Generating Fake Documents using Probabilistic Logic Graphs

Qian Han, Cristian Molinaro, Antonio Picariello, Giancarlo Sperlì, V.S.Subrahmanian, Yanhai Xiong

Abstract—Past research has shown that over 8 months may elapse between the time when a network is compromised and the time the attack is discovered. During this long gap, attackers can steal valuable intellectual property from the victim. The recent FORGE system [8] has suggested that automatically generating fake—but believable—versions of documents can delay the attacker, cost him money, and increase his uncertainty. However, in order to generate fakes, FORGE only modifies the textual component of the document in question. But in the real world, documents consist of many non-textual components such as charts, equations, formulas, diagrams, and tables. We propose the concept of a Probabilistic Logic Graph (PLG) and show that PLGs provide a single, unified framework within which the different parts of a document can be expressed. We then define the problem of generating, for a given PLG representation of a document, a set of fake yet highly believable PLGs (i.e., documents), so that an attacker looking at them (both the original and the fake ones) cannot easily identify the original document. We show that the problem of generating fake PLGs is intractable—but we propose an approximation algorithm that solves it efficiently. We evaluate the use of PLGs over a corpus of patents and show that our fakes can effectively deceive an adversary.

Index Terms—Deception, cybersecurity, fake documents, intellectual property

1 Introduction

THeft of intellectual property (or IP) from US defense contractors, pharma companies, and others is now rampant—allegations of IP theft are now made almost daily in the press [15], [19]. The situation is further exacerbated by the fact that months might pass by before a successful compromise of an enterprise network is discovered. For example, researchers from Symantec [6] showed that on average, there is a gap of 312 days before a zero-day attack is discovered. During this time, the attacker has a relatively free hand to steal IP.

In order to address this problem, [8] developed the Fake Online Repository Generation Engine (or FORGE) system. The basic idea behind FORGE is simple: automatically generate k fake versions of every real document. When an attacker compromises an enterprises and steals their IP, one of two outcomes can occur. On the one hand, he might perform actions (e.g., build an engine) using one of the stolen documents as his guide—if this occurs randomly, his chance of using the right document is $\frac{1}{k+1}$, which is small. On the other hand, if he knows that the network has fake documents in it, he will need to make some effort to identify the real document. This takes time (which will be valuable for the victim), imposes costs on the attacker, and leaves his technical people frustrated and uncertain about whether they found the right design. Simply put, the attacker is faced with a needle in the haystack problem. FORGE was applied only to technical documents—we continue focusing on technical documents in this paper as well.

Technical articles are challenging for several reasons. They involve technical sub-languages and jargon, and contain elements such as equations, graphs, charts, figures, and tables that are not directly amenable to traditional NLP. FORGE only considers the textual part of a document and generates fakes by replacing some concepts c in the document by related concepts c'—but only in the textual part. If a concept c appears both in the text and in a diagram or a table in the document, then FORGE would replace c with some new concept c' in the text, but not in the diagram or table, leading to an inconsistency that an adversary can easily exploit when trying to separate the real document from the fakes.

Deceptive fake versions of a document should be created by modifying important technical components of the original document. In order to achieve this, three important problems need to be solved. *(i)* We need to develop a formalism to represent the many different kinds of technical content (e.g., text, tables, diagrams, equations, formulas, etc.) appearing in documents. *(ii)* We then need to generate fake yet believable versions of technical content captured via the representation of the document in the aforementioned formalism. After this, we need to be able to map back this representation into fake documents. *(iii)* We need to ensure that the fakes are not easily detectable by an adversary. We assume an adversary model consisting of two parts: (a) the hackers who penetrate the enterprise and steal documents, and (b) the domain experts who look at the documents, real and fake, and assess which one is real. In this paper, we consider a mix of both. We consider a hacker who is sufficiently familiar with network analysis that he can ana-

• *Address communication to: V.S. Subrahmanian. Q. Han, V.S. Subrahmanian, and Y. Xiong are with the Department of Computer Science, Dartmouth College, Hanover, NH 03755, USA. E-mail: {Qian.Han.GR,vs,Yanhai.Xiong}@dartmouth.edu*
• *C. Molinaro is with the DIMES Department, University of Calabria, Via P. Bucci 42C, 87036 Rende (CS), Italy. E-mail: cmolinaro@dimes.unical.it*
• *A. Picariello and G. Sperlì are with are with Dipartimento di Informatica e Sistemistica, Università di Napoli "Federico II", Via Claudio 21, 80125 Napoli, Italy. E-mail: {giancarlo.sperli, picus}@unina.it*

Manuscript received

lyze a graph showing the relationship between the original document and the fakes. We must ensure that the resulting graph prevents him from figuring out which the original document is. A big part of our focus is on (b) because the hackers are unlikely to be experts in the problem domain—for example, there have been many recent news articles about theft of COVID related vaccine data from Western companies, e.g. Spain[1] and the USA[2]. It is unlikely that hackers will be experts in medicine, just as it is unlikely that they will be experts in design of missiles or batteries or a new pharmaceutical. We then try to generate fakes that are "close enough" to the original to be considered plausible by domain experts, but sufficiently "far away" to likely be technically incorrect. The main goal of this paper is to address the aforementioned problems, which we do as discussed below.

Contributions. We make the following contributions.

- We introduce formal syntax and semantics of (Probabilistic) Logic Graphs, a simple yet powerful formalism to model a wide variety of content present in documents (e.g., charts, equations, formulas, tables, and many others), along with uncertainty. Probabilistic Logic Graphs (PLGs) allow us to represent and process documents containing different kinds of information in a uniform way.
- After introducing PLGs, we tackle the problem of generating fake yet believable PLGs (i.e., documents modeled via PLGs). We formally define the *Fake PLG Generation Problem* in such a way that the Adversary Model is taken into account. We then study its computational complexity. Specifically, we show that it is NP-hard.
- In light of the intractability result, we develop a greedy algorithm to approximately solve the problem and analyze its complexity.
- We experimentally evaluate the effectiveness of our algorithm in deceiving experts by automatically generating fake versions of computer science patents and presenting them to a panel of computer scientists. Our results show that our framework achieves high levels of deception.

The basic use of PLGs for generating fake documents is shown in Figure 1. An original document d_0 is first represented as a PLG pg_0 using the PLG paradigm introduced in this paper. After this, the PLG pg_0 is transformed into a set of fake PLGs $pg_1, \ldots, pg_n$ which are sufficiently "consistent" (we will define this concept formally in the paper) with the original to be believable, but sufficiently inconsistent that they are likely wrong. Moreover, they are required to satisfy additional properties that make it difficult for an adversary to identify the original PLG pg_0. Each fake PLG pg_i can then be used to generate a fake document d_i, leading to a set of fakes that can be introduced into the enterprise network[3].

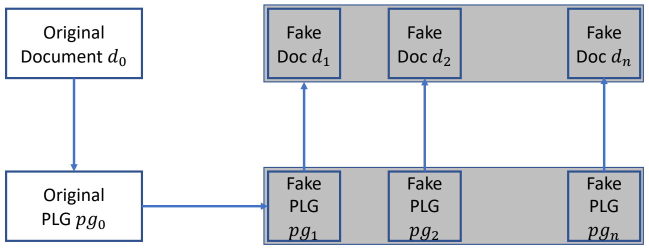


Fig. 1: Overall use of PLGs for Fake Document Generation.

Organization. The paper is organized as follows. We introduce (Probabilistic) Logic Graphs in Section 2. We define the Fake PLG Generation Problem and study its computational complexity in Section 3. We then propose a greedy algorithm to approximately solve the problem in Section 4 and also assess its complexity. We illustrate our framework with a real-world example in Section 5. An experimental evaluation of our approach is reported in Section 6. Related work is discussed in Section 7. We draw conclusions and outline directions for future work in Section 8.

2 (Probabilistic) Logic Graphs

In this section, we first introduce *logic graphs* (LGs) and then introduce *probabilistic logic graphs* (PLGs).

Logic graphs are directed graphs enriched with semantic information annotating vertices and edges. They are a simple but expressive formalism to naturally model different kinds of content encountered in documents, such as diagrams (e.g., flowcharts), expressions (e.g., algebraic or Boolean expressions), equations, tables, and many others. LGs provide a single unifying formalism to represent and process all these different kinds of document content. PLGs are the uncertain counterpart of LGs and thus allow us to model uncertainty when defining LGs—uncertainty can naturally arise in many ways. For instance, if Optimal Character Recognition (OCR) is used to process documents, then there is uncertainty in the automated extraction of information from documents. Algorithmic error is also a possible source of uncertainty—for instance, learning models such as HMMs may be used to predict the next character or word in text, but such models may not work perfectly when predicting the next character in a mathematical equation or chemical formula. Finally, we note that if machine learning models are used for text extraction, then they may naturally return a probability. We will say more about the sources of uncertainty shortly.

Syntax. We assume that within any given area of interest (e.g., computer science), we are given a set P of properties with each property $P \in \mathsf{P}$ having an associated domain $dom(P)$ of possible values. These are used to express properties of vertices and edges.

1. https://english.elpais.com/society/2020-09-18/chinese-hackers-accused-of-stealing-information-from-spanish-centers-working-on-covid-19-vaccine.html

2. https://www.nytimes.com/2020/07/21/us/politics/china-hacking-coronavirus-vaccine.html

3. We are aware of the fact that the presence of fakes may cause a legitimate user of the system to inadvertently use a fake document instead of the real one. We note that a solution to this based on message authentication codes was already presented in [8] and this solution can be used directly within our PLG framework as well.

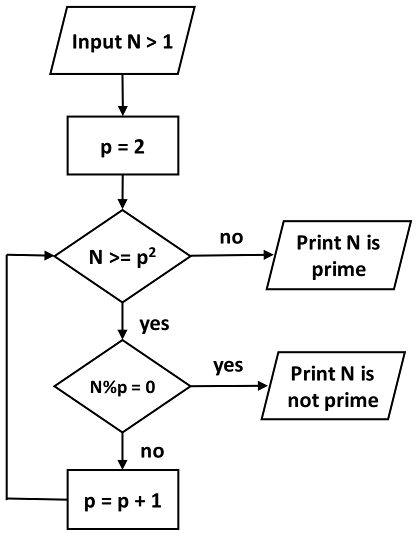


Fig. 2: A flowchart (primality test).

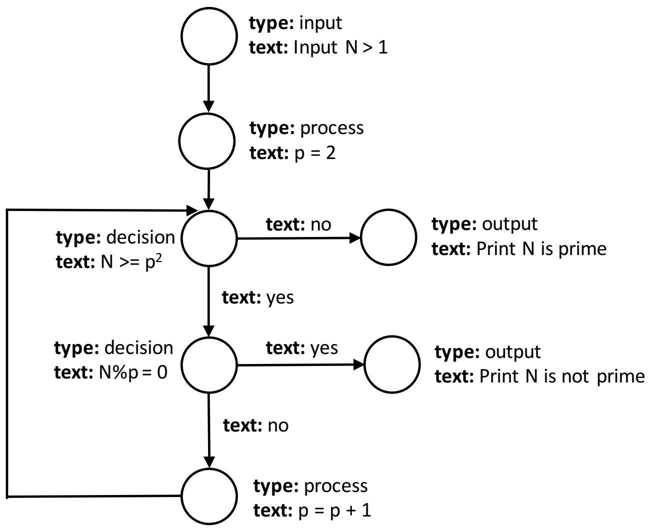


Fig. 3: A logic graph modeling the flowchart of Figure 2.

As usual, a directed graph is a pair (V, E), where V is a finite set of *vertices* and $E \subseteq V \times V$ is a finite set of (directed) *edges*. We enrich graphs with semantic information expressed by means of *annotations*.

An *annotation* is a triple of the form (x, P, p), where $x \in V \cup E$, $P \in \mathsf{P}$, and $p \in dom(P)$—the intuitive meaning is that p is the value of property P for vertex/edge x. A set of annotations is *coherent* if it does not contain two distinct annotations for the same vertex (resp. edge) and property. Thus, a vertex (resp. edge) can have at most one value for a property.

Definition 1 (Logic Graph)**.** *A* logic graph (LG) *is a triple* (V, E, A) *where*

- (V, E) *is a directed graph, and*
- A *is a finite coherent set of annotations.*

LGs allow us to represent the different kinds of content that can be found in documents in a single unified syntax as illustrated in the following two examples.

Example 1. *Consider the flowchart depicted in Figure 2, describing a simple algorithm to check primality. A corresponding LG is depicted in Figure 3, which uses* type *and* text *as properties. The domain of* type *is* {*input, output, decision, process*} *while the domain of* text *is any string.*

Here vertices are used to model flowchart shapes (namely, input, output, decision, and process) while edges model flowlines. Property type *is used to describe the kind of flowchart shape represented by a vertex, while* text *is used to keep track of text annotating flowchart symbols (shapes and flowlines).*

Example 2. *Consider the algebraic expression* $\mathtt{Y} + \mathtt{Z} \times 7$. *A natural LG representing it is depicted in Figure 4, which is essentially the corresponding binary expression tree. Here, each vertex has a* value *property.*

Property value *is used to describe the operator (e.g.,* $+$ *or* $\times$*) or operand (e.g.,* Y*,* Z*, or* 7*) a vertex corresponds to.*

The preceding examples illustrate how LGs enable us to naturally represent different kinds of information. Since the automatic extraction of LGs can be inaccurate, we generalize

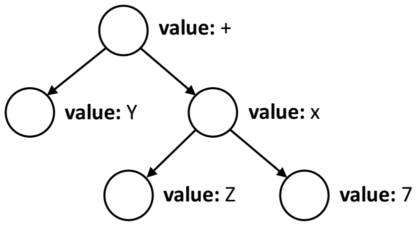


Fig. 4: A logic graph modeling the expression of Example 2.

LGs into *probabilistic logic graphs*, which enable us to model uncertainty.

A *probabilistic annotation* is an annotation along with a probability $pr \in (0, 1]$, denoted as (x, P, p, pr)—the intuitive meaning is that p is the value of property P for vertex/edge x with probability pr. A set of probabilistic annotations PA is *coherent* if

- it does not contain two distinct probabilistic annotations for the same vertex/edge, property, and value; and
- for every $x \in V \cup E$ and $P \in \mathsf{P}$, $\sum_{(x,P,p,pr) \in PA} pr = 1$.

Note that we can model the case where x does not have a value for property P by introducing a distinguished value in $dom(P)$ (e.g., called $null$) meaning that x has no value for P. So we assume that whenever a vertex/edge x of an LG does not have a value for a property P, then the annotation $(x, P, null)$ belongs to the LG.

Definition 2 (Probabilistic Logic Graph)**.** *A* probabilistic logic graph (PLG) *is a triple* (V, E, PA) *where*

- (V, E) *is a directed graph, and*
- PA *is a finite coherent set of probabilistic annotations.*

Example 3. *A PLG for the expression of Example 2 is shown in Figure 5. We show the distribution over the values of property* value *next to each vertex.*

For instance, the lower-right vertex represents the number 7 with probability 0.6 and the number 1 with probability 0.4. These probabilities may result because a classification algorithm may be used during an Optical Character Recognition (OCR) process and it may report a 60% probability that the number is a 7 and a 40%

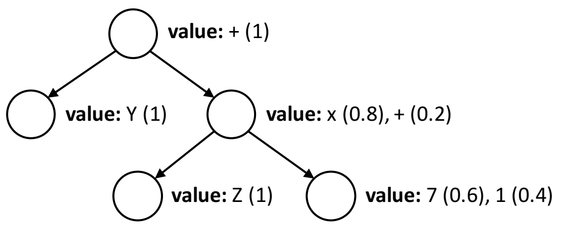


Fig. 5: A PLG for the expression of Example 2.

probability that the number is a 1. We note that virtually every classifier available in packages like `Scikit-Learn` *provides a probability of an object being classified belonging to a class (e.g., the class corresponding to the number 7 vs. the class corresponding to the number 1). Hence, this assumption is compatible with the way classifiers work today. To keep the figure readable, it is assumed that for every edge null is the value of property* value *with probability 1 (which means that every edge has no annotation for sure).*

Semantics. As mentioned above, the goal of PLGs is to model uncertain LGs. We propose a possible-world semantics for PLGs where the semantics of a PLG is given in terms of a probability distribution over the set of all the LGs represented by the PLG.

Definition 3. *An LG $w = (V, E, A)$ is a* possible world *of a PLG $pg = (V', E', PA)$ if*
- $V = V'$,
- $E = E'$, *and*
- *if $(x, P, p) \in A$ then $(x, P, p, pr) \in PA$ for some $pr > 0$.*

The probability of w is

$$Pr(w) = \prod_{(x,P,p,pr) \in PA \wedge (x,P,p) \in A} pr$$

Note that (like much work in AI) the above definition assumes independence between annotations.

Given a PLG *pg*, we use $pw(pg)$ to denote the set of all its possible worlds. It is easy to see that a PLG *pg* induces a probability distribution over the set of its possible worlds, that is, $\sum_{w \in pw(pg)} Pr(w) = 1$.

We conclude this section by noting that given any document d, we can associate a PLG $\mathbb{G}(d)$ with it. Because FORGE [8] has already shown how to represent the textual part of a document as a graph, we do not focus on it here. However, we note that concepts can be extracted from text using many standard concept extraction methods. FORGE uses n-grams as concepts and draws two types of edges between concepts. In FORGE's "syntactic" edge, there is an edge between two concepts if they occur within a window of W concepts. This edge can be labeled with the probability that the two concepts occur within a window of size W. In addition, FORGE introduces a "semantic" notion of edge in which case, there is an edge between concepts c_1, c_2 labeled with the Jaccard distance between the two concepts which can be thought of as a probability $\frac{|c_1 \cap c_2|}{|c_1 \cup c_2|}$. Both these types of graphs in FORGE are PLGs.

3 The Fake PLG Generation Problem

In this section, we formally define the *Fake PLG Generation Problem* and show that it is intractable. We will develop an algorithm to approximately solve the Fake PLG Generation problem in polynomial time in the next section.

Our goal is the following: given a PLG pg_o, we want to generate n (fake) PLGs $pg_1, \ldots, pg_n$ so that an adversary looking at $pg_o, pg_1, \ldots, pg_n$ has no clue about which one was the original PLG. To capture this idea, we require that for every pair of distinct PLGs taken from $\{pg_o, pg_1, \ldots, pg_n\}$ their "difference" should be similar.

Adversary Model. We assume the attacker has two distinct types of people. (a) First, there are computer scientists who are capable of penetrating the victim enterprise. Given a set $\{d_0, d_1, \ldots, d_n\}$ where d_0 is the original document and $d_1, \ldots, d_n$ are fakes, we assume the attacker can create a fully connected network whose nodes are $d_0, d_1, \ldots, d_n$ and where each edge (d_i, d_j) is labeled with the distance between those two documents according to a given distance function Δ. In our attack model, we assume that the attacking computer scientists can analyze this network and find central nodes in it. (b) We assume the attacker also has domain experts (e.g., in chemistry or in pharmaceutical design) who can read a document and determine if it is real or fake. In order to deceive these domain experts, we ensure that the fakes are "near enough" (according to Δ) to the original to be credible but sufficiently "far" in order to be wrong.

This in turn requires defining a distance function Δ for PLGs, which can be done in different ways, depending on the application domain. Rather than committing to a specific one, we assume Δ is an arbitrary metric—however, in the following, we will discuss a concrete distance function that we used in our experimental evaluation, which is reasonable enough to be applied in several domains.

Furthermore, we argue that the generation of fake PLGs should not be done in a completely blind way, as every domain has rules saying when LGs are meaningful, and these rules should play a role in the PLG generation process to avoid PLGs that make no sense from a domain perspective (and hence are easily detectable as fake). We illustrate this factor in the following two examples.

Example 4. *Consider again the flowchart domain and in particular the LG depicted in Figure 6a. It is easy to see that this LG represents the flowchart reported in Figure 6b, but it is "inconsistent" in several different respects. For instance, the upper decision point has only one outgoing flowline, while the lower one has none.*

Example 5. *Consider now the algebraic expression domain and the LG depicted in Figure 7. It is easy to see that it represents the expression* Y $+$ /3$+$, *which does not make much sense.*

The two examples above clearly show that, depending on the domain of interest, an LG might be subject to certain constraints in order for it to be meaningful. Each application domain may impose its own rules on the structure of valid LGs (as a further example, in the algebraic expression domain, LGs are expected to be trees).

As a PLG represents a set of possible LGs, each with a probability, some of them might comply with the domain constraints, while others might not. To measure the extent to which a PLG complies with the domain constraints, we need to look at the LGs it represents, and their probabilities:

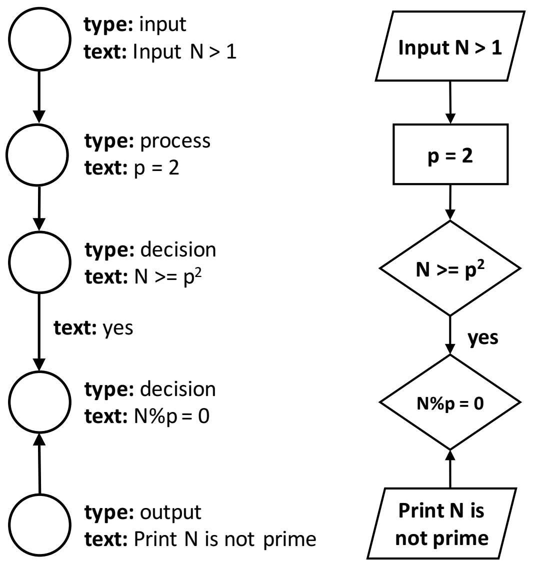


Fig. 6: A (bad) logic graph (left) and the flowchart it represents (right).

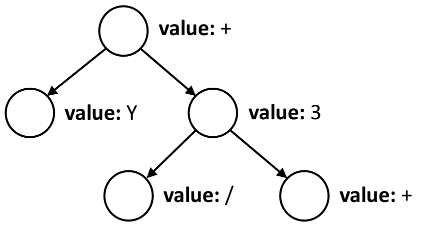


Fig. 7: A (bad) LG representing a non-well-formed expression.

a natural approach is to sum the probabilities of the LGs satisfying the domain constraints, which we define as the *degree of consistency* of the PLG (a formal definition will be provided shortly). Hence, we assume the existence of an *admissibility function* φ taking as input an LG g and returning true if g is admissible and false otherwise. Function φ is a means to embed semantic information of the application domain into our framework and avoid blindly generating a PLG whose possible worlds are so wrong that the PLG is easily identifiable as synthetically generated, as illustrated in the following example.

Example 6. *Consider the PLG of Figure 5. It is easy to see that all its possible worlds are LGs representing valid algebraic expressions.*

Consider now a PLG that is identical to the one of Figure 5, except that the probabilistic annotations for the bottom-right vertex, say v, are $(v, \mathsf{value}, 7, 0.6)$ and $(v, \mathsf{value}, +, 0.4)$. Clearly, the possible worlds where the probabilistic annotation becomes $(v, \mathsf{value}, 7)$ are valid ones, while those where it becomes $(v, \mathsf{value}, +)$ are not.

We can now define the degree of consistency of a PLG as the sum of the probabilities of the possible worlds complying with φ.

Definition 4. *The* degree of consistency *of a PLG pg w.r.t. an admissibility function φ is*

$$C_\varphi(pg) = \sum_{w \in pw(pg) \wedge \varphi(w) = true} Pr(w).$$

As stated in the following theorem, computing the degree of consistency is a #P-hard problem.

Theorem 1. *Let pg be a PLG and φ an admissibility function. Computing the degree of consistency of pg w.r.t. φ is #P-hard, even when φ can be evaluated in polynomial time.*

Proof. We show a reduction from #SAT, that is, the problem of computing the number of satisfying assignments of a propositional formula, which is a #P-hard problem.

Let Φ be an instance of #SAT, that is, a propositional formula whose propositional variables are $x_1, \ldots, x_n$.

We derive a PLG $pg_\Phi = (V, E, PA)$ and an admissibility function φ_Φ as follows.

We use only one property value whose domain is $\{true, \; false\}$. Then, $V = \{x_1, \ldots, x_n\}$, $E = \emptyset$, and $PA = \bigcup_{1 \le i \le n} \{(x_i, \mathsf{value}, true, 0.5), (x_i, \mathsf{value}, false, 0.5)\}$. Notice that each possible world of pg_Φ is an LG (V, E, A) with $A = \{(x_i, \mathsf{value}, t_i), \ldots, (x_n, \mathsf{value}, t_n)\}$, where the t_i's are either $true$ or $false$, and thus it corresponds to a truth-value assignment over the propositional variables of Φ. Finally, for every possible world w of pg_Φ, we define $\varphi_\Phi(w)$ as true if its corresponding truth-value assignment satisfies Φ, false otherwise—notice that this can be verified in polynomial time.

Clearly, for every possible world w of pg_Φ, $Pr(w) = 1/2^n$. Thus, $C_\varphi(pg_\Phi) = \#\Phi/2^n$, where $\#\Phi$ is the number of assignments satisfying Φ. ☐

As already mentioned before, our goal is to generate highly believable fakes PLGs. In this regard, note that C_φ and Δ play two complementary and different roles.

The degree of consistency C_φ measures how consistent a fake PLG is w.r.t. the constraints of the application domain. Such constraints are rarely inferrable from the original PLG and thus must be made explicit, which is why we require φ as an input from a domain expert (or φ could just be a piece of code, e.g. a piece of code to detect if an algebraic expression is syntactically valid or a piece of code to detect if a flowchart is well formed). It is clear that PLGs with a low degree of consistency should be avoided as easily detectable as fake. Thus, the aim of C_φ is to make sure that PLGs are believable enough, *regardless of the original PLG.*

The distance function Δ measures the distance between PLGs, with the aim of generating a set of fake PLGs that along with the original one have a pairwise distance lying within a given interval. This means that every pair of distinct PLGs has pretty much the same distance, so as to prevent attacks such as using network analysis, centroid computation (or similar methods) to recover the original PLG. In other words, here the goal is to make sure that it is *sufficiently hard* to single out the original PLG from a set of reasonable ones.

The *fake PLG generation problem* is defined as follows.

Definition 5. *An instance of the* Fake PLG Generation Problem *is a tuple $\langle pg_o, n, [\ell, u], \theta, \Delta, \varphi \rangle$, where pg_o is a PLG, n is a positive integer, $[\ell, u]$ is an interval, $\theta \in [0, 1]$, Δ is a distance function between PLGs, and φ is an admissibility function.*

A solution is a set of n distinct PLGs $pg_1, \ldots, pg_n$ s.t.

1) *every pg_i is different from pg_o,*
2) *for every pg_i, $C_\varphi(pg_i) \geq \theta$,*
3) *for every pair of distinct PLGs pg' and pg'' in $\{pg_o, pg_1, \ldots, pg_n\}$, $\Delta(pg', pg'') \in [\ell, u]$.*

Thus, for a given PLG pg_o, we want to find n distinct PLGs that are *(1)* different from pg_o, *(2)* with a degree of consistency no lower than a given threshold θ, and *(3)* s.t. for every pair of distinct PLGs, their difference lies in a given interval $[\ell, u]$. It is worth noting that this latter requirement bounds the distance between every pair of PLGs in $\{pg_o, pg_1, \ldots, pg_n\}$, and thus it bounds the distance between the original and the fakes. More specifically, the upper bound u is used to impose the requirement $\Delta(pg', pg'') \leq u$, which allows users to set a maximum desired distance between every pair of distinct PLGs. This in turn means that fake PLGs must differ from the original one at most of u, so that fake PLGs cannot be too far from the original, in order to prevent them to be remarkably different. Likewise, the lower bound ℓ is used to impose the requirement $\ell \leq \Delta(pg', pg'')$, which allows users to set a minimum desired distance between every pair of distinct PLGs in $\{pg_o, pg_1, \ldots, pg_n\}$. This also means that fake PLGs must be "far enough" from the original one. Clearly, ℓ, u, and Δ can be arbitrarily chosen depending on the application and users' needs.

The following result shows that the fake PLG generation problem is NP-hard.

Theorem 2. *The fake PLG generation problem is NP-hard, even when Δ and φ can be computed in polynomial time and $n = 1$.*

Proof. We show a reduction from the (NP-hard) 3COLORABILITY problem to the problem of deciding whether an instance of the fake PLG generation problem has a solution. 3COLORABILITY is the problem of deciding whether a graph is 3-colorable, i.e., whether we can assign one of three colors to every vertex in such a way that no two adjacent vertices have the same color.

Let $G = (V, E)$ be an instance of 3COLORABILITY, that is, a graph. Let $V = \{v_1, \ldots, v_n\}$. We use only one property color with $dom(\text{color}) = \{c_1, c_2, c_3\}$. We derive an instance $\langle pg_o, n, [\ell, u], \theta, \Delta, \varphi \rangle$ of the fake PLG generation problem as follows:

- $pg_o = (V, E, PA)$, where $PA = PVA \cup PEA$ with
$$PVA = \bigcup_{1 \leq i \leq n} \{ \begin{array}{l} (v_i, \text{color}, c_1, 1/3), \\ (v_i, \text{color}, c_2, 1/3), \\ (v_i, \text{color}, c_3, 1/3)\} \end{array}$$
and $PEA = \bigcup_{(v_i, v_j) \in E} \{((v_i, v_j), \text{color}, c_1, 1)\}$.
- $n = 1$.
- $[\ell, u] = [0, 0]$.
- $\theta = 1/3^n$.
- For every pair of PLGs $pg' = (V', E', PA')$ and $pg'' = (V'', E'', PA'')$,
$$\Delta(pg', pg'') = \begin{cases} 0 & \text{if } V' = V'', E' = E'', \text{ and} \\ & \{(v, P, p, Pr) \mid v \in V'\} = \\ & \{(v, P, p, Pr) \mid v \in V''\} \\ 1 & \text{otherwise.} \end{cases}$$
- For every LG $g = (V', E', A')$, $\varphi(g)$ is true iff A' does not contain two distinct annotations (v_1, color, c_i) and (v_2, color, c_i) s.t. (v_1, v_2) or (v_2, v_1) belongs to E', for some $c_i \in dom(\text{color})$.

Then, G is 3-colorable iff the above instance of the fake PLG generation problem has a solution.

($\Rightarrow$) Suppose G is 3-colorable. Let pg_1 be a PLG that is identical to pg_o except that a probabilistic annotation $(e, \text{color}, c_1, 1)$ of pg is changed into $(e, \text{color}, c_2, 1)$ in pg_1 for an arbitrary edge $e \in E$. Below we show pg_1 is a solution of the fake PLG generation problem instance above.

Obviously, pg_1 is different from pg_o.

Consider a possible world $g = (V, E, A)$ of pg_1 s.t. for a valid 3-coloring of G, $(v, \text{color}, c_i) \in A$ iff c_i is the color of v in the 3-coloring—the existence of a valid 3-coloring ensures the existence of such a world. Clearly, $\varphi(g)$ is true, and since $Pr(g) = 1/3^n$, then $C_\varphi(pg_1) \geq \theta$.

Finally, notice that $\Delta(pg_o, pg_1) = 0$, as pg_o and pg_1 have the same vertices and edges and their vertex probabilistic annotations are the same. Thus, $\Delta(pg_o, pg_1) \in [\ell, u] = [0, 0]$.

($\Leftarrow$) Suppose the instance above of the fake PLG generation problem has a solution, say pg_1. By definition of solution and by construction of the instance, $\Delta(pg_o, pg_1)$ has to be 0, which means that vertices, edges, and vertex probabilistic annotations of pg_o and pg_1 are the same. Thus, each possible world of pg_1 corresponds to a color assignment for the vertices of G.

In order for pg_1 to be a solution, $C_\varphi(pg_1) \geq \theta$ must holds true, which means that there must be a world of pg_1 (i.e., a color assignment for the vertices of G) satisfying φ, that is, corresponding to a valid 3-coloring. □

4 A Greedy Algorithm

Since exactly solving the fake PLG generation problem is likely to take inordinate amounts of time because of the NP-hardness result, in this section we propose a greedy algorithm to solve the problem approximately. The basic idea is as follows.

1) The algorithm starts with a set Res containing only the original PLG and iteratively adds one more PLG to Res (this will eventually be the output of the algorithm).
2) At each iteration, one PLG is generated (and added to Res) by applying an operator to one of the PLGs in Res. The choice of the operator and of the PLG to be modified is guided by Δ and C_φ.

In the rest of this section, we first define a suite of operators to generate PLGs. We then discuss a heuristic to properly choose which operator should be applied to which PLG (according to Δ and C_φ). Finally, we present the complete algorithm.

4.1 PLG Operators

To manipulate PLGs, we consider operators that add, delete, and modify vertices and edges. Recall that PLGs embed semantic information via (probabilistic) annotations, that is, vertices and edges are associated with coherent sets of probabilistic annotations (see Section 2 and Definition 2), which must be taken into account by the aforementioned operators. In particular, a new vertex/edge must come along with a coherent set of probabilistic annotations, while the modification of existing probabilisitic annotations must ensure coherency is preserved.

More specifically, we introduce the following operators to manipulate PLGs.

- Add a new vertex/edge along with a coherent set of probabilistic annotations for the vertex/edge.
- Delete a vertex, and all its incident edges (along with their probabilistic annotations).
- Delete an edge (along with its probabilistic annotations).
- Modify the probabilistic annotations for a vertex/edge x and a property P, while preserving coherence (i.e., modify the probability distribution over the possible values of P for x).

More formally, let $pg = (V, E, PA)$ be a PLG. We assume the existence of a set $\mathcal{O}$ of possible *operations* of the following form:

- $\mathsf{AddVertex}(v, PA')$, where v is a new vertex and PA' is a coherent set of probabilistic annotations for v. The result of applying it to pg is the following PLG:
$$(V \cup \{v\}, E, PA \cup PA').$$
- $\mathsf{AddEdge}(e, PA')$, where e is a new edge and PA' is a coherent set of probabilistic annotations for e. The result of applying it to pg is the following PLG:
$$(V, E \cup \{e\}, PA \cup PA').$$
- $\mathsf{DeleteVertex}(v)$, where $v \in V$. The result of applying it to pg is the following PLG:
$$(V \setminus \{v\}, E, PA \setminus \{(v', P', p', pr') \in PA \mid v' = v\}).$$
Furthermore, all edges incident to v are deleted as defined in the next item.
- $\mathsf{DeleteEdge}(e)$, where $e \in E$. The result of applying it to pg is the following PLG:
$$(V, E \setminus \{e\}, PA \setminus \{(e', P', p', pr') \in PA \mid e' = e\}).$$
- $\mathsf{Modify}(x, P, PA')$, where $x \in V \cup E$, $P \in \mathsf{P}$, and PA' is a coherent set of probabilistic annotations for x and P. The result of applying it to pg is the following PLG:
$$(V, E, PA \setminus \{(x', P', p', pr') \in PA \mid x' = x, P' = P\} \cup PA').$$

The result of applying an operation $op \in \mathcal{O}$ to a PLG pg is denoted as $op(pg)$. [4]

4.2 Heuristic

As already mentioned before, the choice of which operation should be applied to which PLG should be guided by Δ and C_φ. We now provide a heuristic to make such a decision. First, we define how Δ is taken into account to guide the decision. We then propose an approximation algorithm to compute C_φ (recall that the exact computation is a #P-hard problem). Finally, we define a heuristic that combines the two criteria.

4. It is important to note that it is possible to expand the PLG framework to include additional operations. We define a core set of such operations for now that others may expand as needed.

Taking Δ into account. Given two PLGs pg and pg', a distance function Δ, and an interval $[\ell, u]$, we define

$$dist_\Delta(pg, pg', [\ell, u]) = \begin{cases} 0 & \text{if } \Delta(pg, pg') \in [\ell, u], \\ \ell - \Delta(pg, pg') & \text{if } \Delta(pg, pg') < \ell, \\ \Delta(pg, pg') - u & \text{if } \Delta(pg, pg') > u. \end{cases}$$

Thus, $dist_\Delta(pg, pg', [\ell, u])$ provides one way to measure the distance (how far) between pg and pg' from the interval $[\ell, u]$.

Approximating C_φ. As stated in Theorem 1, computing the degree of consistency $C_\varphi(pg)$ of a PLG pg w.r.t. an admissibility function φ is a #P-hard problem (even when φ can be evaluated in polynomial time). Recall that $C_\varphi(pg)$ is the sum of the probabilities of the possible worlds of pg satisfying φ. Clearly, the high complexity comes from the exponential number of possible worlds (in the worst case). We propose to approximate $C_\varphi(pg)$ by looking only at the top-k most probable possible worlds. More formally, we use $pw^k(pg)$ to denote the set of top-k most probable possible worlds of pg. Then, our goal is to compute

$$C^k_\varphi(pg) = \sum_{w \in pw^k(pg) \wedge \varphi(w) = true} Pr(w).$$

Clearly, resorting to $C^k_\varphi(pg)$ is useful if we can devise an algorithm to compute it without enumerating all possible worlds. An optimal algorithm for this purpose should look at the possible worlds in $pw^k(pg)$ only, and avoid the enumeration of the remaining ones altogether. Thus, here the challenge is how to enumerate *exactly* (that is, all and only) the top-k most probable possible worlds of pg. Algorithm APPROXIMATETOPKCONSISTENCYDEGREE (cf. Algorithm 1) achieves this goal and exactly computes $C^k_\varphi(pg)$. It takes as input: a PLG $pg = (V, E, PA)$, an admissibility function φ, and a positive integer $k > 0$, and it computes $C^k_\varphi(pg)$. Specifically, it iteratively adds the k most probable worlds to $pw^k(pg)$, and eventually uses them to compute $C^k_\varphi(pg)$. The key aspect of the algorithm is how it determines the next most probable possible world, which allows it to consider only the possible worlds needed to compute $C^k_\varphi(pg)$. We go into the details in more depth below.

The set $\mathcal{S}$ stores tuples of the form $(x, P, PAnn)$, where x is either a vertex or an edge of pg, P is a property, and $PAnn$ is a list containing all probabilistic annotations of pg for x and P, sorted by decreasing Pr value (lines 2–6).

In lines 7–12, the most probable world (V, E, A_1) is computed as follows: for each vertex/edge x and property P, the value of P for x is the one with the highest probability—we use $PAnn[j]$ to denote the j-th element of list $PAnn$.

The remaining possible worlds are then computed in the **for** loop of lines 13–20. Specifically, at each iteration a possible world (V, E, A_i) is computed. This new possible world is obtained from the possible world of the previous iteration by modifying exactly one value for some vertex/edge x^* and property P^*. These are chosen from $\mathcal{S}$ so that $PAnn[0].Pr/PAnn[1].Pr$ is minimum across all (vertex/edge,property) pairs (line 14), where $PAnn[0].Pr$ (resp. $PAnn[1].Pr$) denotes the probability value in the probabilistic annotation $PAnn[0]$ (resp. $PAnn[1]$). This represents the greedy choice step. Thus, p_0 and p_1 are the probability values of $PAnn^*[0]$ and $PAnn^*[1]$, respectively (lines 15–16).

Algorithm 1 ApproximateTopKConsistencyDegree

```
Input: A PLG $pg$, an admissibility function $\varphi$, and $k > 0$.
Output: $C^k_\varphi(pg)$.
 1: Let $pg = (V, E, PA)$;
 2: $\mathcal{S} = \emptyset$;
 3: for each $x \in V \cup E$ and $P \in \mathsf{P}$ do
 4:     Let $PAnn = \{(x, P, p, Pr) \in PA\}$ be sorted by decreasing $Pr$;
 5:     $\mathcal{S} = \mathcal{S} \cup \{(x, P, PAnn)\}$;
 6: end for
 7: $A_1 = \emptyset$;
 8: for each $(x, P, PAnn)$ in $\mathcal{S}$ do
 9:     Let $(x, P, p, Pr) = PAnn[0]$;
10:     $A_1 = A_1 \cup \{(x, P, p)\}$;
11: end for
12: $pw^k(pg) = \{(V, E, A_1)\}$;
13: for $i = 2$ to $k$ do
14:     $(x^*, P^*, PAnn^*) = \underset{(x,P,PAnn)\in\mathcal{S}}{\arg\min}\{PAnn[0].Pr/PAnn[1].Pr\}$;
15:     Let $(x^*, P^*, p_0, Pr_0) = PAnn^*[0]$;
16:     Let $(x^*, P^*, p_1, Pr_1) = PAnn^*[1]$;
17:     Delete $PAnn^*[0]$ from $PAnn^*$;
18:     $A_i = A_{i-1} \setminus \{(x^*, P^*, p_0)\} \cup \{(x^*, P^*, p_1)\}$;
19:     $pw^k(pg) = pw^k(pg) \cup \{(V, E, A_i)\}$;
20: end for
21: return $\sum_{w \in pw^k(pg) \wedge \varphi(w)=true} Pr(w)$;
```

After this, the first element of $PAnn^*$ is deleted (line 17), and the new possible world (V, E, A_i) is obtained from the one of the previous iteration (V, E, A_{i-1}) by replacing the value p_0 (for x^* and P^*) with p_1 (line 18). The new world is then added to $pw^k(pg)$ (line 19). Eventually, worlds in $pw^k(pg)$ are used to compute $C^k_\varphi(pg)$ (line 21).

We now state the correctness theorem and assess the complexity of Algorithm 1.

Theorem 3. *Given a PLG pg, an admissibility function φ, and a positive integer $k > 0$, Algorithm 1 correctly computes $C^k_\varphi(pg)$.*

Proof. First of all, notice that each possible world is derived from pg by choosing one value for each vertex/edge $x \in V \cup E$ and property $P \in \mathsf{P}$ among those in PA, and the probability of the possible world is the product of the values' probabilities.

Clearly, (V, E, A_1) is the most probable world, as it picks the highest probability for each $x \in V \cup E$ and $P \in \mathsf{P}$.

Then, by induction on i $(i > 1)$, we can show that each subsequent possible world w_i is the i-th most probable one if w_{i-1} is the $(i-1)$-th most probable one. World w_i is constructed from w_{i-1} as follows: for some vertex/edge $x \in V \cup E$ and property $P \in P$, their value p_0 having probability Pr_0 is replaced with a value p_1 having probability Pr_1 provided that Pr_0/Pr_1 is minimum across all possible replacements. We show that this (somewhat greedy) choice is indeed optimal for the purpose of finding the next most probable world. By contradiction, suppose there is a world w'_i s.t. $Pr(w'_i) > Pr(w_i)$. Suppose that w'_i is derived from w_{i-1} by replacing a value p'_0 having probability Pr'_0 with a value p'_1 having probability Pr'_1, for some vertex/edge $x' \in V \cup E$ and property $P' \in P$. Then, $Pr(w_i) = Pr_{prev} \times Pr'_0 \times Pr_1$ and $Pr(w'_i) = Pr_{prev} \times Pr_0 \times Pr'_1$, where Pr_{prev} is the product of the probabilities for all vertices/edges and properties except (x, P) and (x', P') in w_{i-1}. As $Pr(w'_i) > Pr(w_i)$, then $(Pr'_0/Pr'_1) < (Pr_0/Pr_1)$, which is a contradiction. □

Proposition 4. *The worst-case time complexity of Algorithm 1 is $O((|V \cup E| \cdot |\mathsf{P}|) \cdot (k + PAnn_{max} \cdot \log PAnn_{max}) + k \cdot f_\varphi)$, where f_φ is the worst-case time complexity of evaluating φ, and $PAnn_{max} = \max_{x \in V \cup E, P \in \mathsf{P}} \{|\{(x, P, p, Pr) \in PA\}|\}$.*

Proof. Obviously, lines 1–2 have cost $O(1)$. The **for each** loop on lines 3–5 performs $|V \cup E| \cdot |\mathsf{P}|$ iterations. At each iteration $PAnn$ is sorted with cost $O(PAnn_{max} \cdot \log PAnn_{max})$, as its length is at most $PAnn_{max}$, while line 5 requires constant time. Thus, the overall cost of the **for each** loop on lines 3–5 is $O(|V \cup E| \cdot |\mathsf{P}| \cdot PAnn_{max} \cdot \log PAnn_{max})$.

Line 7 has cost $O(1)$. The **for each** loop on lines 8–11 performs $|V \cup E| \cdot |\mathsf{P}|$ iterations, where each iteration requires constant time; thus the overall cost is $O(|V \cup E| \cdot |\mathsf{P}|)$. Line 12 has cost $O(1)$.

The **for** loop on lines 13–20 performs $k-1$ iterations. At each iteration, lines 14–19 are executed. Line 14 requires finding a minimum value across all tuples in $\mathcal{S}$, which are $|V \cup E| \cdot |\mathsf{P}|$, which is thus the cost of line 14. Lines 15–19 require constant time. Thus, the overall cost of lines 13-20 is $O(k \cdot |V \cup E| \cdot |\mathsf{P}|)$.

Line 21 looks at k possible worlds and for each of them it evaluates φ, thus its overall cost is $O(k \cdot f_\varphi)$.

The overall cost of the algorithm is thus

$$O(|V \cup E| \cdot |\mathsf{P}| \cdot PAnn_{max} \cdot \log PAnn_{max} + k \cdot |V \cup E| \cdot |\mathsf{P}| + k \cdot f_\varphi),$$

which can be rewritten as in the claim. □

We point out that a variant of ApproximateTopKConsistencyDegree might additionally take as input a threshold θ so that the algorithm halts as soon as it finds $k' \leq k$ possible worlds satisfying φ and whose overall probability is at least θ.

Heuristic definition. We are now ready to define our heuristic, which combines $dist_\Delta$ and C^k_φ. To keep the notation simple, the distance and admissibility functions Δ and φ are implicit and omitted in the H notation below. Furthermore, given a value $C^k_\varphi(pg)$ and $\theta \in [0, 1]$, we define

$$||\theta - C^k_\varphi(pg)|| = \begin{cases} 0 & \text{if } C^k_\varphi(pg) \geq \theta \\ \theta - C^k_\varphi(pg) & \text{otherwise} \end{cases}$$

Let pg be a PLG, PG a set of PLGs, $k > 0$, $\theta \in [0, 1]$, and $[\ell, u]$ an interval. We define $H(pg, PG, k, \theta, [\ell, u])$ as follows:

$$w_1 \cdot (||\theta - C^k_\varphi(pg)||) + w_2 \cdot (\sum_{pg' \in PG} dist_\Delta(pg, pg', [\ell, u]))$$

where w_1 and w_2 are used to weight the contributions of the degree of consistency and of the distance function, respectively.

The lower the H value, the better the choice of pg, as this corresponds to a C^k_φ value closer to (or above) θ—i.e., the (approximate) degree of consistency is closer to or above the threshold—and lower $dist_\Delta$ values—i.e., the distances of pg from the PLGs in PG are not far from the desired interval $[\ell, u]$.

Algorithm 2 GreedyFakePLGs

```
Input: An instance $\langle pg_o, n, [\ell, u], \theta, \Delta, \varphi\rangle$ of the fake PLG generation problem and $k > 0$.
Output: PLGs $pg_o, pg_1, \ldots, pg_n$.
1: $Res = \{pg_o\}$;
2: for $i = 1$ to $n$ do
3:     $(op^*, pg^*) = \arg\min_{op \in \mathcal{O}, pg \in Res, op(pg) \notin Res} H(op(pg), Res, k, \theta, [\ell, u])$;
4:     $pg_i = op^*(pg^*)$;
5:     $Res = Res \cup \{pg_i\}$;
6: end for
7: return $Res$;
```

Proposition 5. *The worst-case time complexity of computing H is $O(|PG| \cdot f_\Delta + f_{C^k_\varphi})$, where f_Δ is the worst-case time complexity of computing Δ and $f_{C^k_\varphi}$ is the wort-case time complexity of computing C^k_φ (cf. Proposition 4).*

Proof. Straightforward. □

4.3 The GreedyFakePLGs Algorithm

To solve the fake PLG generation problem we propose the GreedyFakePLGs algorithm (cf. Algorithm 2) which takes an instance of the fake PLG generation problem along with a positive integer k (which is used for the computation of C^k_φ when the H value is calculated) as input. It returns as output the original PLG along with a set of n fake PLGs.

The algorithms starts by initializing a set Res to contain only the original PLG pg_o (line 1)—Res will eventually be the output (see line 6). It then performs n iterations (lines 2–5), and at each iteration, it chooses the operation op^* and the PLG pg^* from Res that minimize the heuristic value H (line 3). The application of op^* to pg^* yields the new PLG pg_i (line 4), which is added to Res (line 5).

The following simple proposition states the time complexity of Algorithm 2.

Proposition 6. *The worst-case time complexity of Algorithm 2 is $O(n^2 \cdot |\mathcal{O}| \cdot f_H)$, where f_H is the wort-case time complexity of computing H (cf. Proposition 5).*

Proof. The algorithm performs n iterations. At each iteration it has to compute H for at most $n \cdot |\mathcal{O}|$ candidate PLGs, as there are at most $n \cdot |\mathcal{O}|$ pairs of operations and (source) PLGs to be considered. It is straightforward to see that all remaining instructions can be performed in constant time. □

5 A Complete Example

In this section, we illustrate our framework by showing its behavior on an excerpt taken from a real patent.

Figure 8 shows the overall process, which consists of: (Step 1) mapping the snippet of the document into a PLG, (Step 2) generating fake PLG(s), and (Step 3) mapping fake PLG(s) back into (fake) documents. For simplicity of exposition, only one fake document is shown for this patent.

Figure 8(a) shows an excerpt taken from a real patent[5]. containing 3 types of diverse content: text, an equation, and a flowchart. More importantly, these three types of content share common pieces of information. For instance, sn appears in both the text and the equation, while vt and vu appear in both the equation and the flowchart.

A possible PLG for the document of Figure 8(a) is reported in Figure 8(b). For the sake of readability, we assumed that each property has one single value (the one reported next to the property) with probability 1, and we omitted such probability values. Dashed boxes highlight how the text, flowchart, and equation have been modeled, even though there are vertices common to the different kinds of content.

Figure 8(c) shows a fake PLG for the one in Figure 8(b). Notice that sn has been modified into rt, the vertex with **value:** ˆ2 has been added, the edge at the bottom has been deleted, etc.

The document corresponding to the fake PLG is shown in Figure 8(d).

6 Implementation and Experimental Evaluation

Any implementation of PLGs should broadly consist of two parts: a part to transform an original document d_0 into a PLG pg_0 and a part to map the fake PLGs $pg_1, \ldots, pg_n$ into fake documents $d_1, \ldots, d_n$ according to the nomenclature in Figure 1. Our current PLG system has fully implemented the second part, namely the mapping from the fake PLGs $pg_1, \ldots, pg_n$ into fake documents $d_1, \ldots, d_n$. For the first part, we do have code from [8] to map the textual part of a document into graph syntax which can be easily extended to map onto PLGs, but the code needed to transform diagrams, for instance, into PLGs requires considerable effort in image processing. We are currently pursuing this, but as this poses a whole separate challenge, we defer it to future work.

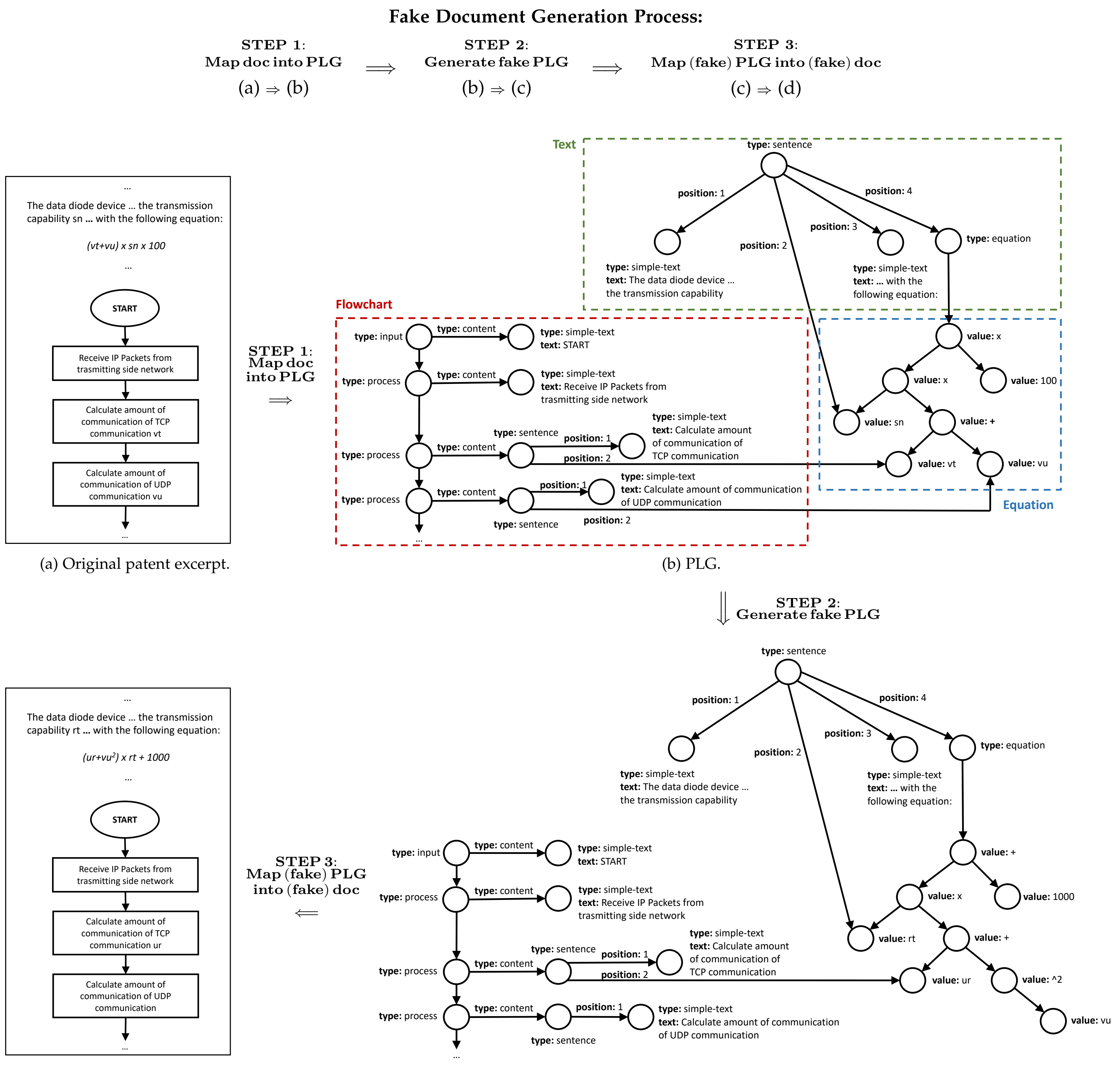
In this section, we report on an experimental evaluation we performed on real patents to assess the effectiveness of our approach. We first detail the experimental setup, then describe our evaluation process, and finally present experimental results.

6.1 Experimental Setup

We applied the GreedyFakePLGs algorithm to real-world patents in the Computer Science domain. Specifically, we collected 10 patents on Google Patents, each consisting of general text, flowcharts, equations, and other kinds of content and manually constructed their PLG pg_0 representation (as per Figure 1). The remainder of the GreedyFakePLGs algorithm is fully automated.

We now describe how we defined Δ, φ, and $\mathcal{O}$ in our setting. The distance function Δ was defined to measure both the topological difference between two PLGs (i.e., differences w.r.t. the vertex and edge sets) and differences in the probabilistic annotations (i.e., for each vertex/edge and property belonging to both PLGs, we measured the difference between their probability distributions). The formal definition of Δ between two PLGs pg_1 and pg_2 is as follows. We sum together the probability difference of each value p for each property P of each vertex and edge. Suppose pg_1 and pg_2 are PLGs having the form (V_1, E_1, PA_1) and

5. https://patents.google.com/patent/US20170118123A1/en


Fake Document Generation Process:
STEP 1: Map doc into PLG (a) ⇒ (b)
STEP 2: Generate fake PLG (b) ⇒ (c)
STEP 3: Map (fake) PLG into (fake) doc (c) ⇒ (d)
...
The data diode device ... the transmission capability sn ... with the following equation:
(vt+vu) x sn x 100
START
Receive IP Packets from trasmitting side network
Calculate amount of communication of TCP communication vt
Calculate amount of communication of UDP communication vu
STEP 1: Map doc into PLG
Text
Flowchart
Equation
type: sentence
position: 1
position: 2
position: 3
position: 4
type: simple-text
text: The data diode device ... the transmission capability
text: ... with the following equation:
type: equation
type: input
type: process
type: content
text: START
text: Receive IP Packets from trasmitting side network
text: Calculate amount of communication of TCP communication
text: Calculate amount of communication of UDP communication
value: x
value: 100
value: sn
value: +
value: vt
value: vu
STEP 2: Generate fake PLG
value: 1000
value: rt
value: ur
value: ^2
STEP 3: Map (fake) PLG into (fake) doc
The data diode device ... the transmission capability rt ... with the following equation:
(ur+vu²) x rt + 1000
Calculate amount of communication of TCP communication ur
Calculate amount of communication of UDP communication

(a) Original patent excerpt.

(b) PLG.

(c) Fake PLG.

(d) Fake patent excerpt.

Fig. 8: Fake document generation process.

(V_2, E_2, PA_2), respectively. Suppose $(x, P, p, -) \in PA_2$ and there is no probabilistic annotation of the form $(x, P, p, -) \in PA_1$, then we add $(x, P, p, 0)$ to PA_1 and vice-versa. In this case, PA_1 and PA_2 would have the same number of elements. We then define:

$$\Delta(pg_1, pg_2) = \sum_{(x,P,p,Pr_1)\in PA_1 \wedge (x,P,p,Pr_2)\in PA_2} |Pr_1 - Pr_2|.$$

Since our dataset consists of patents with flowcharts, algebraic expressions, equations, etc., φ was defined to capture natural properties that characterize LGs modeling such kinds of content. For instance, for flowcharts rules include: the LG is connected, vertices corresponding to input (resp. output) shapes have no incoming (resp. outgoing) edges, vertices corresponding to decision shapes have at least one incoming edge and at least two outgoing edges, etc. As a further example, for LGs representing algebraic expressions typical rules are: intermediate vertices correspond to operators, leaves correspond to operands, etc.

The set $\mathcal{O}$ of possible operations was defined so as to allow arbitrary vertex/edge deletions as well as arbitrary insertions and modifications of vertices and edges using randomly generated probabilistic annotations.

6.2 Evaluation Process

Since the space of parameter settings is huge and evaluating each of them requires time-consuming manual intervention, we designed a two-step evaluation process: in the first step we determine the three best parameter settings; in the second step, we use them to evaluate the effectiveness of our approach. We invited 20 human subjects with expertise in Computer Science, 10 for the first step and 10 different ones for the second step. They were recruited via a public call at the Department of Electrical Engineering and Information Technology of the University Of Naples "Federico II" (Italy). All subjects have a Master or a PhD degree in Computer Science, with age in the range [24,42] (average age 27), 15 males and 5 females, with high-level proficiency in English.

We asked subjects to select the top 3 documents they felt were the original patent, ranking their choices. The documents (the original one + the fakes) were displayed in randomized order. The human subjects were informed that only one of the documents was a real, original patent, while the remaining were all fakes generated from the original.

Step 1. We selected one patent and generated 3 fake documents for each of the 18 parameter settings obtained by varying k, θ, and $[\ell, u]$ as follows:

- $k \in \{4, 8, 16\}$;
- $\theta \in \{0.5, 0.75\}$;
- $[\ell, u] \in \{[0, 0.25], [0.25, 0.5], [0.5, 0.75]\}$;

In total, for each original document, there are 54 fake documents presented to human reviewers. As already mentioned, we invited 10 human subjects and asked them to select the top 3 documents they felt were the original patent. We call these the user's first, second, and third choices in their effort to find the real document. We also asked them to rank their choices based on the subject's confidence of the document being original. The r-th choice is correct if it is the original document—otherwise the users were deceived and chose a fake document, thinking it is the real one. We tested $r = 1, 2, 3$ as well as a case we call top-3 in which the user's choice was deemed correct if any one of his top 3 choices was the real document.

Because there were 10 original documents in our corpus, this meant that our subjects had to look through a total of 540 documents which is a very large task for humans. Therefore, we did Step 1 experiment on just one patent.

In order to identify the 3 best parameter settings, we compared the 18 settings in terms of their *Deception-Lift* ($DLIFT$), which is defined as follows.

The $DLIFT$ of a parameter setting can be computed w.r.t. the first, second, and third choice, as well as the top-3 choices. For a parameter setting and the r-th choice ($r \in \{1, 2, 3, \text{top-3}\}$), we first compute the probability that the fake patents generated by this parameter setting are selected by the user as the r-th choice *in experiments*. The $DLIFT$ of that parameter setting is obtained via dividing it by the probability that the fake patents generated by this parameter setting are selected as the r-th choice *in a random choice selection*. A $DLIFT$ greater than 1 means the deception is working. The higher the $DLIFT$ value, the better the deception performance of the corresponding parameter setting. We want to choose the best 3 parameter settings with the highest $DLIFT$ value from Step 1 experiment, and use them for the next step.

TABLE 1: Deception-Lift across 10 participants in *Step 1* for the best 3 parameter settings.

	$DLIFT$ (Step 1)			
	1st choice	2nd choice	3rd choice	Top-3
Best	3.6	1.8	1.8	2.5
Second-best	1.8	3.8	1.8	2.5
Third-best	1.8	3.8	1.8	2.5

Step 2. The aim of the second step is to assess how our approach can prevent discovery of the original documents using the 3 best parameter settings identified previously in Step 1. Specifically, for each of the 10 original documents, we generated 3 fake documents with each of the 3 best parameter settings. In total, there are 9 fake versions of each original document. We invited 10 human subjects to review the 10 versions (9 fakes and 1 original) of each of the 10 original documents. They were again asked to select the top 3 documents they felt were the original document, and rank them based on subject's confidence of the document being original.

We define a metric called *Deception-Rate* (DR) to evaluate how our proposed approach can deceive human subjects. For each human subject h and each original document d, $DR(h, d, r)$ is the probability that a fake document is selected as the r-th choice by human subject h. Furthermore, for each r-th choice, we can compute the average DR of each subject (resp. each document) over all documents (resp. all subjects), i.e., $DR_h^r = \sum_d DR(h, d, r)/10$ (resp. $DR_d^r = \sum_h DR(h, d, r)/10$).

We analyze the distribution of DR_h^r and DR_d^r in order to quantify how well GREEDYFAKEPLGS achieves the goal of deception w.r.t. different human subjects and documents.

6.3 Evaluation Results

Step 1. In total, fake documents generated by 4 of the 18 parameter settings were never selected as the top-3 choices by participants, and hence all their $DLIFT$ values are 0. We ranked parameter settings according to their top-3 Deception-Lift. If two parameter settings have the same top-3 $DLIFT$, we rank them according to their first, second and third choice $DLIFT$ respectively.

Table 1 shows the $DLIFT$ values for the best 3 parameter settings. They achieve the same performance over top-3 choices, but the best one has a higher $DLIFT$ w.r.t. the first choice. The second and third best parameter settings actually have identical performance. While $DLIFT = 1$ means the performance is the same as random selection, we can conclude that all of the 3 parameter settings perform much better in deceiving a user (i.e., the individual committing IP theft) than with random selection. In fact, some of the parameter settings are almost 4 times better at successful deception of the adversary than random choice.

Step 2. As already mentioned, the three best parameter settings reported in Table 1 are used for the second step.

TABLE 2: Deception-Rate in *Step 2* across the human subjects and the set of original documents.

DR (Step 2)							
DR_h^r across H mean(std.)				DR_d^r across D mean(std.)			
1st choice	2nd choice	3rd choice	Top-3	1st choice	2nd choice	3rd choice	Top-3
96% (0.05)	95% (0.08)	93% (0.07)	84% (0.16)	96% (0.05)	95% (0.05)	93% (0.05)	84% (0.07)

To see how the PLG approach can deceive attackers and prevent them from identifying the original document, we computed the deception rates DR_h^r and DR_d^r for the first, second, and third choice as well as the top-3 choices. In particular, we computed their mean and standard deviation respectively across the human subjects H and across the set of original documents D. The results are reported in Table 2. For instance, consider the case across H. Our PLG-based approach can fool the attackers very well:

1) In 96% of the cases (with a standard deviation of 0.05), the document that users select as the top choice as being real is in fact fake, suggesting that our PLG approach is successful at deceiving users.
2) Likewise, in 95% (resp. 93%) of the cases—with a standard deviation of 0.08 (resp. 0.07), the document they select as the second (resp. third) choice is also fake, again demonstrating that our approach has strong deception ability.
3) When we look at the Top-3 column in Table 2, we see that even if we assume the human subject is correct when any of his top 3 guesses is right, the PLG algorithm is able to deceive him 84% of the time (0.16 standard deviation).

In addition, we can see that the standard deviation across the documents D is lower than that across the human subjects H, which suggests that our PLG-based approach achieves similar performance in achieving deception on different documents.

Discussion. An interesting point is that if the adversary was making a *random guess*, then the deception rate for the "Top-3" choice would be 70% as opposed to the 84% we are seeing, which suggests that it might be better for the adversary to guess randomly. The results suggest that the adversay is *not* guessing randomly—rather they appear to be using certain cues so that their 3 top guesses are correlated rather than independent which is why the deception rate actually exceeds 70%. This suggests either that our deception is working very well or that the adversary's guesses are correlated. They make a mistake in their first guess and then make the same kind of mistake in the second and third choices, suggesting that there is some inherent psychological logic that guides their choice of which documents are real. We will examine this in future work.

7 Related Work

Cyber-deception has been extensively used in different contexts during the past several years [18]. For example, in the digital music industry, fake recordings containing terrible music have been generated in order to dissuade users from downloading unauthorized songs [13].

Deception has also been studied for files/documents [24], [21], [8], software code [14], and at the systems level [10], [9], [22]. A honey-pot scheme is proposed by [24], which distributed decoy honey files throughout the system so that system security officers are alerted as soon as an attacker breaks into the system and accesses a honey file (e.g., password.txt). [21] develop an automated system to translate the text into another language. Meanwhile, untranslatable but enticing nouns (such as company names, hot topics, and bogus login information) are sprinkled throughout the text increasing the probability that the attacker will try to steal the file. [14] 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 [22].

The closest work to this paper is that of [8] which is the first paper to propose the use of fake document generation as a way of mitigating IP theft. Their FORGE system uses meta-centrality metrics to identify key features to be replaced in the text of the document. However, because it is text only, their FORGE is not capable of making changes consistently across the different types of components (text, equations, graphs, diagrams/figures) which may be present in a technical document. This is why we introduce probabilistic logic graphs and show that they can represent these diverse types of components. Because the techniques used in this paper manipulate and reason with PLGs in order to generate fake documents, they are very different and much broader than the work in [8].

Meanwhile, there are also works on deception detection for the purpose of detecting fraud on document, code or text [1], [7], [20]. For instance, [1] exploits linguistic features of written documents to detect stylistic deception and distinguish deceptive documents from original ones. [7] investigate machine learning methods to de-anonymize source code authors of C/C++ using coding style. [20] presents a survey of fake news detection from a data mining perspective.

All of the above works differ from the work reported in this paper because they do not uniformly treat the multimodal nature of technical documents. Today, technical documents contain not just text, but diagrams, charts, equations, tables, and more. These different components of a technical document are usually linked together. For instance, a figure may contain some phrases or terms that are used elsewhere in the document. For a fake document to be believable, concepts/terms replaced in one component of a document must be consistently replaced in other components. None of the works above can deal with heterogeneous information of documents consisting of charts, equations, tables, etc., which we can model with PLGs and treat in a single, unified, theoretical framework.

There has also been work on understanding and reasoning about diagrams. [16] consider geometry questions, that is, textual descriptions accompanied by diagrams, and address the problem of diagram understanding, which consists of identifying visual elements in the diagram, their locations, their geometric properties, and aligning them to corresponding textual descriptions. A step further is made in [17] by trying to solve SAT geometry questions. [12]) identify the structure of a diagram and the semantics of its constituents and their relationships.

Reasoning with diagrams has been addressed in [3], [4], [2], [5]. Specifically, diagrams are seen as two-dimensional grids of "tesserae" taking values from a color scale. Operators for combing diagrams are proposed, which essentially perform a tesserae-wise combination of the color values of diagrams.

[23] address the problem of understanding plant diagrams by leveraging both diagrams and explanation text accompanying them. Thus, the goal is similar to [16], but in a different domain.

[11] present a system supporting the interactive construction of diagrammatic proofs, which are used to automatically derive a generalized proof.

There are several differences between this paper and the aforementioned ones. First of all, none of them address the problem of generating fake documents or fake diagrams. Second, most of the works above target specific domains (e.g., geometry, plant domain) and none of them propose a unifying formalism to express different kinds of diagrams in a variety of application domains as well as uncertainty, like we do.

8 Conclusion

The use of cyber-attacks for stealing the intellectual property of companies is growing and is now a frequent news item in major news media [15], [19]. Furthermore, IP theft is often executed as an "Advanced Persistent Threat" or APT attack in which the adversary penetrates an enterprise network and then slowly exfiltrates data over a long period in order to avoid detection. A Symantec study [6] reports that zero-day attacks are detected on average only in 312 days, a very long time for the attacker to steal valuable IP from the victim.

Past work [24], [14], [21] suggested the placement of decoy documents that entice a reader to access them—when accessed, system managers are notified. The goal is detection of an adversary. In contrast, [8] was one of the first efforts to suggest the automatic generation of fake documents so that when the adversary *steals* the documents, he faces additional costs, namely the cost of identifying which documents he stole are real and which ones are fake. This imposes additional financial costs and frustration on the adversary and also gives the victim organization more time to take appropriate action. Our paper continues this line of work by recognizing that past work only modifies the textual part of a document without considering the fact that equations, diagrams/figures, flowcharts, graphs and chart can all be parts of the document and that they must be consistently modified.

In this paper, we have first proposed the new concept of Probabilistic Logic Graphs (PLGs) and showed that PLGs provide a single unifying framework to represent the diversity of content (e.g., charts, equations, formulas, tables, and many others) present in technical documents. In addition, PLGs are capable of representing the uncertainty when processes such as OCR (Optical Character Recognition) are used to automatically construct PLGs from documents. Besides OCR tools, many other tools to automatically extract content from documents introduce uncertainty—e.g., virtually every classifier available in packages like `Scikit-Learn` provides a probability of an object being classified belonging to a class. As a second contribution, we have formalized the problem of generating a given number of fake documents from text and show that the problem is NP-hard. Third, because of the intractable nature of the fake document generation problem, we have developed a practical greedy algorithm and analyzed its complexity. Fourth, we have proved the effectiveness of our proposed approach of employing PLGs to generate fake documents, showing that it achieves high level of deception with experimental results using a panel of 20 human subjects.

Acknowledgments

Part of this work was funded by ONR Grants N00014-16-1-2896 and N00014-15-1-2007. We thank the anonymous referees for many useful comments that helped improve the paper significantly.

References

[1] S. Afroz, M. Brennan, and R. Greenstadt. Detecting hoaxes, frauds, and deception in writing style online. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 461–475, 2012.

[2] M. Anderson. Towards diagram processing: A diagrammatic information system. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 394–401, 1999.

[3] M. Anderson and R. McCartney. Inter-diagrammatic reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 878–884, 1995.

[4] M. Anderson and R. McCartney. Diagrammatic reasoning and cases. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1004–1009, 1996.

[5] M. Anderson and R. McCartney. Diagram processing: Computing with diagrams. *Artificial Intelligence*, 145(1-2):181–226, 2003.

[6] L. Bilge and T. Dumitraş. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 833–844. ACM, 2012.

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

[8] T. Chakraborty, S. Jajodia, J. Katz, A. Picariello, G. Sperli, and V. Subrahmanian. Forge: A fake online repository generation engine for cyber deception. *IEEE Transactions on Dependable and Secure Computing*, 2019.

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

[10] S. Jajodia, V. Subrahmanian, V. Swarup, and C. Wang. *Cyber deception*. Springer, 2016.

[11] M. Jamnik, A. Bundy, and I. Green. Automation of diagrammatic reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 528–533, 1997.

[12] A. Kembhavi, M. Salvato, E. Kolve, M. J. Seo, H. Hajishirzi, and A. Farhadi. A diagram is worth a dozen images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 235–251, 2016.

[13] D. Kushner. Digital decoys [fake mp3 song files to deter music pirating]. *IEEE Spectrum*, 40(5):27, 2003.

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

[15] E. Rosenbaum. 1 in 5 corporations say China has stolen their IP within the last year: CNBC CFO survey. In *CNBC*, Mar 1 2019.
[16] M. J. Seo, H. Hajishirzi, A. Farhadi, and O. Etzioni. Diagram understanding in geometry questions. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2831–2838, 2014.
[17] M. J. Seo, H. Hajishirzi, A. Farhadi, O. Etzioni, and C. Malcolm. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1466–1476, 2015.
[18] A. Shabtai, Y. Elovici, and L. Rokach. *A Survey of Data Leakage Detection and Prevention Solutions*. Springer Briefs in Computer Science. Springer, 2012.
[19] T. Shields. Chasing China theft, U.S. uncovers bonuses for stolen data. In *Bloomberg News*, Feb 27 2019.
[20] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.
[21] J. Voris, N. Boggs, and S. J. Stolfo. Lost in translation: Improving decoy documents via automated translation. In *Proceedings of IEEE Symposium on Security and Privacy Workshops*, pages 129–133, 2012.
[22] W. Wang, J. Bickford, I. Murynets, R. Subbaraman, A. G. Forte, and G. Singaraju. Catching the wily hacker: A multilayer deception system. In *IEEE Sarnoff Symposium*, pages 1–6. IEEE, 2012.
[23] Y. Watanabe and M. Nagao. Diagram understanding using integration of layout information and textual information. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1998.
[24] J. Yuill, M. Zappe, D. Denning, and F. Feer. Honeyfiles: deceptive files for intrusion detection. In *Proceedings of IEEE SMC Information Assurance Workshop*, pages 116–122, 2004.

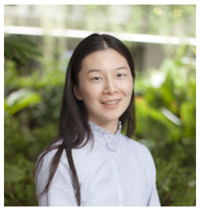
Qian Han is a third-year Ph.D. student at Dartmouth College advised by Prof. V.S. Subrahmanian. He received a BEng in department of Electronic Engineering from Tsinghua University in 2016. During 2015, he spent 3 months as a visiting research assistant at Nanyang Technological University, Singapore. His research interests lie in cybersecurity, data-mining, game theory, and social network analysis.

Cristian Molinaro received the PhD degree in Computer Science Engineering from the University of Calabria, Italy. He was a Visiting Scholar at the State University of New York at Buffalo (2008) and at George Mason University (2012). He was a Faculty Research Assistant at the University of Maryland Institute for Advanced Computer Studies (2009-2011). Currently, he is an Assistant Professor at the University of Calabria. His research interests include database theory, logic programming, and social network analysis.

Antonio Picariello is a Full Professor at Department of Electrical Engineering and Information Technology, University of Naples Federico II. He got a ph. d. in Computer Engineering at the University of Napoli Federico II. He is the director of the National Lab of Computer Science, Telematics and Multimedia (ITEM) of the Italian Consortium on Computer Science and Engineering (CINI). He works in the field of Multimedia Database and Multimedia Information Systems, Multimedia Ontology and Semantic Web, Natural Language Processing, Big Data, Big Data analytics and Social Networks Analysis.

Giancarlo Sperlí is a Researcher at the Consorzio Interuniversitario per l'Informatica (CINI). He hold a Master's Degree and a Bachelor's Degree in Computer Science and Engineering, both from the University of Naples Federico II and in 2018 he received the Ph.D degree in Information Technology and Electrical Engineering of University of Naples ”Federico II”. His main research interests are in the area of Cybersecurity, Semantic Analysis of Multimedia Data and Social Networks Analysis.

V.S. Subrahmanian is the Dartmouth College Distinguished Professor in Cybersecurity, Technology, and Society and Director of the Institute for Security, Technology, and Society at Dartmouth. He previously served as a Professor of Computer Science at the University of Maryland from 1989-2017 where he created and headed both the Lab for Computational Cultural Dynamics and the Center for Digital International Government. He also served for 6+ years as Director of the University of Maryland's Institute for Advanced Computer Studies. Prof. Subrahmanian is an expert on big data analytics including methods to analyze text/geospatial/relational/social network data, learn behavioral models from the data, forecast actions, and influence behaviors with applications to cybersecurity and counterterrorism. He has written five books, edited ten, and published over 300 refereed articles. He is a Fellow of the American Association for the Advancement of Science and the Association for the Advancement of Artificial Intelligence and received numerous other honors and awards. His work has been featured in numerous outlets such as the Baltimore Sun, the Economist, Science, Nature, the Washington Post, American Public Media. He serves on the editorial boards of numerous journals including Science, the Board of Directors of the Development Gateway Foundation (set up by the World Bank), SentiMetrix, Inc., and on the Research Advisory Board of Tata Consultancy Services. He previously served on DARPA's Executive Advisory Council on Advanced Logistics and as an ad-hoc member of the US Air Force Science Advisory Board.
Homepage: http://home.cs.dartmouth.edu/ vs/



Yanhai Xiong is a Postdoc working in Dartmouth College since July, 2018. She received her PhD degree in Computer Science and Engineering from Nanyang Technological University, Singapore and the Bachelor degree in Automation from University of Science and Technological University of China. Her research interests lie in optimization, machine learning, cybersecurity and smart cities.

APPENDIX

We report excerpts taken from three real patents used in our experimental evaluation. For each excerpt, we show two corresponding fakes generated by our framework.

https://patents.google.com/patent/US6078938 **(Original)**

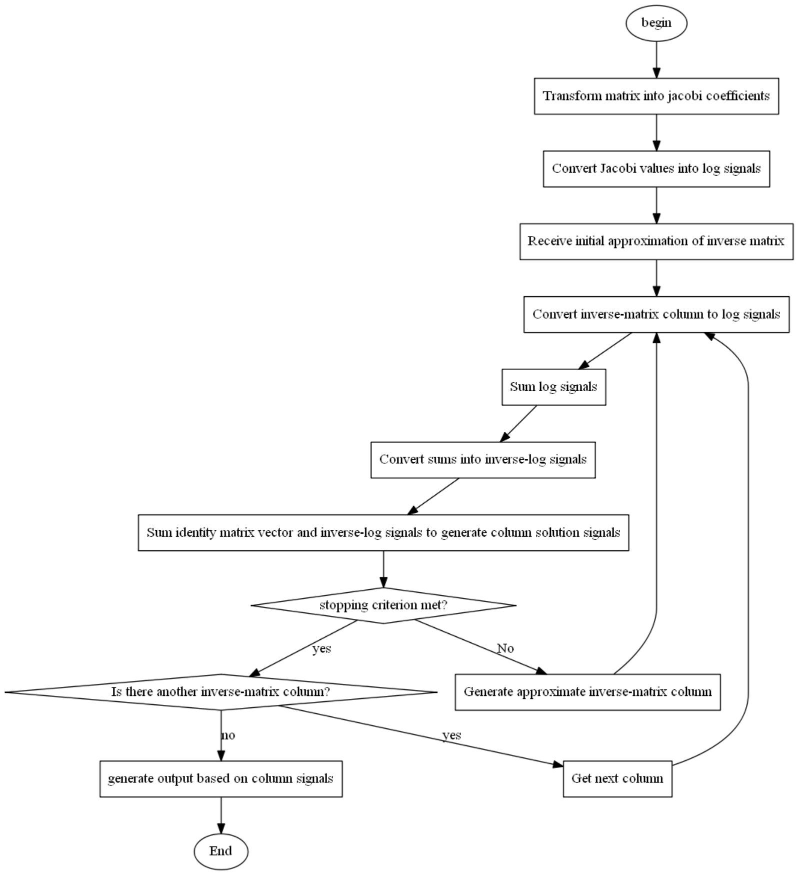

begin
Transform matrix into jacobi coefficients
Convert Jacobi values into log signals
Receive initial approximation of inverse matrix
Convert inverse-matrix column to log signals
Sum log signals
Convert sums into inverse-log signals
Sum identity matrix vector and inverse-log signals to generate column solution signals
stopping criterion met?
yes
No
Is there another inverse-matrix column?
Generate approximate inverse-matrix column
no
yes
generate output based on column signals
Get next column
End

FIG. 6

FIG. 6 shows a flow diagram of a method of inverting a matrix using the computer processor.

[.................]

With logarithmic arithmetic, each of the multiplication operations is accomplished using addition, rather than multiplication. For the purpose of the present disclosure, logarithmic arithmetic is defined as the process of converting operand signals into log signals, summing the log signals to produce a sum, and then converting the sum into an inverse log Signal. This process is used in lieu of a multiplication operation. Logarithmic arithmetic provides advantage because a digital circuit for executing the iterative technique requires substantially less power and space than a circuit employing conventional multipliers.
The iterative technique of Solving linear Systems is based on either the Jacobi iterative technique or the Gauss-Seidel iterative technique. Preferably, the Jacobi iterative technique is used. The Jacobi iterative technique consists of solving the ith equation in AX=b for X, to obtain (provided $a_{ii} \neq 0$):

$$x_i = \sum_{j=1}^{n} (-a_{ij} x_j / a_{ii}) + b_i / a_{ii}$$

and generating each $x_i^{(k)}$ from components of $x^{(k-1)}$ for $k \geq 1$ by:

$$x_i^k = \sum_{j=1}^{n} (-a_{ij} x_j^{(k-1)} / a_{ii}) + b_i / a_{ii}$$

[.................]

https://patents.google.com/patent/US6078938 **(Fake 1)**

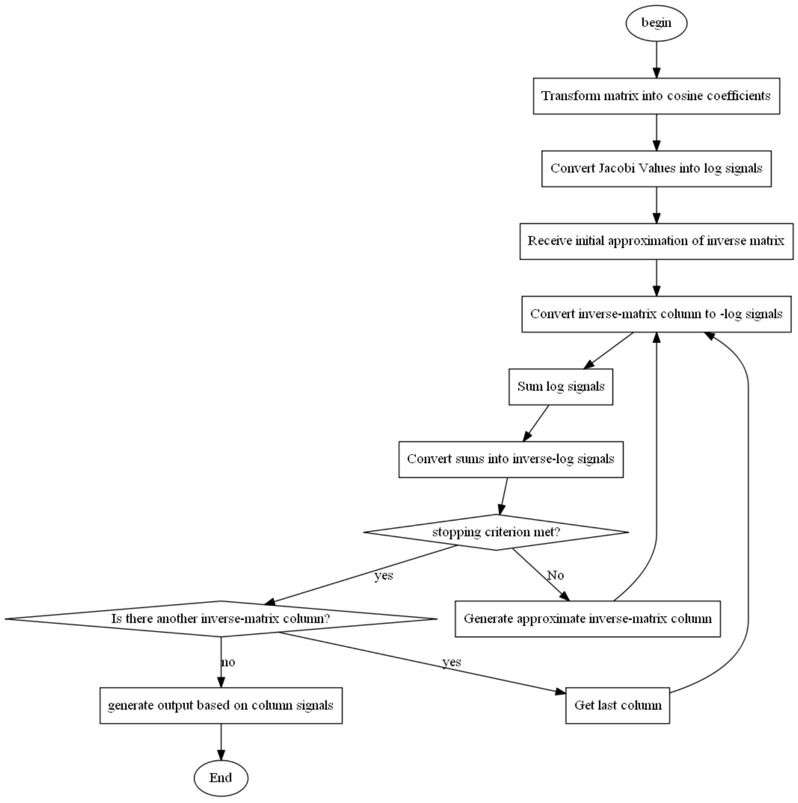

begin
Transform matrix into cosine coefficients
Convert Jacobi Values into log signals
Receive initial approximation of inverse matrix
Convert inverse-matrix column to -log signals
Sum log signals
Convert sums into inverse-log signals
stopping criterion met?
yes
No
Is there another inverse-matrix column?
Generate approximate inverse-matrix column
no
yes
generate output based on column signals
Get last column
End

FIG. 6

FIG. 6 shows a flow diagram of a method of inverting a matrix using the computer processor.

[................]

With logarithmic arithmetic, each of the multiplication operations is accomplished using addition, rather than multiplication. For the purpose of the present disclosure, logarithmic arithmetic is defined as the process of converting operand signals into log signals, summing the log signals to produce a sum, and then converting the sum into an inverse log Signal. This process is used in lieu of a multiplication operation. Logarithmic arithmetic provides advantage because a digital circuit for executing the iterative technique requires substantially less power and space than a circuit employing conventional multipliers.

The iterative technique of Solving linear Systems is based on either the Jacobi iterative technique or the Gauss-Seidel iterative technique. Preferably, the Jacobi iterative technique is used. The Jacobi iterative technique consists of solving the ith equation in AX=b for X, to obtain (provided $a_{ii} \neq 0$):

$$x_i = \sum_{j=1}^{i-1}(-a_{ij}x_j/a_{ii}) + \sum_{j=i}^{n}(a_{ij}x_j/a_{ii}) + b_i/a_{ii}$$

and generating each $x_i^{(k)}$ from components of $x^{(k-1)}$ for $k \geq 1$ by:

$$x_i^k = -\sum_{j=1}^{i-1}(-a_{ij}x_j^{(k-1)}/a_{ii}) - \sum_{j=i}^{n}(a_{ij}x_j^{(k-1)}/a_{ii}) + b_i/a_{ii}$$

[................]

https://patents.google.com/patent/US6078938 **(Fake 2)**

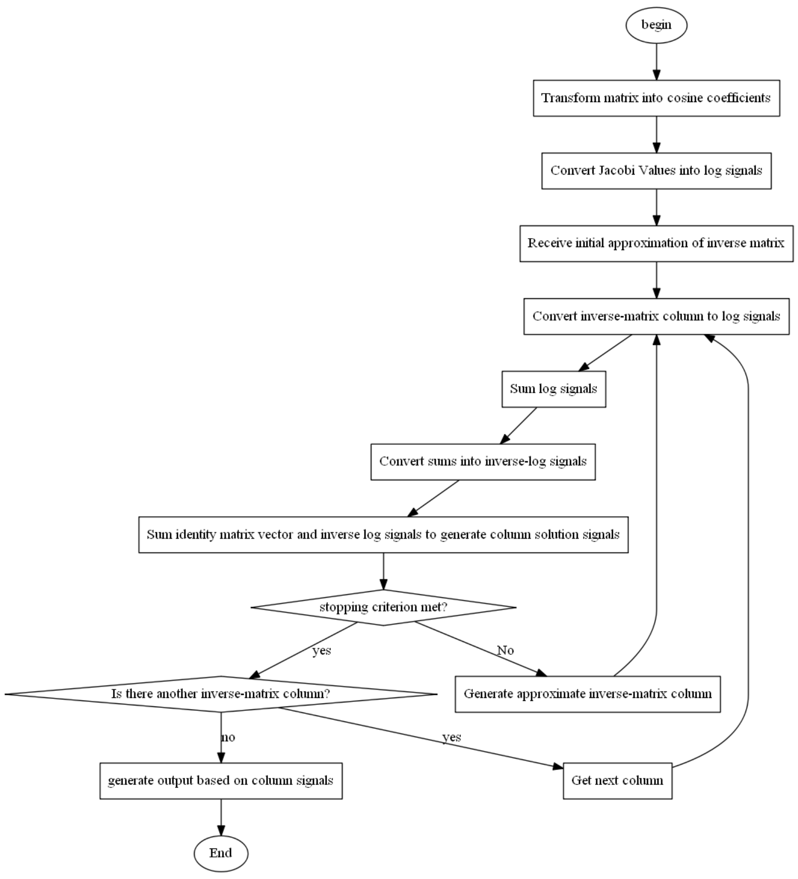

begin
Transform matrix into cosine coefficients
Convert Jacobi Values into log signals
Receive initial approximation of inverse matrix
Convert inverse-matrix column to log signals
Sum log signals
Convert sums into inverse-log signals
Sum identity matrix vector and inverse log signals to generate column solution signals
stopping criterion met?
yes
No
Is there another inverse-matrix column?
Generate approximate inverse-matrix column
no
yes
generate output based on column signals
Get next column
End

FIG. 6

FIG. 6 shows a flow diagram of a method of inverting a matrix using the computer processor.

[.................]

With logarithmic arithmetic, each of the multiplication operations is accomplished using addition, rather than multiplication. For the purpose of the present disclosure, logarithmic arithmetic is defined as the process of converting operand signals into log signals, summing the log signals to produce a sum, and then converting the sum into an inverse log Signal. This process is used in lieu of a multiplication operation. Logarithmic arithmetic provides advantage because a digital circuit for executing the iterative technique requires substantially less power and space than a circuit employing conventional multipliers.
The iterative technique of solving linear systems is based on either the Jacobi iterative technique or the Gauss-Seidel iterative technique. Preferably, the Jacobi iterative technique is used. The Jacobi iterative technique consists of solving the ith equation in AX=b for X, to obtain (provided $a_{ii} \neq 0$):

$$x_i = \sum_{j=1}^{n} (a_{ij} x_j / a_{ii}) + b_i$$

and generating each $x_i^{(k)}$ from components of $x^{(k-1)}$ for $k \geq 1$ by:

$$x_i^k = \sum_{j=1}^{i-1} (a_{ij} x_j^{(k-1)} / a_{ii}) + b_i$$

[.................]

https://patents.google.com/patent/US9523723B2/en **(Original)**

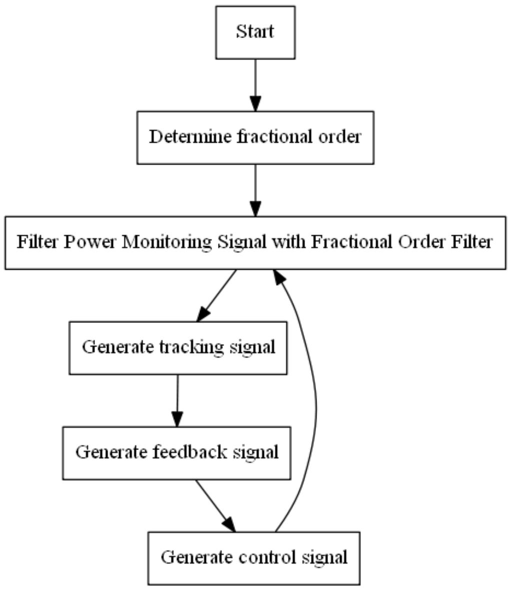

Start
Determine fractional order
Filter Power Monitoring Signal with Fractional Order Filter
Generate tracking signal
Generate feedback signal
Generate control signal

FIG. 7

FIG. 7 is a schematic flow chart diagram illustrating one embodiment of maximum power point tracking method 520. The method 520 may be performed by the system 100 and MPPT controller 150. The method 520 may be performed by analog hardware embedded in the MPPT controller 150. Alternatively, the method 520 may be performed by a computer 400 embedded in the MPPT controller 150. In one embodiment, the method 500 is performed by combinations of analog hardware and the computer 400.

The method 520 starts, and in one embodiment the fractional order apparatus 450 determines 522 the fractional order of the tracking fractional order filter 310, the low pass fractional order filter 314, and/or the integrator fractional order filter 320. The fractional order apparatus 450 may employ the method 500 of FIG. 6 to determine the stable coefficients and fractional order(s). In one embodiment, the fractional order apparatus 450 dynamically determines 522 each fractional order. The fractional order may be reduced to increase a response for a fractional order filter. In addition, the fractional order may be increased to reduce the response for the fractional order filter.

In an alternative embodiment, the fractional orders of the tracking fractional order filter 310, the low pass fractional order filter 314, and/or the integrator fractional order filter 320 are fixed. For example, the fractional orders may be initialized to specified values when the MPPT controller 150 is initialized. The tracking fractional order filter 310 of the demodulation module 210 may filter 524 the power monitoring signal 142 to generate the filtered power monitoring signal 312.

In one embodiment, the demodulation module 210 generates 526 the tracking signal 212 tracking the power point 188 from the filtered power monitoring signal 312. The demodulator 315 may demodulate the filtered power monitoring signal 312 with the perturbation signal 333 to generate the tracking signal 212. The demodulation module 210 may further filter the demodulated power monitoring signal 317 from the demodulator 315 with the low pass fractional order filter 314 to generate the tracking signal 212.

[.................]

In one embodiment, the tracking fractional order filter 310 is a high pass Bode Ideal Cutoff (BICO) filter. The frequency domain response H(s) of the BICO filter may be calculated using equation 3, where ω_0 is frequency of the output power 122, s is frequency, K is a constant, and $0 < q \leq 1$.

$$H(s) = \frac{K}{(\frac{s}{\omega_0} + ((\frac{s}{\omega_0})^2 + 1)^{\frac{1}{2}})^q}$$

[.................]

https://patents.google.com/patent/US9523723B2/en **(Fake 1)**

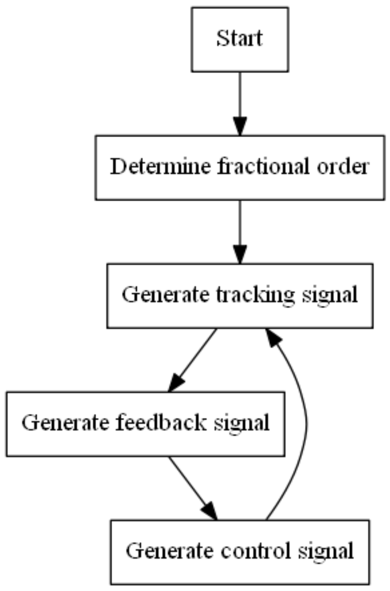


FIG. 7

FIG. 7 is a schematic flow chart diagram illustrating one embodiment of maximum power point tracking method 520. The method 520 may be performed by the system 100 and MPPT controller 150. The method 520 may be performed by analog hardware embedded in the MPPT controller 150. Alternatively, the method 520 may be performed by a computer 400 embedded in the MPPT controller 150. In one embodiment, the method 500 is performed by combinations of analog hardware and the computer 400.

The method 520 starts, and in one embodiment the fractional order apparatus 450 determines 522 the fractional order of the tracking fractional order filter 310, the low pass fractional order filter 314, and/or the integrator fractional order filter 320. The fractional order apparatus 450 may employ the method 500 of FIG. 6 to determine the stable coefficients and fractional order(s). In one embodiment, the fractional order apparatus 450 dynamically determines 522 each fractional order. The fractional order may be reduced to increase a response for a fractional order filter. In addition, the fractional order may be increased to reduce the response for the fractional order filter.

In an alternative embodiment, the fractional orders of the tracking fractional order filter 310, the low pass fractional order filter 314, and/or the integrator fractional order filter 320 are fixed. For example, the fractional orders may be initialized to specified values when the MPPT controller 150 is initialized. The tracking fractional order filter 310 of the demodulation module 210 may filter 524 the power monitoring signal 142 to generate the filtered power monitoring signal 312.

In one embodiment, the demodulation module 210 generates 526 the tracking signal 212 tracking the power point 188 from the filtered power monitoring signal 312. The demodulator 315 may demodulate the filtered power monitoring signal 312 with the perturbation signal 333 to generate the tracking signal 212. The demodulation module 210 may further filter the demodulated power monitoring signal 317 from the demodulator 315 with the low pass fractional order filter 314 to generate the tracking signal 212.

[.................]

In one embodiment, the tracking fractional order filter 310 is a high pass Bode Ideal Cutoff (BICO) filter. The frequency domain response H(s) of the BICO filter may be calculated using equation 3, where ω_0 is frequency of the output power 122, s is frequency, K is a constant, and $0 < q \leq 1$.

$$H(s) = \frac{K}{(\frac{s}{\omega_0} + ((\frac{s}{\omega_0})^2 + 1)^2)^q}$$

[.................]

https://patents.google.com/patent/US9523723B2/en **(Fake 2)**

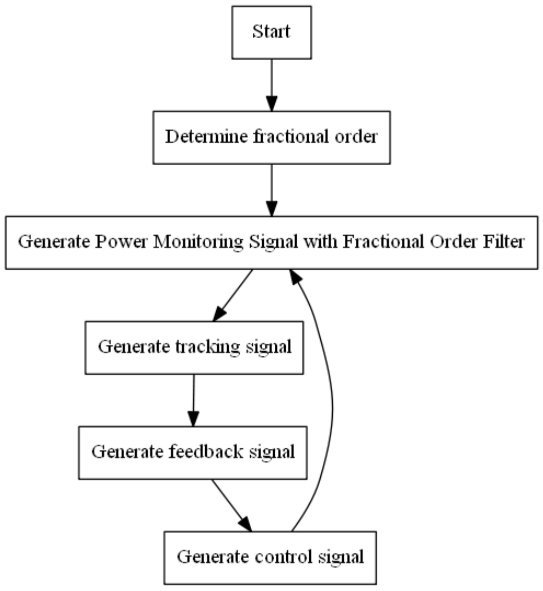

Start
Determine fractional order
Generate Power Monitoring Signal with Fractional Order Filter
Generate tracking signal
Generate feedback signal
Generate control signal

FIG. 7

FIG. 7 is a schematic flow chart diagram illustrating one embodiment of maximum power point tracking method 520. The method 520 may be performed by the system 100 and MPPT controller 150. The method 520 may be performed by analog hardware embedded in the MPPT controller 150. Alternatively, the method 520 may be performed by a computer 400 embedded in the MPPT controller 150. In one embodiment, the method 500 is performed by combinations of analog hardware and the computer 400.

The method 520 starts, and in one embodiment the fractional order apparatus 450 determines 522 the fractional order of the tracking fractional order filter 310, the low pass fractional order filter 314, and/or the integrator fractional order filter 320. The fractional order apparatus 450 may employ the method 500 of FIG. 6 to determine the stable coefficients and fractional order(s). In one embodiment, the fractional order apparatus 450 dynamically determines 522 each fractional order. The fractional order may be reduced to increase a response for a fractional order filter. In addition, the fractional order may be increased to reduce the response for the fractional order filter.

In an alternative embodiment, the fractional orders of the tracking fractional order filter 310, the low pass fractional order filter 314, and/or the integrator fractional order filter 320 are fixed. For example, the fractional orders may be initialized to specified values when the MPPT controller 150 is initialized. The tracking fractional order filter 310 of the demodulation module 210 may filter 524 the power monitoring signal 142 to generate the filtered power monitoring signal 312.

In one embodiment, the demodulation module 210 generates 526 the tracking signal 212 tracking the power point 188 from the filtered power monitoring signal 312. The demodulator 315 may demodulate the filtered power monitoring signal 312 with the perturbation signal 333 to generate the tracking signal 212. The demodulation module 210 may further filter the demodulated power monitoring signal 317 from the demodulator 315 with the low pass fractional order filter 314 to generate the tracking signal 212.

[.................]

In one embodiment, the tracking fractional order filter 310 is a high pass Bode Ideal Cutoff (BICO) filter. The frequency domain response H(s) of the BICO filter may be calculated using equation 3, where ω_0 is frequency of the output power 122, s is frequency, K is a constant, and $0 < q \leq 1$.

$$H(s) = \frac{K}{(s * \omega_0 + ((s * \omega_0)^2 + 1)^2)^q}$$

[.................]

https://patents.google.com/patent/US20180329022A1 **(Original)**

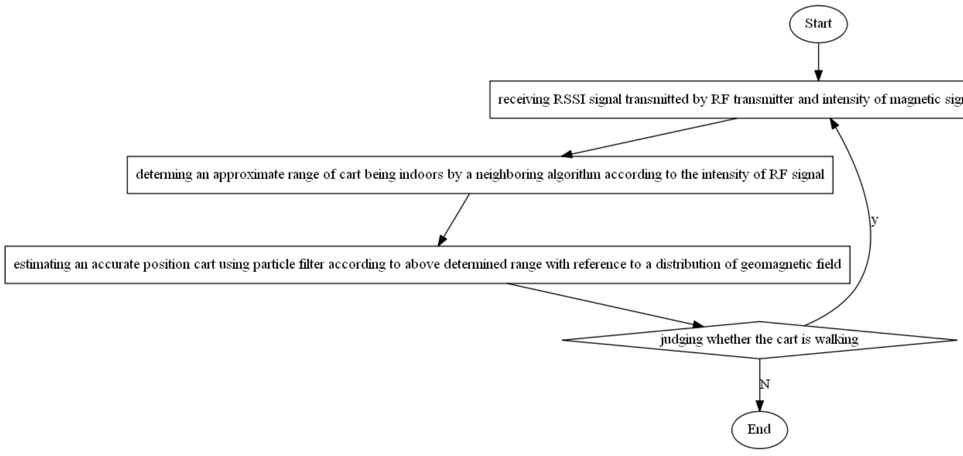

Start
receiving RSSI signal transmitted by RF transmitter and intensity of magnetic signal at the same time by the cart
determing an approximate range of cart being indoors by a neighboring algorithm according to the intensity of RF signal
estimating an accurate position cart using particle filter according to above determined range with reference to a distribution of geomagnetic field
judging whether the cart is walking
y
N
End

FIG. 3

FIG. 3 is a flowchart showing a method for locating an object using cluster-type magnetic field according to another embodiment of the present invention. Taking as an example of locating a robot (i.e., above-mentioned object) of the hand-held cart. As shown in FIG. 3, the locating method may comprise: S301: start (initialization). In this embodiment, the initial particles are uniformly distributed in the room, and the location of a particle is indicated by X_0 (i), the weight of a particle is indicated by $\omega_0(i)(i = 1, 2, \ldots, N_s)$ and the number of particles N_s is 200. In this embodiment, a particle is a certain point of algorithm, and each particle represents a possibility that the object position is within the current range of activity. The cart receives the RSSI signal transmitted by the RF transmitter and intensity of the magnetic field signal at the same time. S303: First, the cart determines its approximate range in the room by means of Nearest Neighbor algorithm according to the intensity of the RSSI signal transmitted by the RF transmitter, i.e., performing a rough locating for the cart. In this embodiment, the approximate range may be, for example, the number of meters around the determined RF transmitting node, the round area centered on the transmitting node or other related areas, or the radiation range of an RF signal and so on. S304: estimating a relative accurate position of the robot of the hand-held cart using particle filter according to above determined range with reference to a distribution of the geomagnetic field within the radiation range of certain RF signal.

[.................]

In this embodiment, after the value of the magnetic field intensity Z, at time t is obtained, the Bayesian criterion is used to update the predicted value of state. The state updating equation is:

$$p(x_t|z_{1:t-1}) = \frac{p(z_t|x_t) * p(x_t|z_{1:t-1})}{\int p(z_{t|x_t}) * p(x_t|z_{1:t-1}) * dx_{t-1}}$$

Here x, indicates the coordinate and position state of moving object at time t (i.e., coordinate point and orientation of the object); z, indicates the value of magnetic field intensity of the object at time t; $p(x_0|y_0)$ indicates an initial distribution function; $p(x_t|z_{1:t})$ indicates an importance density function, and $p(x_t|z_{1:t-1})$ indicates a posterior probability density distribution of the object at time t. Thus, the posterior probability density of the object (i.e., the current location of the object) can be calculated. And this iterative recursive relation constitutes the Bayesian estimation. The particle filtering is based on the law of large numbers using the Monte Carlo algorithm to achieve the integral operation of the Bayesian estimation. Its essence is to approximate the posterior probability density of the object using a random discrete measure composed by the particle positions and their different weights , and to update the random discrete measure by recursion of the algorithm. In this embodiment , the particle filtering algorithm is used to calculate the posteriori probability of the object in the Bayesian estimation.

[.................]

https://patents.google.com/patent/US20180329022A1 **(Fake 1)**

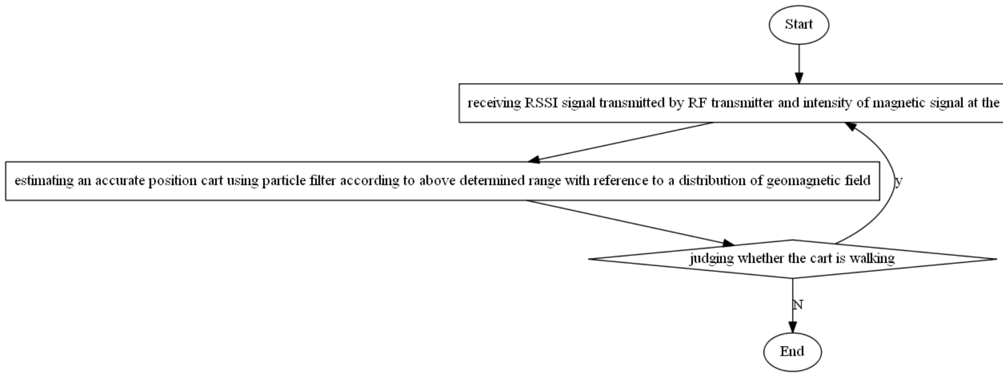

Start
receiving RSSI signal transmitted by RF transmitter and intensity of magnetic signal at the same time by the cart
estimating an accurate position cart using particle filter according to above determined range with reference to a distribution of geomagnetic field
y
judging whether the cart is walking
N
End

FIG. 3

FIG. 3 is a flowchart showing a method for locating an object using cluster-type magnetic field according to another embodiment of the present invention. Taking as an example of locating a robot (i.e., above-mentioned object) of the hand-held cart. As shown in FIG. 3, the locating method may comprise: S301: start (initialization). In this embodiment, the initial particles are uniformly distributed in the room, and the location of a particle is indicated by X_0 (i), the weight of a particle is indicated by $\omega_0(i)(i = 1, 2, \ldots, N_s)$ and the number of particles N_s is 200. In this embodiment, a particle is a certain point of algorithm, and each particle represents a possibility that the object position is within the current range of activity. The cart receives the RSSI signal transmitted by the RF transmitter and intensity of the magnetic field signal at the same time. S303: First, the cart determines its approximate range in the room by means of Nearest Neighbor algorithm according to the intensity of the RSSI signal transmitted by the RF transmitter, i.e., performing a rough locating for the cart. In this embodiment, the approximate range may be, for example, the number of meters around the determined RF transmitting node, the round area centered on the transmitting node or other related areas, or the radiation range of an RF signal and so on. S304: estimating a relative accurate position of the robot of the hand-held cart using particle filter according to above determined range with reference to a distribution of the geomagnetic field within the radiation range of certain RF signal.

[.................]

In this embodiment, after the value of the magnetic field intensity Z, at time t is obtained, the Bayesian criterion is used to update the predicted value of state. The state updating equation is:

$$p(x_t|z_{1:t-1}) = \frac{\int p(z_t|x_t) * p(x_t|z_{1:t-1}) * dx_{t-1}}{p(z_{t|x_t}) * p(x_t|z_{1:t-1})}$$

Here x, indicates the coordinate and position state of moving object at time t (i.e., coordinate point and orientation of the object); z, indicates the value of magnetic field intensity of the object at time t; $p(x_0|y_0)$ indicates an initial distribution function; $p(x_t|z_{1:t})$ indicates an importance density function, and $p(x_t|z_{1:t-1})$ indicates a posterior probability density distribution of the object at time t. Thus, the posterior probability density of the object (i.e., the current location of the object) can be calculated. And this iterative recursive relation constitutes the Bayesian estimation. The particle filtering is based on the law of large numbers using the Monte Carlo algorithm to achieve the integral operation of the Bayesian estimation. Its essence is to approximate the posterior probability density of the object using a random discrete measure composed by the particle positions and their different weights , and to update the random discrete measure by recursion of the algorithm. In this embodiment , the particle filtering algorithm is used to calculate the posteriori probability of the object in the Bayesian estimation.

[.................]

https://patents.google.com/patent/US20180329022A1 **(Fake 2)**

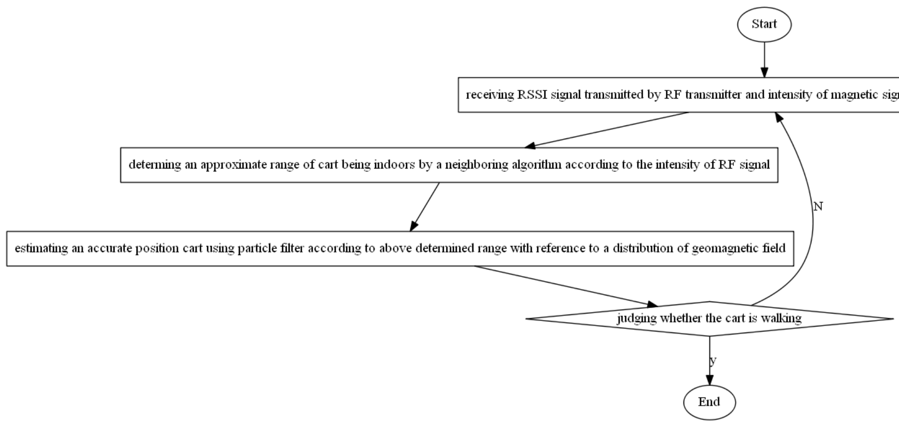

Start
receiving RSSI signal transmitted by RF transmitter and intensity of magnetic signal at the same time by the cart
determing an approximate range of cart being indoors by a neighboring algorithm according to the intensity of RF signal
estimating an accurate position cart using particle filter according to above determined range with reference to a distribution of geomagnetic field
judging whether the cart is walking
N
y
End

FIG. 3

FIG. 3 is a flowchart showing a method for locating an object using cluster-type magnetic field according to another embodiment of the present invention. Taking as an example of locating a robot (i.e., above-mentioned object) of the hand-held cart. As shown in FIG. 3, the locating method may comprise: S301: start (initialization). In this embodiment, the initial particles are uniformly distributed in the room, and the location of a particle is indicated by X_0 (i), the weight of a particle is indicated by $\omega_0(i)(i = 1, 2, \ldots, N_s)$ and the number of particles N_s is 200. In this embodiment, a particle is a certain point of algorithm, and each particle represents a possibility that the object position is within the current range of activity. The cart receives the RSSI signal transmitted by the RF transmitter and intensity of the magnetic field signal at the same time. S303: First, the cart determines its approximate range in the room by means of Nearest Neighbor algorithm according to the intensity of the RSSI signal transmitted by the RF transmitter, i.e., performing a rough locating for the cart. In this embodiment, the approximate range may be, for example, the number of meters around the determined RF transmitting node, the round area centered on the transmitting node or other related areas, or the radiation range of an RF signal and so on. S304: estimating a relative accurate position of the robot of the hand-held cart using particle filter according to above determined range with reference to a distribution of the geomagnetic field within the radiation range of certain RF signal.

[.................]

In this embodiment, after the value of the magnetic field intensity Z, at time t is obtained, the Bayesian criterion is used to update the predicted value of state. The state updating equation is:

$$p(x_t|z_{1:t-1}) = \frac{\frac{p(z_t|x_t)}{p(x_t|z_{1:t-1})}}{\int \frac{p(z_{t|x_t})}{p(x_t|z_{1:t-1})} * dx_{t-1}}$$

Here x, indicates the coordinate and position state of moving object at time t (i.e., coordinate point and orientation of the object); z, indicates the value of magnetic field intensity of the object at time t; $p(x_0|y_0)$ indicates an initial distribution function; $p(x_t|z_{1:t})$ indicates an importance density function, and $p(x_t|z_{1:t-1})$ indicates a posterior probability density distribution of the object at time t. Thus, the posterior probability density of the object (i.e., the current location of the object) can be calculated. And this iterative recursive relation constitutes the Bayesian estimation. The particle filtering is based on the law of large numbers using the Monte Carlo algorithm to achieve the integral operation of the Bayesian estimation. Its essence is to approximate the posterior probability density of the object using a random discrete measure composed by the particle positions and their different weights , and to update the random discrete measure by recursion of the algorithm. In this embodiment , the particle filtering algorithm is used to calculate the posteriori probability of the object in the Bayesian estimation.

[.................]