Provenance Graph Kernel

David Kohan Marzagão, Trung Dong Huynh, Ayah Helal, Sean Baccas, Luc Moreau

Abstract—Provenance is a standardised record that describes how entities, activities, and agents have influenced a piece of data; it is commonly represented as graphs with relevant labels on both their nodes and edges. With the growing adoption of provenance in a wide range of application domains, users are increasingly confronted with an abundance of graph data, which may prove challenging to process. Graph kernels, on the other hand, have been successfully used to efficiently analyse graphs. In this paper, we introduce a novel graph kernel called *provenance kernel*, which is inspired by and tailored for provenance data. We employ provenance kernels to classify provenance graphs from three application domains. Our evaluation shows that they perform well in terms of classification accuracy and yield competitive results when compared against existing graph kernel methods and the provenance network analytics method while more efficient in computing time. Moreover, the provenance types used by provenance kernels are a symbolic representation of a tree pattern which can, in turn, be described using the domain-agnostic vocabulary of provenance. Therefore, provenance types thus allow for the creation of explanations of predictive models built on them.

Index Terms—kernel methods, data provenance, graph classification, provenance analytics, interpretable machine learning

1 Introduction

PROVENANCE is metadata that provides an account of the actions a system performs, describing the application data involved, the flow of such data, the processes carried out over the data, and the people and organisations involved in it. Provenance is increasingly being captured in a variety of application domains, from scientific workflows [1], [2] supporting their reproducibility, to climate science [3], and human-agent teams in disaster response [4]. A data model for Provenance was standardized at the World Wide Web Consortium (W3C), which defined provenance as a *record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing in the world* [5]. Structurally, provenance metadata is a form of knowledge graph [6], that provides an enrichment of application data allowing a wide range of use cases to be supported, including deciding whether the information is to be trusted, how it should be integrated with other diverse information sources, and how to give credit to its originators when it is reused [7]. Provenance metadata is deeply connected to the flow of time, as specified by its temporal interpretation [8]; to be valid, provenance is expected to satisfy some temporal constraints [9]. The distinct nature of application data and provenance metadata should be noted; they are complementary as provenance metadata does not seek to replicate application data, and they support different requirements, and sometimes even require distinct security measures [10]. As the desire to ascertain the authenticity of data surges, a greater number of applications are being "provenance-enabled", resulting in users being confronted with an increasing volume of provenance metadata, especially from automated systems. The volume of provenance metadata makes it a challenge to extract meaning and significance.

The focus of this paper is on the study of techniques to make sense of provenance metadata, subsequently referred to as *provenance graphs*. The need for such techniques is particularly relevant in domains where patterns of behaviours are important and are not readily extractable from actual application data. Thus, our focus is on what happened, in what order, and how often, rather than the details of actual application data values that may result from these processes. In this context, a fundamental problem is the ability to determine whether provenance graphs between two executions are similar; this allows a wide range of use cases to be determined, such as outlier detection and process compliance. In short, the focus of this paper is the study of classification methods for provenance graphs according to their similarity. Furthermore, a classification method can no longer be a black box: in addition to being able to determine the similarity of graphs, there is an expectation that a description of the similarity properties of the retrieved graphs is also provided. Thus, the classification method must also procedurally construct explanations for a given classification decision.

To that end, we introduce two techniques. First, similarly to programming languages, which define types as descriptions of sets of values, [11], we introduce the concept of a *provenance type* as a description of a set of nodes with a common historical context in provenance graphs; a node's historical context is a size-limited symbolic representation of its past. Second, Building on such provenance types, and the count of their occurrences in a provenance graph to create its feature vector, we then introduce a provenance graph kernel method to determine the similarity of two provenance graphs. Provenance types' symbolic representation is an inductively defined structure that lends itself to constructing explanations of the most influential features in

- *D. Kohan Marzagão, T.D. Huynh, and L. Moreau are with Department of Informatics, King's College London, UK.*
- *A. Helal is with Department of Computer Science, Exeter University, UK. She is also affiliated to King's College London.*
- *S. Baccas is with Polysurance.*

This work is supported by a Department of Navy award (Award No. N62909-18-1-2079) issued by the Office of Naval Research. The United States Government has a royalty-free license throughout the world in all copyrightable material contained herein.

a classification decision.

We show that the computational complexity of inferring provenance types up to depth h of the historical context in a graph with m edges is bounded by $\mathcal{O}(h^2 m)$. We employ provenance kernels in classifying provenance graphs in six different data sets from three application domains. We compare the performance of provenance kernels against that of existing graph kernels and other existing algorithms tailored for provenance graphs in those classification tasks, showing that provenance kernels are competitive in terms of accuracy and at the same time as fast in terms of execution times, while procedurally providing explanations for their decisions. Compared to Provenance Network Analytics (PNA) [12], an existing classification method specific to provenance graphs, provenance kernels outperform it both in terms of accuracy and computation time. Relying on a third-party technique to identify the most influential provenance types for a particular classification task, a verbal description is generated computationally to facilitate the explanation of a classification decision.

In summary, the contribution of this paper is threefold:

1) The definition of the novel notion of provenance type (Section 4.2), and
2) The definition, implementation, and evaluation of a novel kernel method for provenance graphs, i.e. provenance kernels, which are shown to perform well in classifying provenance graphs when compared to standard graph kernels and the PNA method (Section 4.3).
3) An illustration of how provenance types can help improve the explainability of classification models built with provenance kernels (Section 6).

Section 2 introduces the definitions of provenance graphs and motivates the problem being studied. The related work is then discussed in Section 3 with a comparison of the theoretical computational complexity between provenance kernels and existing graph kernels. Section 4 introduces the provenance kernel framework, including an efficient algorithm to infer feature vectors from input graphs to be used for their classification. Section 5 presents the empirical evaluation of provenance kernels in six classification tasks in comparison with generic graph kernels and the PNA method. Section 6 then discusses the use of provenance types in conjunction with LIME to explain classification decisions. Finally, Section 7 concludes the paper and outlines the future work.

2 Provenance Graphs and Motivation

In this section, we informally introduce the notion of provenance through an illustrative example. We then provide a formal definition of provenance graphs that allows us to formulate a precise problem statement, which is then tackled in the rest of the paper. Both formal and informal definitions of provenance follow the data model for provenance, called PROV, standardised at the W3C [5].

Consider the problem of analysing in-hospital patient data. For example, in classifying each patient (partial) journey into different scenarios (e.g., whether it may lead to an in-hospital fatality). To that end, one may want to store some of the relations between the patients and, for example, the treatments applied to them, or the wards in which they were treated. Fig. 1 shows how a provenance graph can be used in such a case. In this short example, a patient is admitted to a ward and then moved to another ward for treatment, until they are eventually discharged. Note that, according to the standardised PROV data model [5]. , edges 'point to the past'. For example, the edge labelled 'der' indicates $\texttt{Patient7}_1$ was derived from $\texttt{Patient7}_0$. For reference, Table 1 gives the list of edge labels (which refer to past events) and their meaning.

The three node labels shown at the top of the table are called *PROV generic* labels because they are used irrespective of particular provenance applications. The labels of start and destination nodes connected by edges of each given edge label are also specified in, respectively, the third and

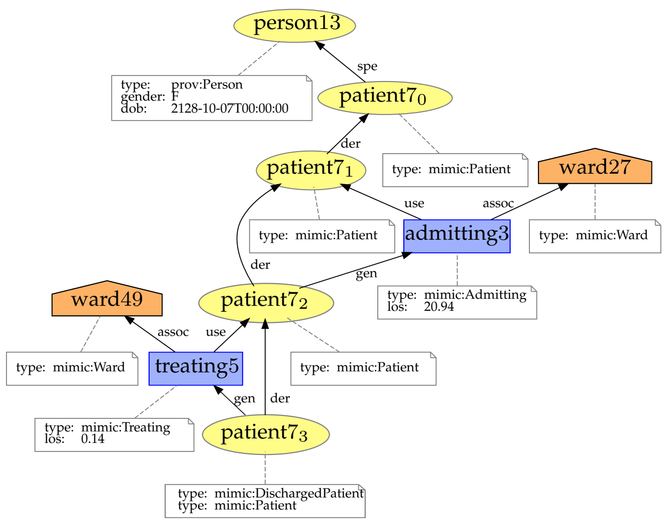

Figure 1: A provenance graph representing a hospital admission, in which a patient is admitted into a hospital ward, and is later treated at another ward. Yellow ellipses denote entities, blue rectangles denote activities, and orange trapeziums denote agents. The same patient is recorded multiple times via different entities to represent their different states over time.

Table 1: PROV generic labels for nodes (first three rows) and edges in a provenance graph, and their notation used throughout this paper. The third and fourth columns show, respectively, the expected labels of source and destination nodes for each edge label.

Label	Notation	Source	Destination
Agent	*ag*	-	-
Activity	*act*	-	-
Entity	*ent*	-	-
wasDerivedFrom	der	*ent*	*ent*
specializationOf	spe	*ent*	*ent*
alternateOf	alt	*ent*	*ent*
wasInvalidatedBy	wib	*ent*	*act*
wasGeneratedBy	gen	*ent*	*act*
used	use	*act*	*ent*
wasAttributedTo	wat	*ent*	*ag*
wasAssociatedWith	assoc	*act*	*ag*
actedOnBehalfOf	abo	*ag*	*ag*
wasStartedBy	wsb	*act*	*ent*
wasEndeddBy	web	*act*	*ent*
wasInformedBy	wifb	*act*	*act*

fourth columns. For a complete description of the PROV data model, refer to [5].

The least descriptive provenance graphs will only contain the labels of Fig. 1 to categorise the nodes and edges of the graph, and no other descriptor. In the spirit of the Semantic Web approach, more descriptive provenance graphs will enrich nodes and edges with type annotations, such as Patient and Ward in Fig. 1 in a hospital context. It is even possible for some provenance graphs to also contain edge or node attributes – however, we do not support such continuous attributes in our model, and expect only categorical features. While this may appear as a restriction to general graph practitioner, this is aligned with the nature of provenance graph, which focus on the topological interconnection of data, rather than application data.

2.1 Provenance Graphs

We denote $G = (V, E, S, L)$ a *provenance graph* in which V corresponds to the set of nodes of G with $|V| = n$, E the set of its directed edges, with $|E| = m$. The sets of labels of nodes and edges of G are denoted, respectively, by S and L. An edge $e \in E$ is a triplet $e = (v, u, l)$, where $v \in V$ is its *starting point*, $u \in V$ is its *ending point*, and $l \in L$, also denoted (when there is need to avoid ambiguity) as $lab(e)$, is the edge's set of *labels*.[1] Note that defining edges as triplets instead of pairs allows provenance graphs to have more than one edge between the same pair of vertices. Each node $v \in V$ can have more than one label, and thus $lab(v) \in \mathbb{P}(S)$, where $\mathbb{P}(S)$ denotes the power-set of S, i.e., the set of subsets of S. Note that provenance graphs are *finite*, *directed*, and *multi-graphs* (as there might exist more than one edge between the same pair of nodes). Note that, in this work, we refer to *labels* as categorical features, as opposed to continuous features (such as time, price, etc).

Considering only PROV generic labels, $S = \{ag, act, ent\}$. In cases where application-specific labels are used, however, set S is enlarged to also include such specific labels. These specific labels can be used, for example, to incorporate information otherwise not present in provenance graphs, e.g., duration of an activity (long, short, etc.), nature of an entity, or details of an agent (see Section 2.1 of the Supplementary Material for concrete examples). Regarding edge-labels, typically $L = \{der, spe, alt, wib, gen, \dots\}$, where the edge $(v, u, assoc)$, for example, indicates that activity v was associated with agent u.

We will often work with more than one graph, and thus we define $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{L})$ as a (finite) family of graphs, in which $\mathcal{V}$, $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{L}$, are the union of the sets of, respectively, nodes, edges, node labels, and edge labels of graphs in $\mathcal{G}$. For an inter-operable specification of provenance graphs, see [5]. For a temporal interpretation of provenance graphs, see [13], [14]. Finally, there are several methodologies for developing software that generates provenance graphs (see [15], [16], [17]); their description, however, is beyond the scope of this paper.

1. We allow for edges to have more than one label, thus consider the set of labels of an edge. We may abuse notation and write edges without brackets when singletons. In the absence of ambiguity, we may also abuse notation and refer to edges as simply pairs (v, u).

2.2 Problem Definition

Comparing, contrasting, and capturing similarities between provenance graphs is key in order to perform classification tasks. To that end, this paper addresses the following question: given a set of provenance graphs $\mathcal{G}$, how to establish the existence of a positive semi-definite kernel function to compare them, both efficiently and effectively, in order to infer classification predictions based on them? Furthermore, how to create explanations for such decisions? In the remainder of the paper, we will introduce provenance types and provenance kernels (Section 4), and show that they satisfy the requirements above.

3 Related Work

The notion of provenance type has been proposed as a tool for provenance graph summarisation [18], [19]. In both these approaches, the idea of the history of provenance nodes being related to a sequence of transformations described by edge labels is used. To the best of our knowledge, however, this is the first work to explore the use of provenance types in the context of machine learning methods. When comparing the efficiency of the other summarisation methods with our own, we find that the definition by [18] requires an exponential time $\mathcal{O}(nd^h)$, where d is the maximum degree of nodes in an input graph, and n its number of nodes. On the other hand, [19] proposed a faster algorithm that takes $\mathcal{O}(hm)$, where m is the number of edges of an input graph. This is faster than our algorithm by a factor of h, which is typically small and does not depend on the size of the graph. This faster algorithm, however, shares similarities with Weisfeiler-Lehman graph kernels to a point in which patterns with very close meaning in provenance are classified differently. More specifically, the algorithm does not inspect sub-trees beyond their first level in order to discard repetitions. This is discussed in detail later in this section and exemplified in Fig. 2.

ML techniques on graphs have been proposed in the domain of provenance. Provenance Network Analytics (PNA) [12], for example, creates, for each graph, a feature vector that encodes a sequence of graph topological properties. Some of these are provenance agnostic, such as the number of nodes, or the number of edges. Others, in contrast, record the longest shortest paths between, e.g., two provenance entities, or between an agent and an activity. In Section 5, we compare the performance of provenance kernels and PNA in the same classification tasks. For other works in provenance and ML see [20] and [21].

Similarities between graphs have often been studied in the context of graph classification and detection of malicious activity, with a plethora of different methods being proposed to that end (for recent surveys on graph kernels, see [22], [23], and [24]). Such methods explore graph properties using concepts such as shortest paths between nodes [25], sub-trees [26], [27], or random walks [28], to mention just a few. In many cases, the graphs being analysed may have continuous or categorical labels for their nodes or edges, but little attention has been given to developing a graph kernel that considers *both* edge and node categorical labels, a key property of provenance graphs.

A promising field of ML algorithms for graph classification is the one of graph neural networks (GNNs) [29], [30], [31], [32]. They also differ in terms of whether they take into account edge or node labels and attributes. Some GNNs would allow for edge weights [33], while other architectures, such as Graph Attention Networks (GAT) [34] take into account edge labels. There are several similarities between GNNs and some graph kernels. The idea of message passing in GNNs is present in several graph kernel methods, including Weisfeiler-Lehman (WL). In fact, [35] has proven that GNNs are as expressive as the WL isomorphism test. Similarities aside, there are a few differences with GNN methods and our approach with PK. For one, the training time of GNNs in our datasets are shown to be several magnitudes higher (see Section 5 of the Supplementary Materials for more details), although it must be noted that training of GNNs cannot be directly compared with training of graph kernels. On the explainability side, whilst there is a lot of work on explaining GNNs without the creation of feature vectors, the explainability strategy presented in this paper is not immediately applicable to GNNs since our explainability approach relies on feature vectors representing substructures of the graphs themselves. See [36] for GNNs applied to knowledge graphs with a focus on missing links and nodes.

Provenance kernels compare and classify graphs, as opposed to comparing and classifying nodes on graphs. A classic example to highlight their difference is the task of classifying a social network as a whole, such as communities, versus the classification of nodes within a social network, such as people that influence that network in a certain way. For the former, one can use graph kernels, whereas, for the latter, one may use what is often known as kernels *on graphs* or *graph embedding* techniques. A notable example of the graph embedding technique is Struc2Vec [37], which is a learning technique based on the structural identity of nodes on graphs that embed nodes into a Euclidean space according to the structure of their neighbourhood. Similarly to provenance kernels, Struc2Vec has a hierarchical approach when looking at the structure of the neighbourhood of nodes. The main difference to our work is that Struc2Vec does not consider edge nor node labels, but instead the number of occurrences of neighbouring elements (and their degrees). Another commonality is that both works define a distance between nodes based on their neighbours and a suitable metric. In the context of knowledge graphs, graph embedding was used for link prediction [38]. Struc2Vec used implicit computations to compare a pair of graphs, while [38] introduced explicit methods allowing for faster computation (for a survey on similar models, see [39]). ML methods for Resource Description Framework (RDF) graphs have been studied by [40] (RDF2Vec), [41] and [42]. Other known approaches that for example aim to find missing labels or links in graphs include NodeSketch [43], DeepWalk [44], and Node2Vec [45].

Using the classification from [24], we can place provenance kernels in the *Information Propagation* graph kernel family, and, more specifically, in the group of *Kernels based on iterative label refinement*. That is because we iteratively update node labels up to h times based on the other node labels. Other graph kernel methods in the same family

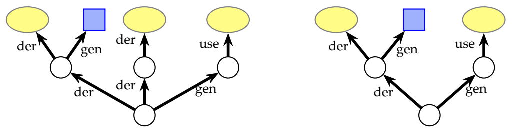

Figure 2: Two examples of different tree-patterns that have a similar meaning in provenance. Yellow circles denote entities, blue squares denote activities, and plain circles denote nodes of which types are not captured by the algorithm. Provenance kernels classify both these tree-patterns as the same 2-type, i.e., $(\{der, gen\}, \{der, gen, use\}, \{act, ent\})$.

include Neighbourhood Hash kernels (NH) [46], Weisfeiler-Lehman kernels (WL) [26], and Neighborhood Subgraph Pairwise Distance kernel (NSPD) [47]. Out of those, only the latter originally took into account the edge labels in addition to node labels in its classification. A possible extension for labelled edges, however, is mentioned in WL original work [26] and made explicit by [48].

On the other hand, the WL graph kernel algorithm presented by [26] does not consider edge labels. A close variant to this WL extension was also proposed by [19], with the differences that (1) repetitions of branches from the same given parent in its walk-tree are discarded and; (2) although all edge-labels were considered, only nodes at the leaves of trees had their labels taken in account. Another close variant of WL, that uses truncated trees as features, was proposed recently in [49]. Discarding repetitions, however, is key in reducing the sizes of feature vectors by putting together patterns that have a similar meaning in provenance. In this paper, we extend this process further by agglutinating even more similar patterns into the same provenance type. Fig. 2 shows two examples of such types. In essence, the extra sequence of two derivations in Pattern 1 does not add qualitative meaning to the set of transformations that lead to the creation of the root node. Thus, provenance kernels consider these two patterns as having the same type. One can see provenance types for provenance kernels as a *flattened* version of the ones defined by [19] since the idea of branches is hidden by the sole enumeration of labels at a given depth.

In terms of theoretical computational complexity, provenance kernels are situated in the efficient spectrum. Note that with provenance kernels, the computational complexity for one graph with m edges, $\mathcal{O}(h^2 m)$, is bounded by $\mathcal{O}(h^2 nd)$, where d is the maximum (out) degree, and n is the number of nodes of this graph. Also, $m = nd$ if and only if the input graph is regular. Graphlet Sampling kernel (GS) [50] counts the number of small subgraphs present in each input graph. These small subgraphs typically have size $k \in \{3, 4, 5\}$. The original time required to run this kernel, $\mathcal{O}(n^k)$, is prohibitively expensive. Later, [51] improved this bound to $\mathcal{O}(nd^{k-1})$, where d is the maximum degree of nodes in an input graph. Neighbourhood Hash kernel (NH) [46] compares graphs by counting the number of common node labels, which are updated with the employment of logical operations that take into account the label of neighbouring nodes. These operations do not use the original categorical node label, but a binary array, so XOR operations can be employed. The complexity of this kernel is bounded by $\mathcal{O}(bhn\bar{D})$, where $\bar{D}$ is the average degree of the vertices, h is how many times the update is executed, and b is the number of bit labels. We need $2^b - 1 \gg |\Sigma|$, the size of the label set. Finally, Neighborhood Subgraph Pairwise Distance kernel (NSPD) [47] compares subgraphs S in the neighbourhood of nodes up to a distance r for all pairs of nodes in a given graph. The complexity, $\mathcal{O}(n|S||E(S)|\log|E(S)|)$, is such that S is the subgraph induced by the neighbourhood of a vertex and radius r. $E(S)$ is the number of edges of S.

We can see that some of the theoretical gaps in running times between provenance kernels and the others discussed above come down to variables that do not depend on a graph's size or topology, such as the number of bit labels or the size of graphlets. Others, such as the maximum degrees and number of edges of neighbourhood subgraphs, are more dependent on input graphs and tend to have more impact on running times, which indicates that provenance kernels are computed faster. These differences are reflected in the empirical running times evaluated in Section 5, in which we show that provenance kernels outperform the three methods described above (GS, NH, and NSPD) in terms of efficiency. Moreover, none of these is designed to take as input the edge categorical labels of graphs. Subgraph Matching kernel, on the other hand, does consider both edge and node labels as it counts the number of common subgraphs of bounded size k between two graphs. It requires, however, an impractical computational running time of $\mathcal{O}(kn^{k-1})$, where n this time stands for the sum of sizes of the two graphs being compared. In fact, this kernel timed out in our experiments and its accuracy could not be measured.

Table 2 presents a summary of all graph kernels to be compared to provenance kernels in Section 5. In this table, we note whether node or edge categorical labels were taken into account in the implementation we used, as well as their theoretical complexity. We now briefly discuss the remaining kernel methods. The Shortest Path kernel (SP) [25], for each graph, constructs a new graph that captures the original graph's shortest paths and then uses a base kernel to compare two shortest path graphs. The complexity of SP is $\mathcal{O}(n^4)$. We use the algorithm of Vertex Histogram (VH) and Edge Histogram (EH) kernels as presented by [52]. VH creates, for each graph, a feature vector that captures the number of nodes with each given node label l, whereas EH does the analogous for edge labels. Their computational complexities are $\mathcal{O}(n)$ for VH and $\mathcal{O}(m)$ for EH. Note that provenance kernels of depth 0 coincide with VH. Hadamard Code kernel (HC) [53] is similar to NH, even by showing the same computational complexity. It explores the neighbourhood of nodes iteratively for different levels (or depths). Its name comes from the use of Hadarmard code matrices.

Finally, when compared against the PNA method, provenance kernels outperform in terms of theoretical computational complexity. One of the features considered in the PNA method is the longest shortest path between two nodes of a given label (two entities, one entity and one activity, and so on). This gives us a lower bound for PNA's computational complexity of $\Omega(nm)$.

4 Provenance Kernel Framework

In this section, we motivate and present a graph kernel method inspired by particular characteristics of provenance graphs, namely, labelled nodes and edges, and their underpinning temporal semantics. We base our definitions on the PROV data model [5], a *de jure* standard for provenance data. We first present some background, including a definition of kernel functions. We then move to our contributions: motivate and formally define provenance types, provide a formal definition of provenance kernel, to finally present an algorithm to infer provenance types, analysing its theoretical computational complexity.

4.1 Preliminaries

We formally introduce graph concepts and kernel functions, to be used in our definition of provenance kernels later in this Section.

The *forward-neighbourhood* of a given node $v \in \mathcal{V}$ is the set of nodes it "points to", i.e., $v^+ = \{u \mid (v,u,l) \in \mathcal{E}\}$. Analogously, the *backward-neighbourhood* of v is denoted by $v^- = \{u \mid (u,v,l) \in \mathcal{E}\}$. We say a node u is *distant from* v by x if there is a *walk* from v to u of length x, where v is the walk's *starting* node, and u its *ending* node. That is, there is a sequence of x (not necessarily distinct) consecutive edges starting at v and ending at u. More formally, a sequence $(e_1,\ldots,e_s)$ is of *consecutive edges* if for $1 \le i < s$, the pair $e_i = (v_i,u_i,l_i)$ and $e_{i+1} = (v_{i+1},u_{i+1},l_{i+1})$ is such that $u_i = v_{i+1}$. The $x+1$ nodes in a walk of length x need not be distinct, as provenance graphs allow for cycles in specific circumstances (see the end of section). A *path*, on the other hand, is defined as a walk where nodes do not repeat.

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a valid *kernel* on set $\mathcal{X}$ if there is a real Hilbert space $\mathcal{H}$ and a mapping ψ such that $k(x,y) = \langle \psi(x), \psi(y) \rangle$. In order to show such a Hilbert space exists (and therefore k is called its *reproducing kernel*), it is enough to prove that k is symmetric and positive semi-definite (p.s.d.) [54, Theorem 3], i.e., for every subset $\{x_1,\ldots,x_t\} \subset \mathcal{X}$, we have that the $t \times t$ matrix M defined by $M(i,j) = k(x_i,x_j)$ is p.s.d. For M to be p.s.d, we simply need $\sum_{i,j} c_i c_j M(i,j) \ge 0$ for any constants $c_1,\ldots,c_t \in \mathbb{R}$.

In this work, we will study provenance kernels in both scenarios: when application-specific labels are provided and when they are not. In terms of the generality of graph structures considered in this paper, however, observe that the existence of cycles may not be discarded entirely: a long-running activity *a* that generates *e*, i.e., there is an edge gen(e,a), and at a later stage uses *e*, i.e., there is a new edge use(a,e), creates a cycle. This is to say that, although edges, for well-defined provenance, in general, 'point to the past', cycles cannot be excluded and, for that reason, our definitions make no restrictive assumptions on graph properties, such as acyclicity. Indeed, definitions apply to a general graph with labelled edges and nodes.

4.2 Provenance Types

Consider a node v in a provenance graph. The set of edges starting at this node may be seen as related to its recent history, i.e., such edges point to other nodes that may, in the case of entities, be the activity that generated it, or, in the

Table 2: The graph kernels evaluated in Section 5 and their properties: whether node and edge categorical labels are considered and their theoretical computational complexity. Symbol $\approx$ indicates that the support depends on the choice of the base kernel.

Kernel Method	Node label	Edge label	Complexity
Provenance Kernels (PK)	✓	✓	$\mathcal{O}(h^2 m)$
Shortest Path (SP)	✓	-	$\mathcal{O}(n^4)$
Vertex Histogram (VH)	✓	-	$\mathcal{O}(n)$
Edge Histogram (EH)	-	✓	$\mathcal{O}(m)$
Graphlet Sampling (GS)	-	-	$\mathcal{O}(nd^{k-1})$
Hadamard Code (HC)	✓	$\approx$	$\mathcal{O}(hm)$
Weisfeiler-Lehman (WL)	✓	$\approx$	$\mathcal{O}(hm)$
Neighbourhood Hash (NH)	✓	-	$\mathcal{O}(hm)$
Neigh. Subgraph P. Dist. (NSPD)	✓	✓	$\mathcal{O}(n\lvert S\rvert\lvert E(S)\rvert \log \lvert E(S)\rvert)$

case of activities, be the agent responsible for its execution. Going further, edges that are, in turn, connected to the neighbours of v, represent the earlier history of v, and so on. The idea of capturing the label of such edges, as well as that of nodes, taking into account their distance to the root, lies in the core of what we define as provenance types. Provenance types are the building blocks for provenance kernels.

We first show an example of how node names and labels appear in a provenance graph.

Example 1. *Refer back to Fig. 1. It depicts an example of a short patient hospitalisation, from admission to discharge. Here,* $V = \{\texttt{person13}, \texttt{patient7}_0, \texttt{ward27}, \ldots\}$, *while* $S = \{ent, ag, act\}$. *If application-specific types are used, S is enlarged to include labels such as* `mimic:Patient`, `mimic:Ward`, *etc. We adopt the de facto colour and layout convention for provenance graphs that shows entities as yellow-filled ellipses, activities as blue-filled rectangles, and agents as orange-filled trapeziums. Time flows downwards in this convention, in which the entities* $\texttt{patient7}_0, \ldots, \texttt{patient7}_3$ *represent the different 'states' of the same person originally represented by node* `person13`, *culminating at* $\texttt{patient7}_3$ *which also contains the provenance-specific label* `mimic:DischargedPatient`, *indicating that the hospitalisation ended with the discharging of the patient. The activities in this scenario are those that either admit the patient to a ward (*`admitting3`*) or represent a treatment (*`treating5`*). Each is associated with the respective hospital ward in which the activity took place.*

As a motivation for provenance types, consider nodes `admitting3` and `treating5` in Fig. 1. We can say that they share some similarities as both represent activities in this provenance graph, even though one is an admission and the other a treatment. Further, we can say that they share even more similarities as they are related to entities (via the *use* relation) and to some agent (via the *assoc* edge label). Going one step further, however, these nodes do not present the same 'history': `treating5` used an entity which was, in turn, generated by some other activity, whereas `admitting3` did not. We formalise this idea of capturing a simplification of a node's history as the *provenance types* of a node. First, we define label-walks, which will be later used in our definition of types.

Definition 1 (Label-walk)**.** *Let $G = (V, E)$ be a directed multigraph, and let $w = (e_1, \ldots, e_h)$ be a sequence of h consecutive edges from v to u.We define a label-walk as*

$$LAB(w) = (lab(e_1), lab(e_2), \ldots, lab(e_h), lab(u)) \quad (1)$$

$LAB(w)$ is then a sequence of h edge labels followed by the label of vertex $u \in V$. Thus, we say that $LAB(w)$ is a label-walk of length h. We further define $\mathcal{W}^h(v)$ to be the set of all label-walks of length h starting at a given node v. In the degenerated case of $h = 0$, the start and end nodes coincide and thus $\mathcal{W}^0(v)$ is defined by simply $(lab(v))$.

In simple terms, a label-walk is the sequence of the labels of edges along a walk followed by the label of its ending node. The intuition behind capturing the label of ending nodes of walks is twofold. First, we are able to define a base case, in which we consider a walk of null size, i.e., capturing only the node's labels (recall it can be a set). Secondly, we are able to make use of application-specific node labels (such as `mimic:Patient`, in Fig. 1), which may provide crucial information for the analysis of provenance data.

Example 2. *In this example, we consider only PROV generic types. Consider the sequence of two consecutive edges given by* $w = \Big((\texttt{patient7}_3, \texttt{treating5}, gen), (\texttt{treating5}, \texttt{patient7}_2, use)\Big)$. *We have* $LAB(w) = (gen, use, ent)$. *Moreover,* $\mathcal{W}^2(\texttt{patient7}_3) = \{$*(gen, use, ent), (gen, assoc, ag), (der, der, ent), (der, gen,act)*$\}$.

Note that label-walks do not necessarily follow shortest paths. In fact, we will in the following definition introduce the idea of provenance h-types, which use **all** label-walks of a given length and from a given node.

Definition 2 (Provenance h-types)**.** *Let G be a graph and $v \in V$. Let $\mathcal{W}^h(v)$ be the set of all label-walks of length h starting at v. We now capture all the labels that are equally distant from v: for $h \geq 1$, we define, for $1 \leq i \leq h$,*

$$\tau_i^h = \{lab(e_{h-i}) \mid (lab(e_1), lab(e_2), \ldots, lab(e_h), lab(u)) \in \mathcal{W}^h(v)\} \quad (2)$$

as the set of edge labels that are at a distance $(h-i)$-th when counting from v.[2] *For $i = 0$, we define*

$$\tau_0^h = \{lab(u) \mid (lab(e_1), \ldots, lab(e_h), lab(u)) \in \mathcal{W}^h(v)\} \quad (3)$$

2. This apparent reversed choice of indexing will simplify the algorithm to infer such provenance types. The intuition is that, with this notation, τ_0^h always refers to node labels for any h.

We define the provenance h-type of v as the sequence of sets

$$\phi^h(v) = (\tau_h^h, \dots, \tau_0^h) \tag{4}$$

In the case $\mathcal{W}^h(v) = \emptyset$, i.e. there is no walk of length h from v, we define $\phi^h(v) = \emptyset$. When clear from the context, we shall denote $\tau_i^h(v)$ as simply $\tau_i(v)$, or even τ_i.

For the degenerate case of $h = 0$, we define $\phi^0(v) = \mathcal{W}^0(v) = (lab(v))$.

Example 3. *Consider $\texttt{patient}7_3$ discussed in Example 2. Its 2-type combines the label-walks in $\mathcal{W}^2(\texttt{patient}7_3)$ to generate:*

$$\begin{aligned}&\phi^2(\texttt{patient}7_3) = \\ &= (\underbrace{\{gen, der\}}_{\tau_2^2}, \underbrace{\{use, assoc, der, gen\}}_{\tau_1^2}, \underbrace{\{ag, act, ent\}}_{\tau_0^2})\end{aligned} \tag{5}$$

Further, consider nodes $\texttt{admitting3}$ and $\texttt{treating5}$ once again in Fig. 1. Their 0-types, 1-types, and 2-types are:

$$\begin{aligned}&\phi^0(\texttt{admitting3}) = \phi^0(\texttt{treating5}) = (\{act\})\\ &\phi^1(\texttt{admitting3}) = \phi^1(\texttt{treating5}) = (\{assoc, use\},\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{act, ag\})\\ &\phi^2(\texttt{admitting3}) = (\{use\}, \{der\}, \{ent\})\\ &\phi^2(\texttt{treating5}) = (\{use\}, \{der, gen\}, \{act, ent\})\end{aligned}$$

These were both examples using only PROV generic types. By using application-specific types for the same two nodes $\texttt{admitting3}$ and $\texttt{treating5}$, on the other hand, we have $(\{act, \text{mimic:Admitting}\}) = \phi^0(\texttt{admitting3}) \neq \phi^0(\texttt{treating5}) = (\{act, \text{mimic:Treating}\})$.

Note that two different sets of label-walks may give rise to the same provenance type, although the converse is not true, i.e., two different types cannot come from the same set of label-walks. This implies that the function that maps sets of label-walks to types is not, in general, injective. We claim that this is beneficial as it unifies different sets of label-walks that have similar meanings in provenance. Note also that multiple occurrences of the same walks starting at v carry out no difference in the provenance types of v as opposed to just one copy of each different walk. Note also that the eventual presence of cycles does not pose any 'feedback loop' problem due to the fact that types are defined recursively - a h-type depends only $\tilde{h}$-type for $\tilde{h} < h$.

4.3 Provenance Kernel

Finally, we define the mapping of graphs into a high dimensional space by simply counting the number of occurrences of each provenance h-types up to depth h. We formally define a feature vector in the following definition.

Definition 3 (Feature Vector)**.** *Let $\mathcal{G}$ be a family of graphs and define $\phi^h(\mathcal{V}) = \{F_1, \dots, F_s\}$ as the (enumerated) set of all provenance types of depth **up to** h encountered in $\mathcal{V}$. The feature vector of a graph $G \in \mathcal{G}$ is given by:*

$$VEC^h(G) = (x_1, x_2, \dots, x_s) \tag{6}$$

where $x_i = \left|\{v \mid \phi^h(v) = F_i, \text{ and } v \in G\}\right|$

We apply this definition to our graph in Fig. 1 as an example.

Example 4. *Consider the provenance graph G depicted in Fig. 1. In order to infer $VEC^1(G)$, we need G's 0-types and 1-types. There are three of the former and four of the latter, and we denote them by:*

$$\begin{aligned}&F_1 = (\{act\}), \quad F_2 = (\{ag\}), \quad F_3 = (\{ent\}),\\ &F_4 = (\{assoc, use\}, \{act, ag\}), \quad F_5 = (\{spe\}, \{ent\}),\\ &F_6 = (\{der\}, \{ent\}), \quad F_7 = (\{gen, der\}, \{act, ent\})\end{aligned}$$

And thus

$$VEC^1(G) = (5, 2, 2, 2, 1, 1, 2) \tag{7}$$

We now use the definition of a feature vector to define provenance kernels.

Definition 4 (Provenance Kernel)**.** *Given two graphs $G, G' \in \mathcal{G}$ and $\phi^i(\mathcal{V})$ for all $0 \leq s \leq h$, we define the kernel between G and G' as*

$$k^h(G, G') = \sum_{s=0}^{h} \langle VEC^s(G), VEC^s(G') \rangle \tag{8}$$

where $\langle x, y \rangle$ denotes the dot product between x and y.

Note that types found in one graph may not occur in the other, in which case a zero must appear in the relevant dimension of a feature vector. In order to keep track of which type corresponds to which dimension, one can order all F_j found across all graphs for a given depth (say, by alphabetical order). With, that, the dimension of a given type is well-defined.

The goal of kernels in general is create representations of the data (graphs, in case of graph kernels) in a high dimensional space in which the classes are linearly separable. In practice, it is not always the case that such goal is fully achieved. Nonetheless, the better separated the data becomes, the better the results we obtained, with the caveat the overfitting may play a role in case the kernel is too fine-grained.

In order to avoid having to infer the feature vector for the same graph G multiple times (each time we want $k(G, G')$ for some G'), we may first infer all feature vectors and then calculate the inner products between each pair of graphs, similarly to the approach proposed by [26].

Proposition 5. *Provenance kernels are positive semi-definite.*

Proof. Let $c_1, \dots, c_t \in \mathbb{R}$ and $G_1, \dots, G_t \in \mathcal{G}$. Consider for a given depth s and pair of indices i, j, the dot product $\langle \text{VEC}^s(G_i), \text{VEC}^s(G_j) \rangle$. Then, $\sum_{i=1}^t \sum_{j=1}^t c_i c_j \langle \text{VEC}^s(G_i), \text{VEC}^s(G_j) \rangle =$ $\langle \sum_{i=1}^t c_i \text{VEC}^s(G_i), \sum_{j=1}^t c_j \text{VEC}^i(G_j) \rangle \geq 0$. The inequality follows from the fact that both sums add to exactly the same value and from $\langle x, x \rangle \geq 0$ for all x. Since the sum of non-negative numbers is non-negative, k^h is positive semi-definite. □

4.3.1 The Algorithm

We now present an algorithm that infers all ϕ^i for $0 \leq i \leq h$ for nodes in a family of graphs $\mathcal{G}$ in $O(h^2 M)$, where M is the total number edges among graphs in $\mathcal{G}$. For a single graph, the algorithm runs in $O(h^2 m)$, where m is the number of edges in this particular graph.

Algorithm 1: INFERRING TYPES UP TO h $(\mathcal{V}, \mathcal{E}, h)$

```
1  initialise for all $v \in \mathcal{V}$ and for all $i \leq h$
2      $\phi^i(v) \leftarrow \emptyset$
3      for all $0 \leq j \leq i$, $\tau^i_j(v) \leftarrow \emptyset$
4  for $v \in \mathcal{V}$
5      $\tau^0_0(v) \leftarrow lab(v)$
6      $\phi^0(v) \leftarrow (\tau^0_0(v))$
7  for $1 \leq i \leq h$
8      for each $e = (v, u) \in \mathcal{E}$ s.t. $\phi^{i-1}(u) \neq \emptyset$
9          $\tau^i_i(v) \leftarrow \tau^i_i(v) \cup lab(e)$
10         for $0 \leq j \leq i - 1$
11             $\tau^i_j(v) \leftarrow \tau^i_j(v) \cup \tau^{i-1}_j(u)$
12     $\phi^i(v) \leftarrow (\tau^i_h(v), \ldots, \tau^i_0(v))$
13 return $\phi^i(v)$ for all $v \in \mathcal{V}$ and $0 \leq i \leq k$
```

Figure 3: An algorithm that receives nodes and edges of a family of graphs, a parameter h, and outputs all provenance types $\phi^0(v), \ldots, \phi^h(v)$ for all nodes v.

Fig. 3 provides an algorithm to infer provenance h-types. First, we initialise all types $\phi^i(v)$ with the empty set for all depths up to h and all nodes. Moreover, we initialise as empty sets the building blocks of our h-types that record the labels of edges at a given distance from each node (lines 1-3). The intuition behind this explicit initialisation is that if we do not update a given $\phi^i(v)$, the empty set will indicate that there are no label-walks of size i starting at v. This will be used later as a condition in line 8.

The loop starting at line 4 infers the base of our algorithm: the 0-type of all nodes, i.e., $\phi^0(v)$, which is simply the set of labels of v for each $v \in \mathcal{V}$. Each iteration of the loop starting in line 7 will infer the i-type for all nodes. We first loop through all edges in $\mathcal{E}$ that can 'lead us somewhere'. In other words, we are considering only edges $e = (v, u)$ that belong to walks of size i starting at v. This is true if and only if $\phi^{i-1}(u) \neq \emptyset$. We then make sure that labels of e are added to the set $\tau^i_i(v)$ (line 9). Finally, the loop starting at line 10 adds to the set of $\tau^i_j(v)$ the labels from set $\tau^{i-1}_j(u)$. We can finally in line 12 construct the entire i-type for all nodes.

To see that the algorithm correctly infers provenance types, note that line 8 guarantees that all label-walks of length i starting at v will be identified. Further, the lines 10 and 11 make sure that all labels in each of these label-walks will be fully inspected and added to v's type accordingly.

4.3.2 Complexity Analysis

We are now showing that we need $O(h^2M)$ operations to infer the $\phi^h(v)$ for each node v in a family of graphs $G_1, \ldots, G_s := \mathcal{G}$. Here, N stands for the sum of the number of nodes in all provenance graphs and M for the sum of the number of edges. We borrow part of the argument from [26]. Lines 1-3 can be done in $O(h^2N)$, since we are initialising $\frac{1}{2}(h+1)(h+2)$ sets for each node in $\mathcal{V}$. Lines 4-6 take $O(N)$. Let us now investigate the *for* loop initiated in line 7. We are entering this loop h times, and in each of them we are investigating each edge at most once (loop stating in line 8), and finally, for each edge, we are performing at most h pairwise operations on sets of constant size (bounded by $\max\{|T|, |L|\}$). Line 12 takes $O(N)$. Thus loop starting at line 7 can be done in $O(h^2M)$, assuming $N = O(M)$, which gives us the overall running time bounded by $O(h^2M)$ when we assume $N = O(M)$.[3]

5 EMPIRICAL EVALUATION

As a tool for extracting features from provenance graphs, provenance kernels can be used to compare a provenance graph to another and build classifiers for them. To demonstrate the approach, we employ provenance kernels in classification tasks on six provenance data sets (see below). In our evaluation, we compare the accuracy of provenance kernels against generic graph kernels and the PNA method (discussed in Section 3) in the same classification tasks. We describe the evaluation methodology in Section 5.2 and report the evaluation's results in Section 5.3.

5.1 Data sets and classification task

We employed six provenance data sets in our evaluation; they were produced by three different applications: MIMIC [55], CollabMap [56], and a Pokémon Go simulator. These applications cover a spectrum of human and computational processes. The first, MIMIC, records solely human activity; the second, CollabMap, is created with computational workflows driven by human inputs, whereas the Pokémon Go simulator is a fully synthetic system. Section 2 of the Supplementary Materials provides more details on these applications and their six data sets (summarised in Table 3). In brief, each data set contains multiple provenance graphs recorded in the three applications; each graph is associated with a label.

- `MIMIC` provenance graphs describe the journey of patients going through a hospital, e.g. the ward they stayed in, the treatments they received and who performed those procedures. Each graph is associated with a label representing in-hospital mortality, which has either a value of 0 or 1.
- The CollabMap application provides three data sets: `CM-B`, `CM-R`, and `CM-RS`, which, respectively, contain the provenance graphs for the buildings, routes, and route sets created by CollabMap contributors. Each graph is associated with either a label 'trusted' or 'uncertain', representing the level of uncertainty associated with a building, route, or route set (as judged by an expert).
- The Pokémon Go simulator generated two data sets, `PG-T` and `PG-D`. A provenance graph therein recorded the activities of a (simulated) Pokémon Go player following their own team's game strategy on targeting Pokémons (`PG-T`) and disposing of Pokémons (`PG-D`). Each graph is associated with the team's name, either `Valor`, `Mystic`, or `Instinct`.

Given a provenance graph from a data set, the classification task here is to predict or re-infer the label associated with that graph; see Table 3 for an overview of the six classification tasks.

3. We empirically evaluated the computational cost of the algorithm over randomly-generated provenance graphs of various sizes and found that it scales linearly with the number of nodes in a graph. See Section 4 for more details.

Table 3: The classification tasks, the number of samples picked from each data set, and the number of application types present in each data set (in addition to PROV generic types).

Application:	MIMIC	CollabMap			Pokémon Go simulator	
Data set:	MIMIC	CM-B	CM-R	CM-RS	PG-T	PG-D
Classification labels:	0/1	trusted/uncertain			Valor/Mystic/Instinct	
Random baseline:	50%	50%	50%	50%	33%	33%
Sample size:	4,586	1,368	2,178	3,382	1,200	1,200
No. application types:	125	8	8	8	8	8

5.2 Methodology

For each classification task, in order to ensure the robust evaluation of provenance kernels' performance compared to that of existing graph kernels and the PNA method, we carry out the following: balancing the input data set (if unbalanced), training classifiers with provenance kernels (PK), generic graph kernels, and provenance network metrics, measuring the performance of each classifier, and comparing their performance. An overview of the full evaluation pipeline, implemented in Python, is depicted in Fig. 4.

Data balancing The MIMIC and CollabMap data sets are significantly skewed, being originated from real-world human activities. Therefore, for those data sets, we balance the number of samples in each class by selecting all the samples in the minority class and randomly under-sampling the majority class to produce a balanced data set. Table 3 shows the number of samples used in each classification task after balancing.

Classification methods In addition to building classifiers for a classification task in question using provenance kernels, we also build classifiers using existing graph kernels, implemented by the Grakel library [57], and the provenance network metrics as proposed in the PNA method [12].

- Provenance Kernels: A provenance kernel is built on provenance types of depth up to a specified level h which may include (1) *only* the PROV generic (application-agnostic) types, i.e. *ent*, *act*, and *ag*, or (2) application-specific types (such as the `mimic:Patient` and `mimic:Ward` types shown in Fig. 1) *in addition to* the PROV generic types. We test both provenance kernels using only generic types and those including application types in our evaluation; we call the former group PK-G and the latter PK-A. We also evaluate provenance kernels for different levels of h, from 0 to 5. Hence, the methods we test in these two groups are: G0, G1, . . . , G5, A0, . . . , A5; the first letter in their names denotes whether they use only PROV generic types (G) or not (A) and the second denotes the specified level h; twelve PK-based methods are tested in total.
- Graph Kernels: The graph kernels we test are Shortest Path (SP) [25], Vertex Histogram (VH) [52], Edge Histogram (EH) [52], Graphlet Sampling (GS) [51], Weisfeiler-Lehman (WL) [26], Weisfeiler-Lehman Optimal Assignment (WL-OA) [58], Hadamard Code (HC) [53], Ordered Dag Decomposition (ODD) [59], Neighbourhood Hash (NH) [46], Neighbourhood Subgraph Pairwise Distance (NSPD) [47].[4] Similar to provenance kernels, Weisfeiler-Lehman and Hadamard Code kernels can be computed up to a specified level h; we test those kernels with $h \in [1, 5]$. Hence, a total of 16 graph kernels are tested. As shown in Fig. 4, support-vector machines (SVM) are used to build classifiers from both provenance kernels and generic graph kernels. Among the tested graph kernels, the SP, GS, ODD, NH, NSPD, WL-OA kernels take a significantly longer time to run compared to the rest. We, therefore, for comparison purposes, put these methods in a group called GK-slow and the remaining kernels in GK-fast.
- Provenance Network Analytics: The PNA method calculates 22 network metrics for each provenance graph and using those as the feature vector for that graph. Such feature vectors can be readily taken as inputs by a variety of ML classifiers. Given the six data sets, we are uncertain which classifiers work best with their provenance network metrics; hence, we employ a host of off-the-shelf algorithms to build classifiers on the metrics and pick the most accurate classifier for our performance comparisons. The chosen algorithms are Decision Tree (DT), Random Forest (RF), K-Neighbour (KN), Gaussian Naive Bayes (NB), Multi-layer Perceptron neural network (NN), and Support Vector Machines (SVM).[5] Hence, six methods are tested in total, all are provided by the Scikit-learn library [60]. This group of classifiers, which rely on provenance network metrics, is called PNA.

For methods that rely on SVM, its C parameter is optimally chosen from a grid search (with an inner cross-validation process) from $\{10^{-4}, 0.001, 0.01, 0.1, 1.0, 10, 100, 1000\}$ to give the best accuracy. GNNs, as discussed in previous work, focus on generalising the message passing concept at the same time as using a neural network architecture. The comparison with graph kernel methods, however, is not direct: there is no creation of feature vectors or kernel matrices; and as a consequence the training time in GNNs includes both creation of features and the classification ML algorithm for graph kernel methods. In theory, GNNs should take longer to train. In practice, that is also what we observe. Please refer to the supplementary material for details.

Performance metrics The performance of each method is measured by its accuracy, defined as the number of correct predictions over the total number of samples, in predicting the correct label of a provenance graph which is provided with the above data sets. To robustly measure the perfor-

4. We also tested other kernels provided by the Grakel library. However, they either timed out or produced errors when processing graphs in our data sets and, hence, are not included in our evaluation.

5. The parameters to set up the above algorithms are provided in a Jupyter notebook as detailed in Section 3 of the Supplementary Materials.

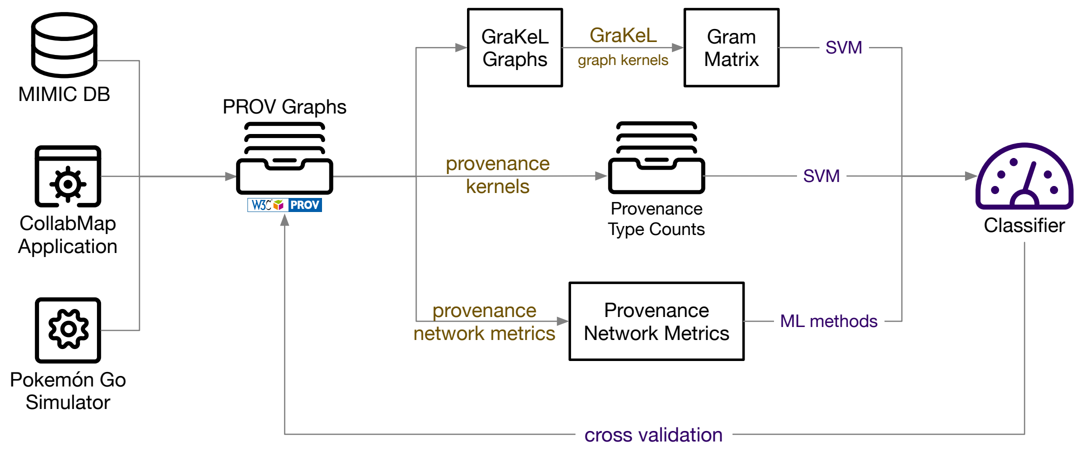

Figure 4: Overview of the evaluation pipeline. Six data sets of labelled provenance graphs from three applications are used to build classifiers using provenance kernels, generic graph kernels (from the GraKeL library), and provenance network metrics (PNA). Repeated 10-fold cross-validation is carried out to measure the classification accuracy of each method. We also measure the time each method takes to produce their kernels/network metrics (i.e. the yellow step).

mance, 10-fold cross-validation is employed. In particular, with all the available provenance graphs randomly split into 10 equal subsets, we perform 10 rounds of learning; on each round, a 1/10 subset is held out as the test set and the remaining are used as training data. This process is repeated 10 times; hence, 100 measures of accuracy are collected for each method per experiment. In addition, to understand the computation cost of each method, we measure the time it takes to produce provenance kernels, graph kernels, and provenance network metrics (the yellow step in Fig. 4) given the same data set used in a classification task. The time measurements do not include the time taken in training the classifiers nor the time preparing the input GraKeL graphs.

Comparing performance Due to the large number of methods evaluated from the five groups (i.e. PK-G, PK-A, GK-slow, GK-fast, PNA), we report here only one best-performing method from each group, i.e. the one with the highest mean classification accuracy within its group. We then compare the mean accuracy of the best-performing PK-based methods (i.e. PK-G, PK-A) against those from the remaining three groups to establish whether PK-based methods offer improved accuracy in the six classification tasks over existing graph kernel and PNA methods. In order to ensure that our comparison results are statistically significant, we carry out the Wilcoxon–Mann–Whitney ranks test [61, Ch. 10], also known as the Wilcoxon rank-sum test, when comparing the accuracy measurements of two methods. If the test produces a *p*-value that is less than 0.05, we reject the null hypothesis that states that the accuracy measurements are from the same distribution, i.e one method performs statistically better than the other. Otherwise, both methods are considered to have a similar level of performance. In addition, in real terms, we disregard accuracy differences of less than 1% and consider the corresponding methods to have a similar level of performance.

Table 4: Within each data set, we report the time cost of the best-performing method (shown in parentheses) from each comparison group *relative* to the time taken by the best-performing PK-A method (whose time cost is shown as 1x).

	PK-G	PK-A	GK-fast	GK-slow	PNA
MIMIC	4.3x (G5)	1x (A0)	2x (WL2)	243x (GS)	280x
CM-B	1.7x (G2)	1x (A0)	2x (HC3)	60x (WLO4)	160x
CM-R	0.9x (G4)	1x (A5)	1x (WL2)	37x (WLO5)	92x
CM-RS	0.9x (G5)	1x (A3)	1.8x (WL4)	57x (WLO5)	104x
PG-T	0.7x (G3)	1x (A3)	0.7x (WL5)	4.8x (ODD)	80x
PG-D	1.6x (G5)	1x (A2)	1x (WL5)	4.1x (SP)	166x

5.3 Evaluation Results

In this section, we report the performance of provenance kernels (PK-G and PK-A) compared to that of existing generic graph kernels (GK-slow and GK-fast) and the PNA method (PNA) across the classification tasks corresponding to the six provenance data sets (MIMIC, CM-B, CM-R, CM-RS, PG-T, and PG-D). As previously mentioned, for brevity, we only report the best-performing method in each group in terms of their mean classification accuracy.

Computational costs Before delving into the performance of the five comparison groups, it is pertinent to have an idea of the time costs incurred by them. Within each classification task, using the time cost of the best-performing PK-A method as the relative time unit (i.e. 1x), Table 4 shows the relative time costs[6] of the best-performing method from each comparison group as multiples of the chosen time unit. Formally, let $X \in \{\text{PK-G}, \text{PK-A}, \text{GK-fast}, \text{GK-slow}, \text{PNA}\}$. The entry for a given data set associated with the comparison group X is given by (time-of-best-in-X) divided by (time-of-best-in-PK-A), where 'time-of-best-in-X' stands for the time cost of the most accurate method in X, whereas 'time-of-best-in-PK-A' stands for the time taken by the most accurate PK-A method.

6. The actual time costs are provided in the online Jupyter notebook.

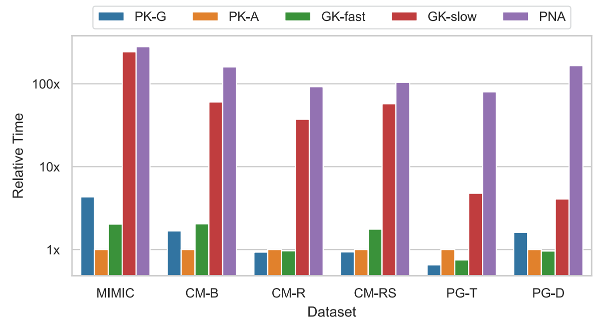

Figure 5: The *relative* time costs of the best-performing methods reported in Table 4 plotted on the log scale. The time cost of the best PK method is 1x.

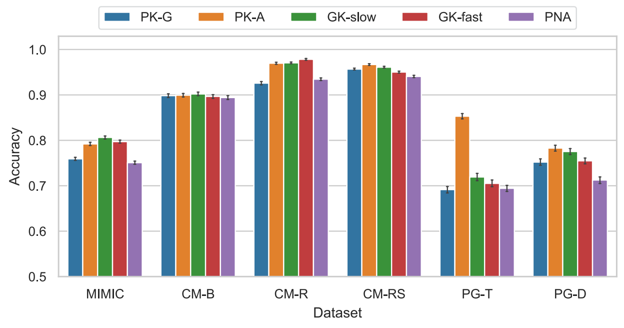

Figure 6: The mean classification accuracy of the best-performing provenance kernels, generic graph kernels, and PNA method across the six classification tasks. The error bars show the 95-percent confidence intervals.

We also plot those time costs in Fig. 5 using the logarithmic scale to highlight their differences. Across the data sets, the PK-G and PK-A methods take somewhat a similar time to produce the provenance kernels from the same set of provenance graphs. Their differences are mainly due to the different h levels (of type propagation). We observe more variation in relative time costs of the GK-fast group's methods, but they stay in the same magnitude of scale. The best-performing graph kernels in the GK-slow group, however, take between 4x to 200x longer than the baseline PK method to compute. The PNA method is the slowest, taking 80x to 280x longer (to compute the provenance network metrics for the same set of provenance graphs). Understanding the computational cost of each method, in addition to its classification performance, will be useful when deciding whether it is suitable for a given classification task.

Classification accuracy Fig. 6 plots the mean accuracy of the best-performing provenance kernels compared against that of the best-performing graph kernel and the best-performing PNA method in the six classification tasks (see Table 4 for the identifier of the best-performing method in each group, shown in parentheses). At first glance, in each task, the accuracy levels attained by the five comparison groups are broadly close and significantly above the random baseline. This shows that the proposed provenance types employed by PK-based methods, the graph information relied on by GK methods, and the provenance network metrics used in PNA method can all serve effectively as predictors for these classification tasks. However, their contributions to the accuracy of the corresponding classifiers vary. To account for statistical variations, we carry out the Wilcoxon–Mann–Whitney ranks tests to compare the accuracy level of the best PK-based method with that of another comparison group in each classification task. Table 5 presents the results of those tests where an "=" sign indicates that the difference in accuracy is either less than 1% or is not statistically significant; otherwise, a positive/negative value represents the accuracy gain/loss attained by the best PK-A method compared to the best method from the corresponding comparison group.

Table 5: Summary of the accuracy differences between the best-performing PK-A method and the best-performing method in the PK-G, GK-slow, GK-fast, and PNA groups. An "=" sign means the accuracy difference is not significant; while a positive/negative value shows how much the PK-based method outperforms/under-performs the corresponding GK/PNA method, respectively.

Data set:	MIMIC	CM-B	CM-R	CM-RS	PG-T	PG-D
PK-G	+3%	=	+4%	+1%	+16%	+3%
GK-slow	=	=	=	=	+13%	=
GK-fast	=	=	=	=	+15%	+3%
PNA	+4%	=	+4%	+3%	+16%	+7%

Comparison with PK using only generic types (PK-G) PK-A outperforms PK-G across the tested data sets, except for CM-B, where both methods have the same accuracy. This shows that application-specific types contribute non-trivially to the classification power of a provenance graph kernel, which justifies the extra investment by subject matter experts introducing application types during the provenance modelling process. In cases where such types are not available, the performance of PK-G can be considered the worst-case performance of provenance graph kernels.

Comparison with Graph Kernels If computational/time cost is not a consideration, we find that the GK-slow group generally outperforms the GK-fast group (see green bars vs. red bars in Fig. 6). Compared to the GK-slow group, PK-A methods, however, yield similar levels of accuracy across the tested classification tasks exception for MIMIC where the Graphlet Sampling kernel yields 1% more accurate classifications (at 243x more time cost) and PG-T where the best PK-A method is 13% more accurate than the best GK-slow method. In terms of computation costs, it should be noted that the best GK-slow methods take 4x to 243x more time than their PK-A counterparts (see Table 4). Compared to the GK-fast group, Table 5 shows that PK-A methods are more accurate in two out of six classification tasks and perform similarly in the remaining four tasks. Hence, under time constraints (that disqualify graph kernels in the GK-slow group), the proposed provenance kernels overall outperform the tested graph kernels in the GK-fast group.

Comparison with PNA method We also report in Table 5 the accuracy differences between the best PK-A methods compared to their PNA counterparts across the six classification tasks. It shows that PK-A methods outperform in all tasks with the exception of `CM-B` where both groups perform similarly. At the same time, given the significant penalty in computation cost incurred by the PNA method in calculating network metrics (80–280 times, see Table 4), PK-A methods prove to be better candidates for analysing provenance graphs. Moreover, the PNA method, in some of the tasks, employs obscure metrics like the average clustering coefficients predominantly in the trained decision models, making it a challenge to understand why certain classification i s d ecided, e ven w ith a n i nterpretable model such as a decision tree classifier.

In addition to good accuracy and time performance, in the following section, we show how provenance types used by the proposed provenance kernels can afford us better interpretability compared to the network metrics employed by the PNA method.

6 Explainability

In the previous section, Support Vector Machines are used to learn from the kernels' feature vectors, which are counts of provenance types, and perform classification tasks. Techniques such as SVMs, however, are known as black-box models when considering their ability to provide explanations of their predictions. In this section, we use LIME [62], short for Local Interpretable Model-Agnostic Explanations, to illustrate how it can help identify provenance types that are most influential i n c lassification de cisions an d, from those, gain insights into classifiers b uilt o n provenance kernels.

LIME aims to explain the decisions of any classifier by introducing local perturbations of input data, through which it learns a linear model which is locally faithful to the instance to be explained. For example, for tabular input data, it changes the value of a given feature, tests the perturbed feature vector with the same classifier, and checks whether such a change affects the prediction probabilities. The steeper the decrease in the prediction probability, the higher the contribution of that feature towards a specific prediction.

Although provenance types alone do not explain the entirety of a process, they provide practitioners with a tool to better understand the decision process of graph classifiers. We outline the steps to explain classification decisions using provenance kernels as follows.

E1 Important Types Identification: Provenance kernel is an explicit graph kernel, i.e., we are able to inspect the feature vector used in the classification of each given graph. This, in turn, provides us with the set of provenance types that were present and used in such predictions. From the explanations for each instance produced by LIME, we aggregate the importance of each feature across the entire data set and identify which provenance types were most influential in a given classification task.

E2 Instance Retrievability: Once the provenance types of interest are identified as being associated with a certain prediction, we are then able to retrieve original graph instances that contain patterns defined by one of the identified types. This is done by the subtasks (1) *Graph instance identification*: searches which graph has a given feature in its feature vector; and (2) *Subgraph retrieval*: extracts a subgraph that matches a provenance type in a given graph instance. Both combined may give us further insights into why a prediction was made as well as highlighting all provenance graphs that share the same provenance type(s).

E3 Instance Description: Once an instance of the interested pattern is found, we paraphrase such instance in a natural language by leveraging the descriptive capabilities of provenance vocabularies.

Table 6: The average importances of provenance type intervals associated with $F_{2,2}$, $F_{0,5}$, $F_{1,1}$ from `CM-B` (see Table 7 for definitions) as produced by LIME.

Parameter	Importance
$x_{1,1} \leq 2$	0.322
$0 < x_{2,2} \leq 2$	0.203
$x_{0,5} \leq 1$	0.0
$x_{2,2} = 0$	-0.204
$x_{1,1} > 2$	-0.320

We illustrate the above steps with the `CM-B` date set. First, we split it into a train set and a test set (at a ratio of 8:2). We then train an SVM classifier on the train set. Following Definition 3, we denote by $x_{h,y}$ the number of occurrences of type $F_{h,y}$ on a given graph, where h is a given depth and y is a counter for types of that particular depth (e.g., types of depth 2 are $F_{2,1}, F_{2,2}, F_{2,3}$...). For each graph in the test set, we record the importance of each feature from LIME-generated explanations. The explanations were set up in a way such that the number of occurrences of each type in a particular graph ($x_{h,y}$) was split into intervals, acting as a new (binary) feature. For example, in Table 6, the feature $x_{1,1} \leq 2$ is found to have the highest positive importance value. This means that if the type $F_{1,1}$ (see Table 7) occurs 0, 1 or 2 times in a `CM-B` provenance graph, the classifier is likely to decide that the graph is `trusted`. The quantification of this association (e.g., 0.322) is related to the drop in the probability of being considered `trusted` when such a feature is artificially removed from the feature vector. A negative importance value indicates that a feature is associated with an `uncertain` decision, as its removal increases the probability of a `trusted` outcome. Note this analysis also allows us to conclude, for example, that a decision was made based on the *absence*, rather than occurrence, of a given provenance type.

Table 7 provides us with the type definitions associated with each one of the types $F_{1,1}$, $F_{2,2}$, $F_{0,5}$ (**E1**). Although $F_{0,5}$ can simply be paraphrased as 'a Buiding', when it comes to deeper types, however, their natural language descriptions need to make use of more general terms. For example, $F_{2,2}$ can be paraphrased as representing a 'succession of two derivations from a Route or Building'. Type descriptions alone, therefore, may not fully capture the complexity of graph patterns to give the user a broad understanding of their meaning. For that reason, we propose the extraction of a subgraph associated with an occurrence of each one of the influential types in Table 6 (**E2**). Such a subgraph may by no means be the only graph pattern associated with a type but provides the user with a concrete

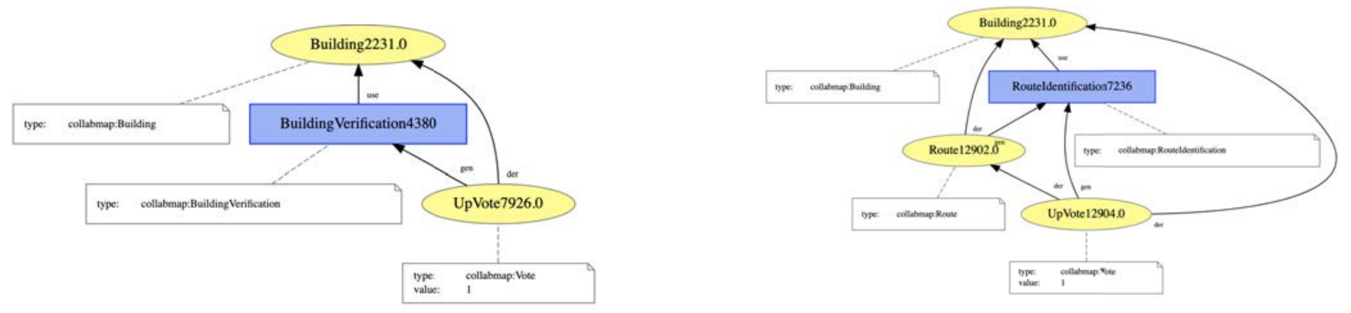

Figure 7: An instance of a $F_{1,1}$ type (left), associated with node *UpVote7926.0* and an instance of a $F_{2,2}$ type (right), associated with node *UpVote12904.0*. Note that similar patterns associated with a DownVote would define the same provenance types, as both positive and negative votes are associated with the same provenance type *collabmap:Vote*.

scenario of its occurrences. Fig. 7, for instance, depicts two subgraphs from `CM-B` representing single instances of types $F_{1,1}$ and $F_{2,2}$. Each is the induced subgraph from all nodes that are up to distance h, where h is the depth (e.g. 1 or 2) of the provenance type of its root note (e.g. *UpVote7926.0*, or *UpVote12904.0*). Finally, the retrieved subgraphs can be paraphrased in a natural language **E3**. For example, the following sentence was computationally generated from the graph on the right of Fig. 7 (i.e., an instance of $F_{2,2}$):

> *Route12902.0 was generated by RouteIdentification7236, UpVote12904.0 relates to Route12902.0., UpVote12904.0 was generated by RouteIdentification7236. RouteIdentification7236 used Building2231.0.*

In Table 6, the type $F_{1,1}$ is associated with `trusted` decisions when appears fewer than two times, and associated with `untrusted` decisions if it is featured more frequently. Note that in this case LIME aggregates the absence of type $F_{1,1}$ (i.e, $x_{1,1} = 0$) and its single occurrence (i.e., $x_{1,1} = 1$) in the same interval. In Fig. 7 this type refers to a vote on a building associated with a verification. A natural way to interpret this is, since a *Vote* can be either an up or down vote, we can take this to mean that having a small number of votes likely means that they were upvotes. This is in line with how the CollabMap workflow [56]: when there was a consensus with just a few votes (often upvotes), the Building was declared `trusted`. Otherwise, if there was a dispute, more verifications, and hence votes, were requested, and thus it is more likely that the Building is deemed `uncertain`. In a narrative, we can say that a disputed decision is associated more with an `uncertain` decision. Moreover, a contrasting pattern is presented with the depth-2 type $F_{2,2}$. In particular, a non-zero number of occurrences was associated with `trusted` decisions, whereas the absence of such type was deemed to be associated with a `uncertain` one. This once again is in line with the application's workflow: if a Building has not been declared `trusted`, no routes are requested from crowd workers for the building.

Note that the number of votes alone does not offer a strong explainable power. This is because a vote can be either associated with a building verification or with a route identification, which has different implications when understanding the trustworthiness of a building. For example, a high number of votes can be associated with a stronger likelihood of a building being deemed `untrusted` if they

Table 7: The provenance type descriptions of types $F_{2,2}$, $F_{1,1}$, and $F_{0,5}$. The prefix *cm:* indicates an application-specific label from the CollabMap domain. *cm:RouteID* represents a *Route identification* type.

Feature	Provenance type represented
$F_{1,1}$	$\{der, gen\}, \{act, ent, \textit{cm:Building}, \textit{cm:BuildingVerification}\}$
$F_{2,2}$	$\{der, gen\}, \{der, gen, use\}, \{act, ent, \textit{cm:Building}, \textit{cm:RouteID}\}$
$F_{0,5}$	$\{ent, \textit{cm:Building}\}$

are all associated with building verification activities. On the other hand, the same high number of votes can be associated with a graph that has few building verification activities together with several route identification ones, which means the building was declared as `trusted` by the workflow. Such real-world examples exemplify the importance of provenance types at higher depths as opposed to graph kernels that simply count the number of occurrences of each type node label (i.e., a vote in this case). Finally, the overall importance of the feature $F_{0,5}$ was determined to be 0, which implies that the number of buildings offers no explanatory power to the decision of the classifier. Once again, this is in line with how the data set was constructed as *Building* appears exactly once in each graph.

In summary, through using an explainer, LIME, to build locally interpretable models for provenance kernel classifiers, the importance of each of the provenance types, including its constraints, can be determined, providing the user with a means to interpret the classifier's decisions. We have shown that, in the context of the CollabMap building classification task, such interpretations reveal the logic captured from the data by the classifier (and later verified by us by checking the application's workflow). Capitalised on the expressiveness of the provenance vocabulary and the patterns encoded by provenance types, insights into the logic of classifiers built on provenance kernels can be identified by following the steps we outlined above in this section.

7 Conclusions and Future Work

With the growing adoption of provenance in a wide range of application domains, the efficient processing and classification of provenance graphs have become imperative. To that end, we introduced a novel graph kernel method tailored for provenance graphs. Provenance kernels make use of provenance types, which are an abstraction of a node's neighbourhood taking into account edges and nodes at different distances from it. A provenance type is associated with each node for each depth value h. A vector is then produced for each graph: it counts the number of occurrences of each (non-empty) provenance type associated with nodes in this graph up to a given depth h. These feature vectors can then be effectively employed by standard machine learning algorithms, such as SVMs and decision trees, as shown in the previous two sections. The computational complexity of producing feature vectors for a family of graphs with a total of M edges is bounded by $\mathcal{O}(h^2M)$. Note that the provenance kernel method is applicable to graphs in any domain as long as both edges and nodes are categorically labelled.

In Section 5, we compared provenance kernels against state-of-the-art graph kernels and the PNA method in supervised learning tasks with six data sets of provenance graphs. We showed that provenance kernels are among the fastest methods and, among those, they show high, if not the highest, classification accuracies in the six data sets we tested. An important benefit brought about by provenance types is that they can help us understand better how certain classification is made by an ML model as demonstrated in Section 6. We provided a sequence of steps to extract explanations from classification tasks by inspecting the importance of different features, making use of the fact that provenance types capture narratives from sequential events in provenance graphs. We thus show how provenance kernels and types may give us further insights into why a particular graph was classified in a particular way. Also, in the context of a CollabMap data set, such explanations were in line with our knowledge with respect to how the graphs were constructed.

A given provenance type, when considering only PROV generic node labels, may re-appear in provenance graphs recorded from different applications. The extent of how many types overlap across application domains is an open question. In line with what has been proposed by [19], an interesting line of future work is to create a library of types across different domains and investigate whether there is a correlation between the high occurrence of certain provenance types and the role they play in classification tasks. We have proposed a method that takes both node and edge labels into account, as opposed to node or edge attributes (such as time, duration, etc). It is an interesting line of future work to study a modification of our approach to also take into account real values in edges and nodes. This could be interesting as some provenance graphs have duration of activities or other time stamps that could be used.

As we found that application-specific types are essential in improving the accuracy of provenance graph kernels (Section 5.3) and explaining a classification decision (Section 6), designing application types to maximise such benefits is a valuable extension of this work. Moreover, one can study the use of random Fourier features [63], [64] to address the challenge of high-dimensional feature vectors. Like the work proposed in [65], an automated method to generate natural language descriptions of extracted instances of provenance types (i.e., small provenance graphs, see Fig. 7) will further improve the interpretability of classifiers built on provenance graphs kernels. Furthermore, it is an interesting line of future work to create perturbations in provenance graphs (e.g., removal of nodes or edges) as means of creating explanations based on such perturbations. One most note, however, that not all perturbations of provenance graphs are valid changes, and violating rules of encoding provenance may lead to inaccurate explanations.

Finally, as shown in our preliminary investigation (Section 5 of the Supplementary Materials), GNNs seem to be able to classify provenance graphs with high accuracy (albeit with significantly high time cost). As discussed there, the research space on GNNs is vast. A follow-on study to determine which GNN layers and configurations are suitable to efficiently analyse provenance graphs is another interesting line of future work.

Code and Data

The data used for this article, along with the associated experiment code, is publicly available at https://github.com/trungdong/provenance-kernel-evaluation.

David Kohan Marzagão is a Lecturer at the Department of Informatics, King's College London. He is also affiliated with the University of Oxford.

Trung Dong Huynh is a Research Fellow in the Department of Informatics, King's College London. He is the main developer of the PROV Python package and ProvStore. Previously, he developed computational models of trust and reputation for multi-agent systems.

Ayah Helal is a Lecturer in Computer Science at Exeter University.

Sean Baccas is a Machine Learning Engineer at Polysurance. He obtained his MMath from the University of Durham.

Luc Moreau is a Professor of Computer Science and Head of the department of Informatics, at King's College London. Previously the co-chair of the W3C Provenance Working Group that produced the PROV standard.

References

[1] P. Alper, K. Belhajjame, C. A. Goble, and P. Karagoz, "Enhancing and abstracting scientific workflow provenance for data publishing," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. New York, NY, USA: ACM, 2013, pp. 313–318.

[2] F. Chirigati, D. Shasha, and J. Freire, "Reprozip: Using provenance to support computational reproducibility," in *Proceedings of the 5th USENIX Conference on Theory and Practice of Provenance*. Berkeley, CA, USA: USENIX Association, 2013.

[3] X. Ma, P. Fox, C. Tilmes, K. Jacobs, and A. Waple, "Capturing provenance of global change information," *Nature Climate Change*, vol. 4, no. 6, pp. 409–413, 2014.

[4] S. D. Ramchurn, T. D. Huynh, F. Wu, Y. Ikuno, J. Flann, L. Moreau, J. E. Fischer, W. Jiang, T. Rodden, E. Simpson, S. Reece, S. Roberts, and N. R. Jennings, "A disaster response system based on human-agent collectives," *Journal of Artificial Intelligence Research*, vol. 57, pp. 661–708, 2016. [Online]. Available: http://www.jair.org/papers/paper5098.html

[5] L. Moreau and P. Missier, "PROV-DM: The PROV data model," World Wide Web Consortium, Tech. Rep., 2013, W3C Recommendation. [Online]. Available: http://www.w3.org/TR/2013/REC-prov-dm-20130430/

[6] X. Zou, "A survey on application of knowledge graph," in *Journal of Physics: Conference Series*, vol. 1487, no. 1. IOP Publishing, 2020, p. 012016.

[7] P. Groth, Y. Gil, J. Cheney, and S. Miles, "Requirements for provenance on the web," *International Journal of Digital Curation*, vol. 7, no. 1, pp. 39–56, 2012.

[8] N. Kwasnikowska, L. Moreau, and J. Van den Bussche, "A formal account of the open provenance model," *ACM Transactions on the Web*, vol. 9, February 2015.

[9] J. Cheney, P. Missier, L. Moreau (eds.), and T. D. Nies, "Constraints of the PROV Data Model," World Wide Web Consortium, W3C Recommendation REC-prov-constraints-20130430, Apr. 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-prov-constraints-20130430/

[10] U. Braun and A. Shinnar, "A security model for provenance," Harvard University Computer Science, Tech. Rep. TR-04-06, Jan. 2006. [Online]. Available: https://dash.harvard.edu/bitstream/handle/1/23586584/tr-04-06.pdf

[11] B. C. Pierce, *Types and programming languages*. MIT press, 2002.

[12] T. D. Huynh, M. Ebden, J. Fischer, S. Roberts, and L. Moreau, "Provenance Network Analytics," *Data Mining and Knowledge Discovery*, feb 2018. [Online]. Available: http://link.springer.com/10.1007/s10618-017-0549-3

[13] N. Kwasnikowska, L. Moreau, and J. V. D. Bussche, "A formal account of the open provenance model," *ACM Transactions on the Web (TWEB)*, vol. 9, no. 2, pp. 1–44, 2015.

[14] J. Cheney, P. Missier, L. Moreau, and T. D. Nies, "Constraints of the PROV data model," World Wide Web Consortium, W3C Recommendation REC-prov-constraints-20130430, Apr. 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-prov-constraints-20130430/

[15] S. Miles, P. Groth, S. Munroe, and L. Moreau, "Prime: A methodology for developing provenance-aware applications," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 3, pp. 1–42, 2011.

[16] C. Sáenz-Adán, B. Pérez, T. D. Huynh, and L. Moreau, "Uml2prov: automating provenance capture in software engineering," in *SOFSEM 2018: Theory and Practice of Computer Science: 44th International Conference on Current Trends in Theory and Practice of Computer Science, Krems, Austria, January 29-February 2, 2018, Proceedings 44*. Springer, 2018, pp. 667–681.

[17] L. Moreau, B. V. Batlajery, T. D. Huynh, D. Michaelides, and H. Packer, "A templating system to generate provenance," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 103–121, 2017.

[18] L. Moreau, "Aggregation by provenance types: A technique for summarising provenance graphs," in *Graphs as Models 2015 (An ETAPS'15 workshop)*. London, UK: Electronic Proceedings in Theoretical Computer Science, apr 2015, pp. 129–144.

[19] D. Kohan Marzagão, T. Huynh, and L. Moreau, "Incremental inference of provenance types," in *8th International Provenance and Annotation Workshop (IPAW'20) - Forthcoming*, 2020.

[20] R. Souza, L. G. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. V. Brazil, M. Moreno, P. Valduriez, M. Mattoso, R. Cerqueira, and M. A. S. Netto, "Workflow Provenance in the Lifecycle of Scientific Machine Learning," *Concurrency and Computation: Practice and Experience*, Aug. 2021. [Online]. Available: https://hal-lirmm.ccsd.cnrs.fr/lirmm-03324881

[21] H. Miao, A. Li, L. S. Davis, and A. Deshpande, "Towards unified data and lifecycle management for deep learning," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 571–582.

[22] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.

[23] G. Nikolentzos, G. Siglidis, and M. Vazirgiannis, "Graph kernels: A survey," *arXiv preprint arXiv:1904.12218*, 2019.

[24] K. Borgwardt, E. Ghisu, F. Llinares-López, L. O'Bray, and B. Rieck, "Graph kernels: State-of-the-art and future challenges," 2020.

[25] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proceedings. Fifth IEEE International Conference on Data Mining*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2005, pp. 74–81. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICDM.2005.132

[26] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[27] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt, "Scalable kernels for graphs with continuous attributes," in *Advances in neural information processing systems*, 2013, pp. 216–224.

[28] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning theory and kernel machines*. Springer, 2003, pp. 129–143.

[29] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[30] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[31] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, jan 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4700287/

[32] R. Sato, "A survey on the expressive power of graph neural networks," *arXiv preprint arXiv:2003.04078*.

[33] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[35] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2018.

[36] D. Tena Cucala, B. Cuenca Grau, E. V. Kostylev, and B. Motik, "Explainable gnn-based models over knowledge graphs," 2022.

[37] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "Struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 385–394. [Online]. Available: https://doi.org/10.1145/3097983.3098061

[38] P. Rosso, D. Yang, and P. Cudré-Mauroux, "Beyond triplets: Hyper-relational knowledge graph embedding for link prediction," in *Proceedings of The Web Conference 2020*, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1885–1896.

[39] P. Ristoski and H. Paulheim, "Semantic web in data mining and knowledge discovery: A comprehensive survey," *Journal of Web Semantics*, vol. 36, pp. 1–22, 2016.

[40] ——, "RDF2Vec: RDF graph embeddings for data mining," in *International Semantic Web Conference*, P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, and Y. Gil, Eds. Springer International Publishing, 2016, pp. 498–514.

[41] U. Lösch, S. Bloehdorn, and A. Rettinger, "Graph kernels for rdf data," in *The Semantic Web: Research and Applications*, E. Simperl, P. Cimiano, A. Polleres, O. Corcho, and V. Presutti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 134–148.

[42] G. K. D. De Vries and S. De Rooij, "Substructure counting graph kernels for machine learning from rdf data," *Journal of Web Semantics*, vol. 35, pp. 71–84, 2015.

[43] D. Yang, P. Rosso, B. Li, and P. Cudre-Mauroux, "Nodesketch: Highly-efficient graph embeddings via recursive sketching," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1162–1172.

[44] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 701–710. [Online]. Available: https://doi.org/10.1145/2623330.2623732

[45] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 855–864. [Online]. Available: https://doi.org/10.1145/2939672.2939754

[46] S. Hido and H. Kashima, "A linear-time graph kernel," in *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ser. ICDM '09. Los Alamitos, CA, USA: IEEE Computer Society, dec 2009, p. 179–188.

[47] F. Costa and K. De Grave, "Fast neighborhood subgraph pairwise distance kernel," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, pp. 255–262.

[48] G. K. D. de Vries, "A fast approximation of the Weisfeiler-Lehman graph kernel for RDF data," in *Machine Learning and Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 606–621.

[49] W. Ye, Z. Wang, R. Redberg, and A. Singh, "Tree++: Truncated tree based graph kernels," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1778–1789, 2021.

[50] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 23, no. 2, pp. e177–e183, 2007.

[51] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," ser. Proceedings of Machine Learning Research, D. van Dyk and M. Welling, Eds., vol. 5. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, 16–18 Apr 2009, pp. 488–495. [Online]. Available: http://proceedings.mlr.press/v5/shervashidze09a.html

[52] M. Sugiyama and K. Borgwardt, "Halting in random walk kernels," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1639–1647. [Online]. Available: https://papers.nips.cc/paper/5688-halting-in-random-walk-kernels

[53] T. Kataoka and A. Inokuchi, "Hadamard code graph kernels for classifying graphs," in *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM,*, INSTICC. SciTePress, 2016, pp. 24–32.

[54] A. Berlinet and C. Thomas-Agnan, *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Boston, MA: Springer, 2011.

[55] A. E. W. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark, "MIMIC-III, a freely accessible critical care database," *Scientific Data*, vol. 3, no. 1, p. 160035, 2016.

[56] S. D. Ramchurn, T. D. Huynh, M. Venanzi, and B. Shi, "CollabMap: Crowdsourcing maps for emergency planning," in *5th ACM Web Science Conference (WebSci '13)*, 2013, pp. 326–335.

[57] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, and M. Vazirgianis, "GraKeL: A Graph Kernel Library in Python," *Journal of Machine Learning Research*, vol. 21, no. 54, pp. 1–5, 2020. [Online]. Available: http://jmlr.org/papers/volume21/18-370/18-370.pdf

[58] N. M. Kriege, P.-L. Giscard, and R. Wilson, "On valid optimal assignment kernels and applications to graph classification," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 1623–1631.

[59] G. D. S. Martino, N. Navarin, and A. Sperduti, "A tree-based kernel for graphs," in *Proceedings of the 2012 SIAM International Conference on Data Mining*, 2012, pp. 975–986.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[61] M. H. DeGroot and M. J. Schervish, *Probability and Statistics*, 4th ed. Pearson, 2012.

[62] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1135–1144. [Online]. Available: https://doi.org/10.1145/2939672.2939778

[63] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *Advances in neural information processing systems*, vol. 20, 2007.

[64] F. Wesel and K. Batselier, "Large-scale learning with fourier features and tensor decompositions," *Advances in Neural Information Processing Systems*, vol. 34, pp. 17 543–17 554, 2021.

[65] D. Richardson and L. Moreau, "Towards the domain agnostic generation of natural language explanations from provenance graphs for casual users," in *6th International Provenance and Annotation Workshop (IPAW'16)*, McLean, VA, US, Jun. 2016, pp. 1–12.