  return $\emptyset$.

returned as a solution.

We can show that CLIQUE-FAKEKG is an adversary-aware algorithm.

Theorem 10. CLIQUE-FAKEKG *is adversary-aware.*

We conclude by providing some insights on why CLIQUE-FAKEKG is adversary-aware, using the scenario discussed in Example 9. Suppose the set $\mathcal{S}$ chosen by CLIQUE-FAKEKG is $\{K_b, K_c\}$, and thus the returned solution is $\mathcal{K} = \{K_a, K_b, K_c\}$. Consider an adversary looking at $\mathcal{K}$, trying to figure out which is the original KG. Suppose that the adversary knows all elements of I but K_0, and knows how CLIQUE-FAKEKG works. Then, the probability that CLIQUE-FAKEKG returns $\mathcal{K}$ when the original KG is K_a is 1/3. In fact, recall that $\{K_b, K_c\}$ is chosen uniformly at random out of the three sets $\{K_b, K_c\}$, $\{K_b, K_d\}$, and $\{K_c, K_d\}$ (cf. Example 9). If the original KG were K_b, then $\{K_a, K_c\}$ would have been chosen uniformly at random out of the three sets $\{K_a, K_c\}$, $\{K_a, K_d\}$, and $\{K_c, K_d\}$, and thus the probability that CLIQUE-FAKEKG returns $\mathcal{K}$ when the original KG is K_b is again 1/3. An analogous argument can be applied to K_c, yielding the same probability. Hence, K_a, K_b, and K_c are equally likely to be the original KG, and from knowledge of CLIQUE-FAKEKG and its input the adversary has learnt nothing!

Experimental Evaluation

In this section we describe the experiments we performed in order to assess how CLIQUE-FAKEKG can prevent discovery of the original KGs. We implemented the algorithm in Python on a 2.3 GHz Dual-Core Intel Core i5 with 8GB of LPDDR3 RAM, running MacOS Catalina Version 10.15.6. For duplication purposes, the code, sample KGs, and sample outputs generated may be downloaded from https://dsaildartmouth.github.io/FakeKG.pdf.

We used three datasets: *Nation*[4] (Kim, Xie, and Ong 2016), which represents international relations among countries, *UMLS*[5] (Kim, Xie, and Ong 2016), which represents biomedical relations, and the Microsoft *FB15K-237* (*FB* for short)[6] (Toutanova et al. 2015), which stores triples and textual mentions of Freebase entity pairs.

For each of the 3 datasets we extracted 22 original KGs and, for each KG, we computed 9 fake KGs—in particular, we computed 3 fake KGs for each of the following ranges of τ: $[0, 1/3]$, $[1/3, 2/3]$, and $[2/3, 1]$. Thus, we built a set T of 66 tests in total, each consisting of 10 KGs including the original one. The set $\mathcal{U}$ was derived as follows: first, we randomly picked a subgraph of the original dataset; then, we built new KGs by randomly adding and deleting vertices/edges/labels to the KGs built so far. The average number of vertices and edges of the original KGs was 15.79 and 9.98, respectively (16.18 and 9.95 for the fake KGs)—we decided to keep the KGs reasonably sized to allow human subjects to accurately analyze them. The size of $\mathcal{U}$ was 50. We used the Jaccard distance function.

Evaluation Process

We invited 10 human subjects to review the 66 tests in T, in order to gather the results of human evaluation over 660 tests overall. We asked the subjects to select, for each test, the top-3 KGs they felt were the original one. We developed a web-based tool to visualize the KGs as directed graphs with (labeled) vertices and edges, and made it available to evaluators. For each test, the KGs (1 original and 9 fakes) were displayed in a random order. A clear description of the domain and of the labels was provided. All subjects have a Master or a Ph.D. degree in Computer Engineering.

We reiterate an important point: while a structural distance function (namely, the Jaccard distance) was used to generate fake KGs, the evaluation also considered semantic aspects of KGs, as evaluators had to visually look at KGs labeled with semantic information. This put our framework in a tougher setting (for our system) by allowing human evaluators to make a more educated choice based on the semantics of the KGs—in contrast to more blind evaluations, such as a random choice (whose probability of success can be arbitrarily lowered by simply augmenting the number of fakes). For instance, in the Nation dataset, Indonesia taking military actions against the US is an unlikely event that can help evaluators to easily identify a KG to be fake if the KG claims that Indonesia carries out military actions against the US .

For each dataset, in order to assess the competence of the subjects on the corresponding topic, we generated 5 questions each of which presented two vertices and asked which labels (out of 5 possible ones) made sense if used on an edge

[4]https://github.com/dongwookim-ml/kg-data/tree/master/nation

[5]https://github.com/dongwookim-ml/kg-data/tree/master/umls

[6]https://www.microsoft.com/en-us/download/details.aspx?id=52312

between the two vertices. Each question was generated by (i) randomly picking a triple $\langle s, r, o \rangle$ from the dataset, (ii) making s and o the basis of the question, (iii) presenting r as a possible answer, together with 4 other randomly picked labels from the dataset, and (iv) considering as right all of the chosen labels r' for which there was actually a triple of the form $\langle s, r', o \rangle$ in the dataset. Each question counted for 5 possible points—a point was given for choosing a correct label or for not choosing an incorrect one. This way, we were able to associate a competence score (then normalized in the $[0, 1]$ interval) with each (subject,dataset) pair. The overall average score was 0.72 (specifically, 0.61 on the Nation dataset, 0.74 on the UMLS dataset, and 0.81 on the FB dataset).

In order to evaluate how our proposed approach can deceive human subjects, we defined a metric called *Deception Rate* (DR). For each subject h, each original KG K_0, and $r \in \{1, 2, 3, \text{top-3}\}$, we write $w(h, K_0, r) = 1$ if a fake KG was selected as the r-th choice by h, and $w(h, K_0, r) = 0$ otherwise—in the top-3 case, we assumed *the human subject was correct when any of his top-3 choices was right*. Then, for each r-th choice, we computed the average value of w of each subject (resp., each KG) over all KGs (resp., all subjects). Thus, we defined

$$DR(r, h) = \sum_{K_0} w(h, K_0, r)/|T|$$

and

$$DR(r, K_0) = \sum_{h} w(h, K_0, r)/10.$$

We computed the deception rates $DR(r, h)$ and $DR(r, K_0)$ for all possible choices. Then, we computed the average of $DR(r, h)$ across the human subjects (i.e., the overall average detection rate for the different choices) and the standard deviation of $DR(r, h)$ (resp. $DR(r, K_0)$) across the human subjects (resp. across T). Finally, for each dataset and for each of 6 different ranges of $DR(r, K_0)$, we computed the number of tests whose deception rate falls in the range.

Results

The results are reported in Figure 3, Table 1, and Figure 4.

They show a strong deception ability. In 86.8% of the cases (Figure 3) the KG that users selected as their top choice was in fact fake. Even in the top-3 case, our approach was able to deceive users in 62.7% of the cases. Moreover, the standard deviation across the human subjects (Table 1) was lower than that across the tests, which suggests that our approach achieves similar performance in achieving deception on different subjects. Finally (Figure 4), the deception

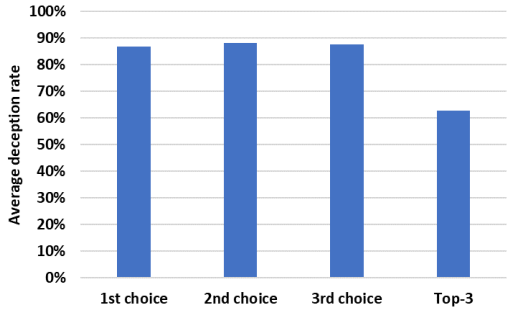
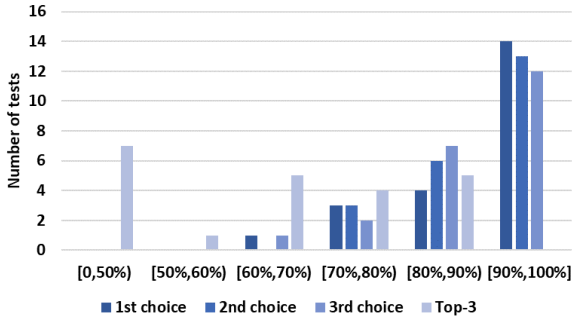


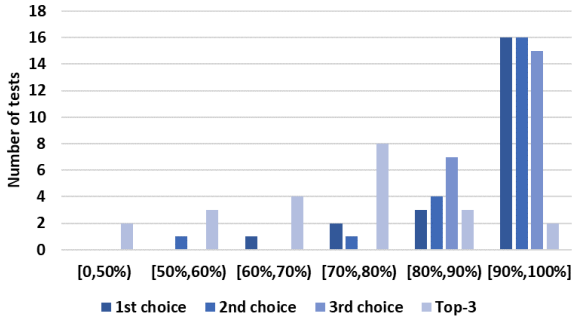
Figure 3: Average deception rate for the different choices.

	St.dev. $DR(r, h)$	St.dev. $DR(r, K_0)$
1st choice	0.06	0.11
2nd choice	0.04	0.09
3rd choice	0.04	0.10
Top-3 choice	0.11	0.16

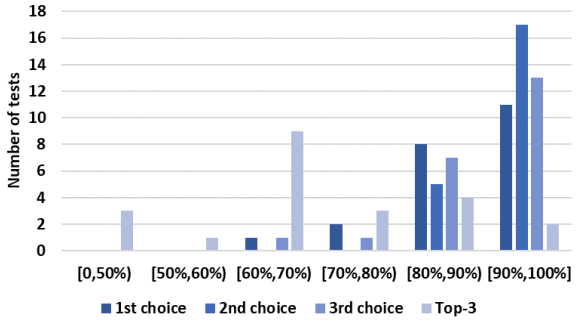
Table 1: Standard deviation of $DR(r, h)$ (resp. $DR(r, K_0)$) across the human subjects (resp. across T).



Nation dataset



UMLS dataset



FB dataset

Figure 4: Number of tests for different ranges of $DR(r, K_0)$.

rate for the first choice was above 90% on 41 out of 66 tests (14 out of 22 for Nation, 16 for UMLS, and 11 for FB) and above 80% on 56 out of 66 tests—for each of the datasets, in just 1 out of 22 test the deception rate was lower than 70%. For the second and third choices, the results were similar. Even in the top-3 case, the deception rate was above 60% in 49 out of 66 tests.

We also looked at the distribution of the KGs selected as top choices by the evaluators. Interestingly, if we consider the three distance ranges chosen for the fake KGs, the KG wrongly picked as top choice was in the range $[0, 1/3]$ in 243 cases, in the range $[1/3, 2/3]$ in 186 cases, and in the range $[2/3, 1]$ in 144 cases (overall, 573 out of 660 incorrect top choices were made). If we instead consider the probability of correctly identifying the original KG, its value was in the range $[0, 0.1]$ in 41 tests, in the range $[0.2, 0.3]$ in 22 tests, and 0.4 in 3 tests—overall, the probability was obviously $1 - \frac{\sum_h DR(1,h)}{10} = 13.2\%$.

Thus, the results suggest that our approach is successful at deceiving users.

The average time to generate one test for the Nation (resp. UMLS, FB) dataset was 10.57 (resp. 12.44, 50.55) seconds.

Related Work

Cyber-deception has been extensively used in different contexts during the past several years (Shabtai, Elovici, and Rokach 2012). For example, in the digital music industry, fake recordings containing terrible music have been generated in order to dissuade users from downloading unauthorized songs (Kushner 2003).

Deception has also been studied for files/documents (Yuill et al. 2004; Voris, Boggs, and Stolfo 2012; Chakraborty et al. 2019), software code (Park and Stolfo 2012), and at the systems level (Jajodia et al. 2016, 2017; Wang et al. 2012). A honeypot scheme has been proposed in (Yuill et al. 2004), which distributed decoy honey files throughout the system so that alerts are triggered as soon as an attacker breaks into the system and accesses a honey file. (Voris, Boggs, and Stolfo 2012) developed an automated system to translate the text into another language, with untranslatable but enticing nouns (such as company names, hot topics, and bogus login information) sprinkled throughout the text, increasing the probability that the attacker will try to steal the file. (Park and Stolfo 2012) works on deception at the code level and generates fake but believable Java code with techniques like obfuscation. A multi-layer deception system that provides in-depth defense against sophisticated attacks is designed by (Wang et al. 2012). (Chakraborty et al. 2019) propose the use of fake document generation as a way of mitigating intellectual property theft. Their FORGE system can deal with text only and uses meta-centrality metrics to identify key features to be replaced in the text of the document.

Meanwhile, there are also works on deception detection for the purpose of detecting fraud on document, code or text (Afroz, Brennan, and Greenstadt 2012; Caliskan-Islam et al. 2015). For instance, (Afroz, Brennan, and Greenstadt 2012) exploits linguistic features of written documents to detect stylistic deception and distinguish deceptive documents from original ones. (Caliskan-Islam et al. 2015) investigate machine learning methods to de-anonymize source code authors of C/C++ using coding style.

Different security problems have been investigated in the context of graph-structured knowledge, such as the compression and encryption of RDF datasets (Fernández et al. 2020) and search over encrypted graph data (Poh, Mohamad, and Z'aba 2012). Methods using some principled ways to introduce noise into knowledge graphs, such as typed sampling, relational sampling, and corrupting positive instances, have been proposed in the literature—e.g., see (Kotnis and Nastase 2017). As a direction for future work, it would be interesting to investigate how such approaches can be integrated into our framework.

Conclusions

In this paper, we consider the problem of generating a set $\mathcal{K} = \{K_0, K_1, \ldots, K_n\}$ of KGs from a given original KG K_0 so that the adversary has to invest time and effort in separating the real KG from the fake ones. Our formulation of this problem takes as input, not only the original KG, but several parameters. Our algorithm assumes a strong adversary—one who knows all the parameters we use as well as the algorithm we use. The only thing s/he does not know is the identity of K_0. Yet, we want to prevent the adversary from identifying K_0. We first formulate the FAKEKG problem and show it is NP-hard, but that it does not necessarily fool the adversary. We then develop our CLIQUE-FAKEKG algorithm and show that it satisfies the adversary-aware requirement. We run experiments on 3 very different datasets and show that CLIQUE-FAKEKG is very effective at deceiving adversaries.

We reiterate that the work in this paper applies to many domains that use knowledge graphs and/or similar constructs such as software design (Berardi, Calvanese, and De Giacomo 2005), biomedical applications (Alshahrani et al. 2017; Alshahrani 2019), finance (Pujara 2017) and business (Atzeni et al. 2020). However, this paper is not a work in isolation. It is intended to be used as part of 3 steps used to generate fake versions of a technical document: (i) Given an original document, we extract a knowledge graph from it using off-the-shelf methods such as (Luan et al. 2018). (ii) We then use the techniques in this paper to generate a set $\mathcal{K}$ of fake KGs. (iii) For each of the fake KGs in $\mathcal{K}$, we then generate a fake document using techniques similar to those in (Koncel-Kedziorski et al. 2019).[7] *This paper focuses on (ii).*

Acknowledgements

This work was partly funded by ONR grant N00014-18-1-2670.

[7] Readers might wonder how legitimate users may separate the real document from the fakes. As a solution based on Message Authentication Codes is already provided in (Chakraborty et al. 2019), we do not address this issue in this paper.

References

Abdibayev, A.; Chen, D.; Chen, H.; Poluru, D.; and Subrahmanian, V. 2021. Using Word Embeddings to Deter Intellectual Property Theft Through Automated Generation of Fake Documents. *ACM Transactions on Management Information Systems* .

Afroz, S.; Brennan, M.; and Greenstadt, R. 2012. Detecting Hoaxes, Frauds, and Deception in Writing Style Online. In *Proceedings of IEEE Symposium on Security and Privacy (SP)*, 461–475.

Alshahrani, M. 2019. *Knowledge Graph Representation Learning: Approaches and Applications in Biomedicine*. Ph.D. thesis, King Abdullah University of Science and Technology.

Alshahrani, M.; Khan, M. A.; Maddouri, O.; Kinjo, A. R.; Queralt-Rosinach, N.; and Hoehndorf, R. 2017. Neuro-symbolic representation learning on biological knowledge graphs. *Bioinformatics* 33(17): 2723–2730.

Atzeni, P.; Bellomarini, L.; Iezzi, M.; Sallinger, E.; and Vlad, A. 2020. Weaving Enterprise Knowledge Graphs: The Case of Company Ownership Graphs. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 555–566.

Berardi, D.; Calvanese, D.; and De Giacomo, G. 2005. Reasoning on UML class diagrams. *Artificial Intelligence* 168(1-2): 70–118.

Caliskan-Islam, A.; Harang, R.; Liu, A.; Narayanan, A.; Voss, C.; Yamaguchi, F.; and Greenstadt, R. 2015. De-anonymizing programmers via code stylometry. In *Proceedings of the USENIX Security Symposium*, 255–270.

Chakraborty, T.; Jajodia, S.; Katz, J.; Picariello, A.; Sperli, G.; and Subrahmanian, V. 2019. FORGE: A Fake Online Repository Generation Engine for Cyber Deception. *IEEE Transactions on Dependable and Secure Computing* .

Cheng, J.; Zhu, L.; Ke, Y.; and Chu, S. 2012. Fast algorithms for maximal clique enumeration with limited memory. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 1240–1248.

Fernández, J. D.; Kirrane, S.; Polleres, A.; and Steyskal, S. 2020. HDTcrypt: Compression and encryption of RDF datasets. *Semantic Web* 11(2): 337–359.

Hogan, A.; Blomqvist, E.; Cochez, M.; d'Amato, C.; de Melo, G.; Gutierrez, C.; Gayo, J. E. L.; Kirrane, S.; Neumaier, S.; Polleres, A.; Navigli, R.; Ngomo, A. N.; Rashid, S. M.; Rula, A.; Schmelzeisen, L.; Sequeda, J. F.; Staab, S.; and Zimmermann, A. 2020. Knowledge Graphs. *CoRR* abs/2003.02320.

Jajodia, S.; Park, N.; Pierazzi, F.; Pugliese, A.; Serra, E.; Simari, G. I.; and Subrahmanian, V. 2017. A probabilistic logic of cyber deception. *IEEE Transactions on Information Forensics and Security* 12(11): 2532–2544.

Jajodia, S.; Subrahmanian, V.; Swarup, V.; and Wang, C. 2016. *Cyber deception*. Springer.

Ji, S.; Pan, S.; Cambria, E.; Marttinen, P.; and Yu, P. S. 2020. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. *CoRR* abs/2002.00388.

Kim, D.; Xie, L.; and Ong, C. S. 2016. Probabilistic Knowledge Graph Construction: Compositional and Incremental Approaches. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2257–2262.

Koncel-Kedziorski, R.; Bekal, D.; Luan, Y.; Lapata, M.; and Hajishirzi, H. 2019. Text Generation from Knowledge Graphs with Graph Transformers. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2284–2293.

Kotnis, B.; and Nastase, V. 2017. Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs. *CoRR* abs/1708.06816.

Kushner, D. 2003. Digital decoys [fake MP3 song files to deter music pirating]. *IEEE Spectrum* 40(5): 27.

Luan, Y.; He, L.; Ostendorf, M.; and Hajishirzi, H. 2018. Multi-Task Identification of Entities, Relations, and Coreference for Scientific Knowledge Graph Construction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 3219–3232.

Park, Y. H.; and Stolfo, S. J. 2012. Software decoys for insider threat. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 93–94.

Poh, G. S.; Mohamad, M. S.; and Z'aba, M. R. 2012. Structured Encryption for Conceptual Graphs. In *Proceedings of the International Workshop on Security (IWSEC)*, 105–122.

Pujara, J. 2017. Extracting Knowledge Graphs from Financial Filings. In *Proceedings of the International Workshop on Data Science for Macro–Modeling with Financial and Economic Datasets (DSMM)*, 1–2.

Shabtai, A.; Elovici, Y.; and Rokach, L. 2012. *A Survey of Data Leakage Detection and Prevention Solutions*. Springer Briefs in Computer Science. Springer.

Toutanova, K.; Chen, D.; Pantel, P.; Poon, H.; Choudhury, P.; and Gamon, M. 2015. Representing Text for Joint Embedding of Text and Knowledge Bases. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)*, 1499–1509.

Voris, J.; Boggs, N.; and Stolfo, S. J. 2012. Lost in Translation: Improving Decoy Documents via Automated Translation. In *Proceedings of IEEE Symposium on Security and Privacy (SP) Workshops*, 129–133.

Wang, W.; Bickford, J.; Murynets, I.; Subbaraman, R.; Forte, A. G.; and Singaraju, G. 2012. Catching the Wily Hacker: A multilayer deception system. In *Proceeding of the IEEE Sarnoff Symposium*, 1–6.

Yuill, J.; Zappe, M.; Denning, D.; and Feer, F. 2004. Honeyfiles: deceptive files for intrusion detection. In *Proceedings of the IEEE SMC Information Assurance Workshop*, 116–122.