Finding the Most Descriptive Substructures in Graphs with Discrete and Numeric Labels

Michael Davis, Weiru Liu, and Paul Miller

Centre for Secure Information Technologies (CSIT),
School of Electronics, Electrical Engineering and Computer Science,
Queen's University, Belfast, United Kingdom
{mdavis05, w.liu}@qub.ac.uk, p.miller@ecit.qub.ac.uk

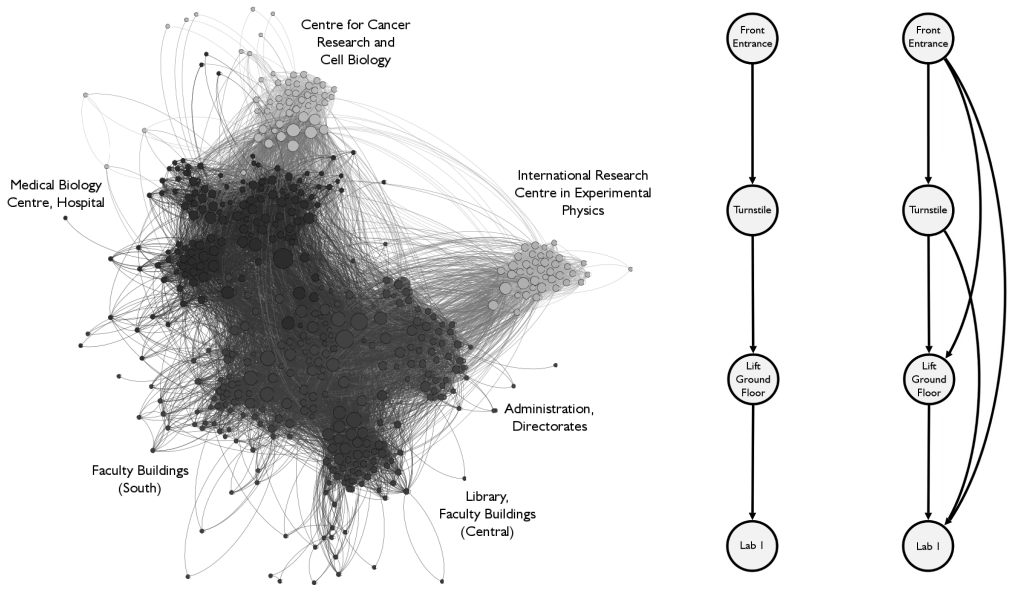
Abstract. Many graph datasets are labelled with discrete and numeric attributes. Frequent substructure discovery algorithms usually ignore numeric attributes; in this paper we show that they can be used to improve discrimination and search performance. Our thesis is that the most descriptive substructures are those which are normative both in terms of their structure and in terms of their numeric values. We explore the relationship between graph structure and the distribution of attribute values and propose an outlier-detection step, which is used as a constraint during substructure discovery. By pruning anomalous vertices and edges, more weight is given to the most descriptive substructures. Our experiments on a real-world access control database returns similar substructures to unconstrained search with 30% fewer graph isomorphism tests.

Keywords: graph mining, frequent substructure discovery, numeric attributes, outlier detection

1 Introduction

A common task in graph mining is to discover frequently-occurring substructures for concept learning, clustering or anomaly detection. Frequent substructures are defined as those which pass some minimum support threshold [10, 13, 18] or in information-theoretic terms, as the patterns which can be used to maximally compress the input graph [5]. In this paper, we consider how numeric attributes can be combined with structural data, to constrain the search for the most descriptive substructures.

To count the frequency of each pattern, discovery algorithms must compare subgraphs for identity, or Graph Isomorphism (GI). GI is computationally complex to decide for the general case [8], but in practice the complexity is highly dependent on the features of the graphs under consideration. Common special cases can be solved in polynomial time, using techniques such as sorting candidate substructures by their canonical labels [10, 13], organising discovered subgraphs into spanning trees [2, 18] or by performing a heuristic search [5]. However, it is always possible to come up with a set of input graphs where even the best algorithms perform poorly.



(a) Single large graph (b) Simple transaction (c) Forward edges

Fig. 1: Graph of Access Control Transactions on a University Campus

Substructure discovery algorithms typically operate on graphs where the labels represent discrete attributes of vertices or edges. Many graph datasets also contain numeric labels or weights, representing attributes such as size, distance, time, frequency or amount. We propose that the best substructures are not only frequent, but also have the most normative numeric attributes. In this paper, we analyse how the dependencies between graph structure and attributes affect the complexity of substructure discovery. Our contribution is to show how numeric attributes can be used to constrain the search in a way that preserves anti-monotonicity. By pruning unlikely candidates early, we focus computational resources on the most descriptive patterns. In our experiments, we were able to discover the most descriptive substructures with a 30% reduction in the number of GI tests required, compared to an unconstrained search.

Applications of substructure discovery include: discovering molecular structures from chemical compounds, *e.g.* for predictive toxicology; learning communication patterns in an e-mail network; and video scene analysis. Our motivating application is to detect "suspicious" behaviour patterns in secure buildings, such as airports, hospitals and power stations. Our experimental data is from the building access control system for a university campus, represented by the graph in Fig. 1a. Vertices represent door sensors and directed edges represent movements between pairs of sensors. The density of transactions is higher in areas with greater security requirements, *viz.* laboratories for laser, radiation and medical research. As we are interested in paths taken by individuals through the network, we reorganised the graph as a transaction database, where each subgraph represents the movement of an individual within a given day (Fig. 1b–1c). We score each subgraph based on its structural elements and its numeric

timing values [6]. Our method shows greater discrimination than scoring based on frequent substructures without numeric attributes.

This paper is organised as follows: Sect. 2 is a brief survey of substructure discovery in graphs and constraint-based graph mining. In Sect. 3, we explore how the distribution of labels and attributes affects the complexity of substructure discovery. In Sect. 4, we outline a method of using numeric outlier detection to constrain the search for normative substructures. Sect. 5 presents our experimental method and datasets (including a method for generating random graph attributes), results and a discussion of complexity; and conclusions are in Sect. 6.

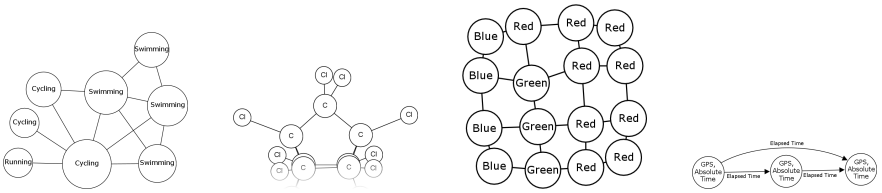
2 Related Work

Frequent Substructure Discovery Frequent substructure discovery algorithms attempt to find the subgraphs which occur most frequently in a graph database. Early approaches were based on Apriori-style itemset mining: AGM [10] and FSG [13] generate candidate substructures by growing them one vertex or one edge at a time, respectively. Frequent substructures are those which exceed a specified minimum support threshold. The main weakness is that candidate generation is expensive, as canonical labels must be calculated for a large number of redundant candidates.

gSpan [18] avoids candidate generation. Canonical labels are determined by the minimum representation of vertex orderings as discovered by a Depth-First Search (DFS). These labels are organised into a hierarchical spanning tree. Frequent structures are discovered by traversing this tree, checking for substructures which exceed minimum support. CloseGraph [19] and SPIN [9] improve on gSpan by mining only "closed" or "maximal" frequent subgraphs, *i.e.* frequent substructures which are not part of any larger frequent substructure. [2] generalises the canonical form found in gSpan and demonstrates that canonical labels based on Breadth-First Search (BFS) are equally valid.

Subdue [5] represents another class of substructure discovery algorithm, based on information theory. Rather than searching for substructures with minimum support, Subdue looks for the substructures which can be used to best compress the input graph based on the Minimum Description Length (MDL) principle. Complexity is managed with a heuristic: candidate substructures are discovered using a greedy beam search (a limited-length queue of the best few patterns found so far). This allows Subdue to search in single large graphs, which is not generally possible with AGM, FSG and gSpan. The disadvantage of the greedy search strategy is that some interesting patterns could be missed.

Constraint-based Graph Mining One of the main problems with pattern-mining algorithms is the large numbers of patterns produced. One possible solution is to introduce *constraints*, which are used to prune away uninteresting patterns and focus on the most meaningful subgraphs. [15] defines a "cohesive pattern constraint" on a connected subgraph, where the vertex attributes are the same within some subspace and some density constraint is met. The cohesive



(a) Social graph (b) Molecular structure (c) Scene Analysis (d) Sensor Network

Fig. 2: Examples of the distribution of attributes on various kinds of graph

pattern constraint is defined for graphs with discrete attributes; in this paper, we define a constraint on graphs with numeric attributes.

In [11], gSpan is extended by including edge weights into the support calculation and pruning low-weighted substructures from the search. Anti-monoticity is an important property of pattern growth-based algorithms: if a substructure fails to achieve minimum support, all of its supergraphs will also fail. Two of the proposed weighting schemes preserve anti-monotonicity, by thresholding the weight measure in addition to thresholding for minimum support. A third weighting scheme uses a heuristic and does not rely on preserving anti-monotonicity. All three weighting schemes assume that higher weights are more significant (which may not necessarily be the case).

Constraints on weighted graphs are considered within a more general framework in [7]. Attribute-based constraints (which are not guaranteed to be anti-monotonic) are used to prune substructures by running a measure function on the edge (or vertex) weights and comparing the output to a threshold. If the definition of the measure function is extended to take multi-dimensional numeric attributes as its input, then the outlier detection step that we propose in Sect. 4 could be considered a measure function within this theoretical framework.

3 Graph Mining with Attribute-based Constraints

Previous work on attribute-based constraints has assumed independence between the structure of a graph and its attributes. If this is the case, then attribute-based constraints are not anti-monotonic. However, in most real-world graphs, the structure and attributes are not independent. In this section, we discuss a graph/attribute model where the attributes of vertices and edges are conditionally dependent on the attributes of similar or neighbouring vertices and edges in the graph. In Sect. 4, we will present a method to constrain substructure discovery based on the values of numeric attributes.

Fig. 2 shows four examples to illustrate the dependence of graph structure and attributes. The social graph in Fig. 2a is labelled with the favourite sport of the actors: people tend to form friendship bonds with people of similar interests. In molecules (Fig. 2b), the vertex label (atom name) is conditionally

dependent on the molecular structure; the degree of each vertex is dependent on the number of free electrons of each atom; and the length of the edges (bonds) is dependent on the atomic weights of the vertices. Fig. 2c is from video scene analysis, showing a green object moving across a red and blue background. Each vertex represents a superpixel in the frame: the colour attribute is conditionally dependent on the colour of adjacent vertices. The velocities of adjacent vertices are also conditionally dependent, as the superpixels in the object will move together and those of the background will move together. Fig. 2d continues the example shown in Fig. 1: the time taken to travel between a pair of sensors is conditionally dependent on the name and GPS coordinates of the sensor.

All of the above examples are labelled graphs, which have an arbitrary number of discrete and numeric labels on their vertices and edges. Formally:

Definition 1. *A labelled graph G is a tuple $\langle V, E, L, \mathcal{L}_V, \mathcal{L}_E \rangle$. V is a set of vertices and E is a set of edges: $E \subseteq \{\langle v, w \rangle : v, w \in V \times V\}$. If the tuple $\langle v, w \rangle$ is ordered, the edge is directed, otherwise it is undirected. L is a set of graph labels; $\mathcal{L}_V$ and $\mathcal{L}_E$ are label-to-value mapping functions.*

Definition 2. *The set of graph labels L is the union of the sets of vertex labels L_V and edge labels L_E. L is partitioned into discrete labels L^D and numeric labels L^N, $L^D \cap L^N = \emptyset$. Thus $L = L_V \cup L_E = L^D \cup L^N$. Let A^D be the set of discrete attribute values and $A^N \subset \mathbb{R}$ be the set of numeric attribute values.*

Definition 3. *The label-to-value mapping function for vertices is denoted as:*

$$\begin{aligned} \mathcal{L}_V : V \times (L_V \cap L^D) &\rightarrow A^D \\ V \times (L_V \cap L^N) &\rightarrow A^N \end{aligned}$$

For a vertex-weighted graph, the weight function $\mathcal{W}(v)$ is treated as a special case of its numeric attributes: $\forall v \in V : \mathcal{W}(v) = \mathcal{L}_V(v, \text{“}weight\text{”})$. (The label-to-value mapping function for edges $\mathcal{L}_E$ can be denoted in a similar manner.)

During substructure discovery, one important method of reducing the complexity of the GI test is vertex partitioning [8, 10, 13, 18]. Vertices can be partitioned into similar disjoint sets or equivalence classes. We extend this notion to also define edge partitions:

Definition 4. *The vertex partition set and edge partition set are defined as:*

$$V = \bigcup_i V_i \qquad E = \bigcup_i E_i$$

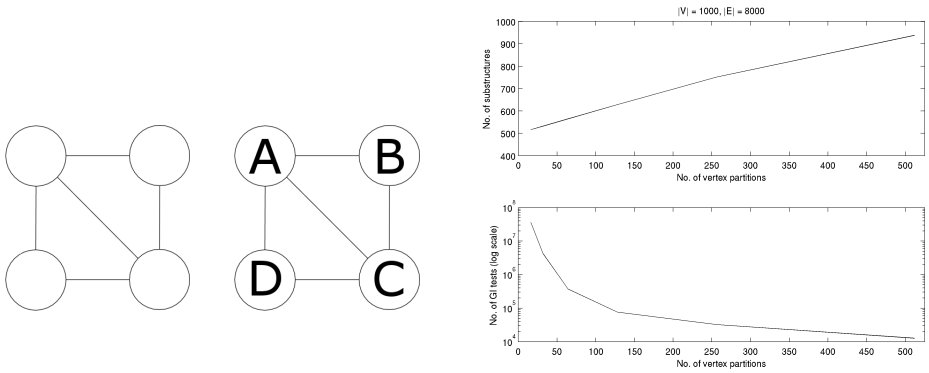
where all vertices in the same partition share the same discrete attribute values:

$$\forall v \in V_i \quad \forall w \in V_i \quad \forall l \in (L_V \cap L^D) : \quad \mathcal{L}_V(v, l) = \mathcal{L}_V(w, l)$$

Similarly, all edges in the same partition share the same discrete attribute values, with the additional constraint that their source and target vertices are from the same partitions:

$$\forall \langle v, w \rangle \in E_i \quad \forall \langle x, y \rangle \in E_i : \quad v \in V_j \wedge x \in V_j \wedge w \in V_k \wedge y \in V_k$$

In the case of an undirected graph, $\langle v, w \rangle \Leftrightarrow \langle w, v \rangle$



(a) Unlabelled (b) Labelled (c) Increasing no. of vertex partitions[1]

Fig. 3: Relationship between number of vertex partitions and complexity

There is an important relationship between the number of partitions and the complexity of GI and substructure discovery. To illustrate this, consider the unlabelled partial clique in Fig. 3a. This graph contains eight distinct subgraphs with two or more vertices; 27 subgraph instances in all. As multiple subgraph instances share the same vertex partition set, it requires 20 GI tests to determine which instances are isomorphic. Compare this to the graph in Fig. 3b, which has the same structure, but each vertex is uniquely labelled. Once again there are 27 substructure instances, but each has a distinct vertex and edge partition set. Subgraphs with different partition sets cannot be isomorphic, so we can determine that there are 27 distinct subgraphs without needing to do any isomorphism tests.

In practice, real-world graphs will lie somewhere between these extremes. Fig. 3c shows the relationship between the number of vertex partitions and the complexity of substructure discovery: the number of distinct subgraphs rises with the number of partitions, but the number of instances of each subgraph falls. This leads to an exponential reduction in the number of GI tests and thus the complexity of substructure discovery.

In addition to reducing the number of GI tests required, increasing the number of partitions constrains the search by reducing the support for each substructure. Formally, we define a constraint as follows:

Definition 5. *A constraint c is a Boolean predicate which any subgraph $g \in G$ must fulfil. c is said to be anti-monotone if it satisfies* $\forall g' \subset g : c(g) \implies c(g')$.

An example of an anti-monotone constraint is minimum support: a graph g can reach minimum support only if all of its subgraphs g' reach minimum support.

[1] Fig. 3c shows the results for R-MAT random graphs, with $0, 1, \ldots, 9$ binary labels, *i.e.* 0–512 vertex partitions. The experiment was repeated 10 times (and across multiple sizes of graph) and the results averaged. The attribute values in Fig. 3c were assigned independently from a uniform distribution. Our experiments on synthetic and real datasets (Sect. 5) verify that the complexity of substructure discovery increases with the homogeneity of vertices and edges, and that this holds when the independence assumption is removed.

In previous work on constraint-based graph mining [7, 11], it has been assumed that the structure of the graph and its attributes are independent, and therefore attribute-based constraints are not anti-monotonic. However, in real-world graphs, the independence assumption does not hold (Fig. 2).

Our approach is to consider the dependencies between attributes as a Random Field [1]: the discrete attribute values on each vertex (or edge) are dependent on its adjacent vertices, but are conditionally independent of the rest of the graph. We use the conditional independence (CI) assumption as the basis of a generator for graph attributes (see Sect. 5). In the next section, we use the CI assumption to define a constraint on numeric attribute values: as attributes depend on graph structure, the constraint is used to prune instances, or reduce support for specific substructures, so the property of anti-monotonicity is preserved.

4 Frequent Substructure Discovery with Numeric Attribute Constraints

In the previous section, we discussed the relationship between the number of discrete partitions and the complexity of substructure discovery. In this section, we discuss how to use numeric attributes as a constraint during substructure discovery. We have argued that graph attributes are dependent on graph structure. Therefore, we can define the most descriptive substructures as those which are normative both in terms of their structure and in terms of their numeric attributes. The corollary is that vertices or edges containing numeric outliers are abnormal and can therefore be pruned early in the discovery process. We determine whether numeric attributes are "normal" or anomalous by means of a numeric outlier detection function:

Definition 6. *We define a numeric outlier function $\mathcal{O}$ on a dataset D as:*

$$\mathcal{O} : D \to \mathbb{R} \qquad \forall d \in D : \mathcal{O}(d) = \begin{cases} q_0 & \textit{if } d \textit{ is "normal" w.r.t. } D \\ q & \textit{otherwise} \end{cases}$$

where q_0 is some constant value and $q \neq q_0$ is a value measuring the degree of outlierness.

The value of q_0 and the range of $\mathcal{O}$ will depend on the specific choice of outlier detection function. For our experiments in Sect. 5, we chose Local Outlier Factors (LOF) [3]. LOF is a density-based measure: the LOF score of a sample p is a measure of its outlierness with respect to its local neighbourhood, computed as:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{|N_{MinPts}(p)|}$$

where N is the number of samples in the dataset, $MinPts$ is the minimum number of points to consider as the local neighbourhood and lrd is a function

which computes the local reachability density of the neighbourhood (the inverse of the average reachability distance in the neighbourhood). Intuitively, the LOF score is based on the distance of a sample from its local neighbourhood and the relative density of the neighbourhood. A sample d belonging to a dense cluster or deep within a sparse cluster has $\text{LOF}(d) \approxeq 1$. Outliers have LOF values several times larger. Thus LOF satisfies the property given in Def. 6: $\text{LOF}(d) \approxeq 1$ for normal values of d and $\text{LOF}(d) \gg 1$ for anomalous values. LOF is well-suited to unsupervised learning, as it makes no assumptions about the underlying distribution of the data and can cope with clusters of different sizes and densities. For more discussion on the choice of outlier function, see the comments in Sects. 5 and 6 and our previous work [6].

To calculate $\mathcal{O}(d_v)$ for a vertex v, we define d_v as a multi-dimensional feature vector across all the numeric attributes of v: $d_v = \mathcal{L}_V(v, L^N)$. The outlier factor for each d_v is calculated relative to the dataset defined by its vertex partition, $\forall d_v \in D_i : v \in V_i$.

To use $\mathcal{O}$ as a constraint on substructure discovery, structural elements are classified as normal or anomalous by Def. 7:

Definition 7. *A vertex $v \in V_i$ is normal if $\mathcal{O}(\mathcal{L}_V(v, L^N)) \approxeq q_0$, anomalous otherwise. An edge $e \in E_i$ is normal if $\mathcal{O}(\mathcal{L}_E(e, L^N)) \approxeq q_0$, anomalous otherwise.*

During substructure discovery, anomalous vertices and edges are pruned from the graph. This can be done as a pre-processing step before generating all frequent 1- and 2-vertex subgraphs. As we only consider elements with "normal" numeric values to be part of normative substructures, this pruning dramatically reduces the number of GI tests required, without significantly affecting which substructures are discovered. We validate this experimentally in the next section.

5 Experiments

The purpose of our experiments is to analyse the effect of pruning anomalous vertices and edges from graphs on the performance and accuracy of substructure discovery. Specifically, we analysed the number of graph isomorphism tests required; the computation time for substructure discovery; and the accuracy and meaningfulness of the discovered substructures. We show that our constraint-based approach can more efficiently find frequent subgraphs than an unconstrained approach. In many cases where the input graph is intractable with an unconstrained approach due to the computational or memory overheads, our approach allows the graph to be processed.

Synthetic Datasets Our experimental setup included Erdős-Rényi random graphs and R-MAT random graphs with up to 10,000 vertices. For the R-MAT graphs, we added edges with mean degree 2, 4, 6 and 8 and probabilities that an edge is placed in one of the four quadrants of the graph as $\langle 0.57, 0.19, 0.19, 0.05 \rangle$, to ensure that the random graphs exhibited clustering/community properties. Attributes were added in three ways: random selection from a uniform distribution ("white noise"); according to pre-defined prior probabilities; and according

to the generative algorithm described below. The purpose of the experiments on synthetic data was to compare the effect of increasing size, structure, density and dependencies between graph structure and attributes on the complexity of substructure discovery, and to evaluate the effect of our constraint-based approach on the performance of substructure discovery.

Random Generation of Graph Attributes The R-MAT graph generator [4] creates random graphs with properties similar to many real-world graphs (small-world, power-law degree distribution, *etc.*). As R-MAT creates unlabelled graphs, we had to devise a way to assign labels and attribute values. A naïve approach is to randomly allocate attribute values according to some prior distribution, but this assumes that the graph structure and attributes are independent. Here we present AttributeGen, an algorithm to generate random attribute values on an unlabelled graph. Alg. 1 shows the version for discrete attributes.

The Random Field model assumes that the attribute values on each vertex depend on its neighbours. For AttributeGen, we used a simpler assumption, that the values on each vertex depend only on its higher-degree neighbour; *i.e.*, the dependencies between attributes will be propagated from hubs to leaves. Vertices with no higher-degree neighbour are assigned values from a prior distribution over A (line 6: we define one distribution per vertex label). Edges and the other vertices are assigned values from a posterior distribution over A. Edge attributes are conditional on the vertex partition of the source vertex (line 8). Vertex attributes on a target vertex are conditional on the vertex partition of the source vertex and the edge partition of the connecting edge (line 10).

Random numeric values are generated by an analagous method. From our analysis of numeric attributes on real-world datasets (see below), we see that numeric attributes are generated by multiple processes; the specific mixture of processes depends on the vertex (or edge) partition. We use this observation to extend Alg. 1 to generate numeric as well as discrete attribute values. Each numeric attribute $\mathcal{L}_V(v, l)$ on a given vertex partition $V_i : v \in V_i$ is modelled as

Algorithm 1 Discrete AttributeGen

Require: Unlabelled Random Graph $G = \langle V, E \rangle$, Labels L_V, L_E, Attribute Values A, Prior distributions P_V, Posterior distribuitions Q_V, Q_E

1: Define vertex labels in G from L_V and edge labels from L_E
2: Sort vertices V by degree in descending order
3: **for all** $v \in V$ in order **do**
4: **for all** $l \in L_V$ **do**
5: **if** $\mathcal{L}_V(v, l)$ is unassigned **then**
6: Assign $\mathcal{L}_V(v, l) \leftarrow a \in A$: a is randomly selected from $p_l \in P_V$
7: **for all** $e \in E : e$ is adjacent to $v, \quad l \in L_E$ **do**
8: Assign $\mathcal{L}_E(e, l) \leftarrow a \in A$: a is randomly selected from $q_{v,l} \in Q_E$
9: Let w be the adjacent vertex: $\exists w \in V : e = \langle v, w \rangle$
10: Assign $\mathcal{L}_V(w, l) \leftarrow a \in A$: a is randomly selected from $q_{v,e,l} \in Q_V$

a mixture of Gaussian processes:

$$P_{V_i,l} = \sum_j \omega_j \cdot \eta_j(\mu_j, \sigma_j)$$

similar to those illustrated in Fig. 5. ω_j is the weight of each component in the mixture; $\sum_j \omega_j = 1$ forms a probability distribution Ω over all the components. Numeric attributes are assigned by randomly selecting component j of the mixture from Ω, then choosing a random numeric value from the Gaussian distribution $\eta_j(\mu_j, \sigma_j)$. Thus the distributions of numeric values are dependent on the vertex partition and conditionally independent of the rest of the graph.

Ideally, we should learn the prior and posterior distributions of the attribute values from data, to create attribute generators for specific types of graph (social graphs, molecular structures, *etc.*). As that was outside the scope of this work, the distributions for the synthetic data in the experiments were created manually.

Access Control System Dataset Our real-world dataset is from the access control system logs of a large university campus. The $\approx$ 1 million log entries are graphically represented in Fig. 1a, showing the movements of approximately 6,500 students and staff. The $\approx$ 800 vertices represent door sensors; directed edges represent movements between pairs of sensors. We are interested in finding patterns representing "suspicious" behaviour, particularly in high-security areas such as laboratories for laser, radiation and medical research [6]. For the purpose of our experiments, we reorganised the graph as a transaction database, where each graph transaction represents the movement of an individual within a given 24-hour period (Fig. 1b). If a user fails to swipe in at a particular sensor (*e.g.* if someone holds open a door for them), this creates missing edges in the graph. We compensated for this effect by including forward edges from each sensor to all subsequent sensors visited by the user (Fig. 1c).

Numeric attributes were calculated from the log entries and added to the edges in the graph as shown in Fig. 4. Absolute time is the time of day (seconds since midnight) when the user presented their ID card to a door sensor at the end of the path segment. Elapsed time is the difference in seconds between the absolute time at the current sensor and the absolute time at the previous sensor. Day of Week (DoW) is strictly an ordinal attribute, but it was convenient to represent it numerically: LOF combined it with the other attributes in multi-dimensional space, effectively clustering different patterns of behaviour on different days. Weekend patterns are quite different from weekday patterns, perhaps representing the movements of security staff or cleaners as well as weekend workers.

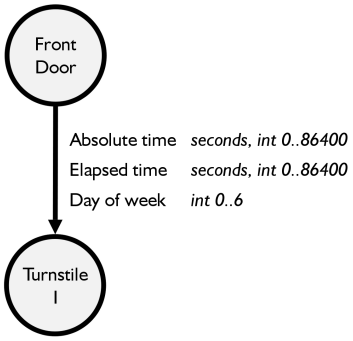


Fig. 4: Numeric edge labels

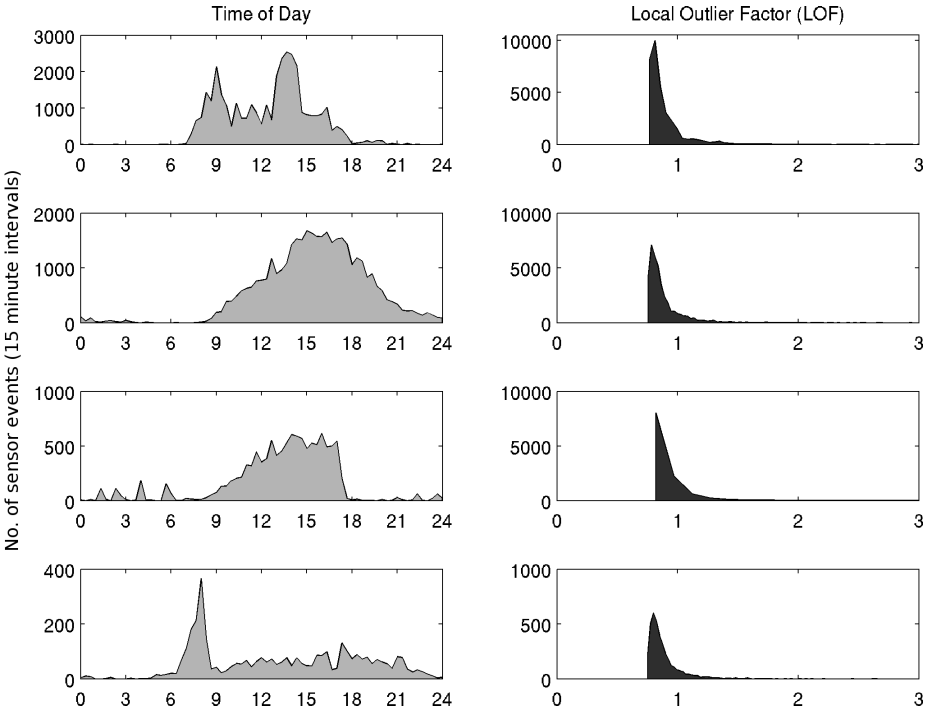
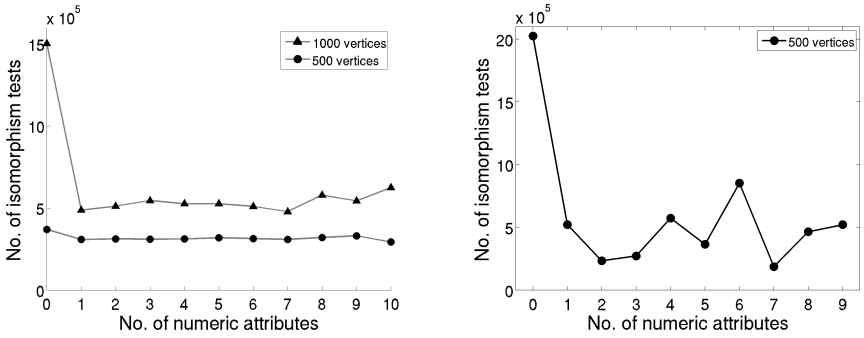


Fig. 5: Distribution of Empirical Data and Local Outlier Factors

We cannot assume that the numeric attributes within real-world graphs are a simple Gaussian; nor can we assume that the probability distribution is the same across all partitions. The left-hand side of Fig. 5 shows four examples of the distribution of the Time of Day numeric attribute across the edges of four of the edge partitions. For each partition, there is a mixture of underlying processes: in a lab, some people work there and stay for many hours; others go in simply to speak to a colleague for a few minutes; security staff may show up periodically for short intervals in the middle of the night. The mixture of processes between partitions is also different: the behaviours in a lab are very different from the behaviours in a lift.

This analysis supports our decision to use a density-based approach to calculate numeric outliers. The distribution of LOF scores for each of the empirical distributions is shown to the right of Fig. 5. Although the data distributions are very different, the distributions of LOF scores are very similar, with normal values clustered around 1 and anomalous values stretching out in a long tail to the right. This verifies that by using LOF, we do not have to make any assumptions about the underlying distribution of the data.

The experimental results in the next section verify that LOF is an effective measure for pruning anomalous vertices and edges.



(a) Independent Attributes

(b) Conditionally Independent Attributes

Fig. 6: Frequent substructure discovery on R-MAT random graphs

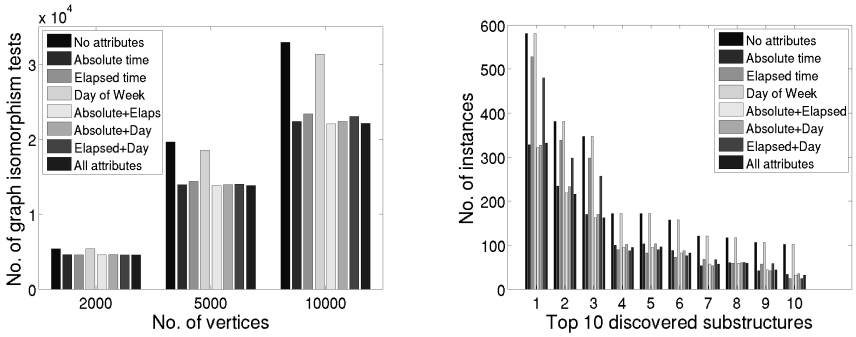
5.1 Results

Synthetic Datasets The experiments on random single large graphs were repeated 10 times for each combination of no. of vertices, no. of vertex partitions and no. of numeric attributes, and the results averaged. In all cases, run time was directly proportional to the number of GI tests. The time taken to calculate numeric outliers was trivial compared to the time to discover substructures: ≈ 0.2 seconds for a graph with 10,000 vertices and 100 numeric attributes. LOF's $O(n^2m)$ complexity is acceptable on graphs where each partition has up to a few thousand vertices or edges. For graphs with larger partitions, some alternative measures are suggested in Sect. 6.

Fig. 6a shows the results for graphs where the attributes were assigned independently from a uniform distribution. On the 1,000 vertex graphs, we measured an average 66% reduction in the number of GI tests required when using numeric outliers to prune the graph, compared to an unconstrained search. (As the attributes are independent, we cannot make any claims about the meaningfulness of the discovered substructures in this case; *cf.* our comparison with random substructure removal, Fig. 8).

Fig. 6b shows the results for graphs where the attributes are conditionally dependent on the graph structure (Alg. 1), so the anti-monotone condition holds. These experiments show an average 80% reduction in the number of GI tests, demonstrating that our constraint-based approach is most effective when the graph exhibits conditional independence between structure and attributes.

We found that graphs with 10,000 vertices and a small number of partitions were not tractable without using numeric attributes, as there were millions of instances of each pattern, requiring more memory to process than was available in our experimental setup. However, we were able to process the 10,000 vertex graphs with the numeric constraint, as there were an order of magnitude fewer instances to compare for isomorphism. This suggests that our method could be useful when processing Very Large Graphs.



(a) Number of GI tests required (b) Frequency of top 10 substructures

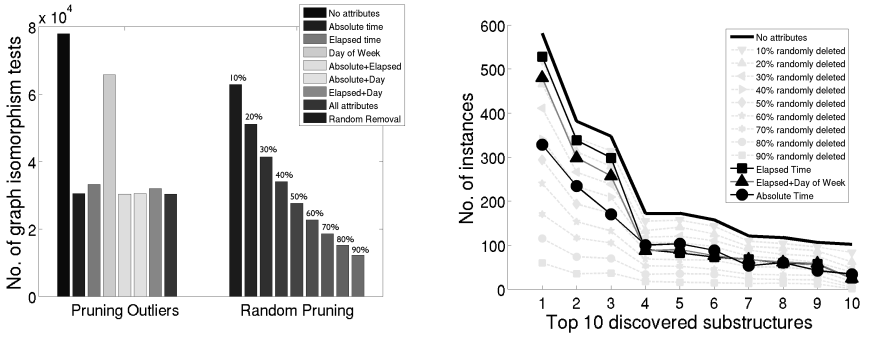
Fig. 7: Effect of pruning numeric anomalies on frequent substructure discovery

Access Control System Dataset The results on the real-world graph transaction database are shown in Fig. 7. Fig. 7a shows the number of GI tests for datasets of 2,000–10,000 vertices. In the real-world data, numeric attributes vary in their ability to discriminate between normal and anomalous patterns, but combining attributes gives the best performance overall. Absolute and elapsed time are both good discriminators, but day of week is very poor. However, if we combine day of week with elapsed time, we get a slightly better result than using elapsed time on its own; and the best results were achieved by combining all three attributes. The benefit of our approach increased with increasing size of database, as we were able to prune more anomalous substructures. In the dataset with 10,000 vertices, we reduced the number of GI tests by around 30%, which equated to a speed-up of 1.45.

Next, we wanted to validate that the discovered substructures are meaningful. Fig. 7b compares the ten best substructures discovered by Subdue (with no numeric attributes) to the substructures discovered when we added attributes. Our approach discovered the same substructures as Subdue, but fewer instances of each. The relative order of the top ten substructures was changed slightly. Where there is a large difference in relative frequency (*e.g.* 1^{st}–3^{rd} substructures), the ordering was unchanged: these substructures are robust against the removal of anomalous edges. In cases where the relative frequencies were very similar, the order was sometimes transposed (*e.g.* 4^{th} and 5^{th} substructures exchanged places). This is because greater weight is given to substructures with normal numeric values.

To investigate this effect further, we compared our method of pruning anomalous edges to random removal of edges from the graph. We conducted experiments where we randomly deleted 10%–90% of the edges in the graph before searching for frequent substructures. The results are shown in Fig. 8.

Fig. 8a shows the effect on performance. It is necessary to remove around 45% of the graph in order to reduce the number of GI tests by a similar amount as our approach.



(a) Number of GI tests required (b) Frequency of top 10 substructures

Fig. 8: Effect of randomly removing graph edges on substructure discovery

Fig. 8b shows the frequency counts of the top 10 substructures. Substructures with edges randomly removed are shown in grey; substructures discovered by our approach are superimposed in black. Randomly removing edges increases the entropy of the graph: the shape of the curve becomes flatter as more of the graph is removed. Substructure 1 is quite distinct; even with 90% edge removal, it remains the most descriptive pattern. Substructures 2–3 are also quite robust; discrimination is lost at around 70% edge removal. However, discrimination between substructures 4–10 diminishes after 20% edge removal and by 50% edge removal, the order is random. These results show that randomly deleting graph edges does not preserve the meaningfulness of the output.

5.2 Analysis of Complexity

Here we briefly discuss the benefits (and costs) of using our approach with respect to the complexity of substructure discovery. Substructure discovery algorithms have essentially two parts: finding instances in a graph (or graph database); and grouping instances together into common substructures for evaluation.

Finding Instances The graph space is searched starting from each vertex instance. If $n = |V|$ and $m = |E|$, an exhaustive search of all possible substructure instances in a single graph has complexity $\mathrm{O}(n^2 + nm)$. For a graph database with K transactions, $\mathcal{D} = \{G_1, \ldots, G_K\}$, the complexity of instance discovery is $\mathrm{O}(n_1^2 + n_1 m_1 + \ldots + n_K^2 + n_K m_K)$. In practice, algorithms do not perform an exhaustive search, as the cost is prohibitive in all but the most trivial graphs. Subdue constrains the search with its parameters; gSpan uses minimum support. gSpan's approach does not scale to single large graphs; graph transactions with more than a few hundred vertices and edges are intractable [11].

The benefit of our approach is to prune away the parts of the search space which contain numeric anomalies. If $0 < p \leq 1$ is the proportion of "normal" vertices and $0 < q \leq 1$ is the proportion of "normal" edges, the complexity of an exhaustive search for substructure instances reduces to $\mathrm{O}((pn)^2 + pqnm)$.

Grouping Instances into Substructures Instances are grouped together using isomorphism tests (or canonical labels, which involves finding a set of automorphisms and determining which is "least"). The complexity of GI is one of the most famous unsolved problems in complexity theory, so a formal analysis is well outside the scope of this paper. The interested reader is referred to [8] for an introduction and to [14] for the state-of-the-art. The best proven worst-case complexity for the general case is $e^{\mathrm{O}(\sqrt{n \log n})}$, though most real-world graphs exhibit polynomial complexity. Suffice to say that the GI test is the most computationally expensive part of the discovery process: any reduction to the number of GI tests will have a dramatic effect on the computational cost.

The upper bound on the number of GI tests required is $\mathrm{O}(|I| \times |S|)$, where I is the set of instances and S is the set of discovered substructures. In practice, we only approach this upper bound where the input graph exhibits low entropy: see the discussion following Def. 4. Our approach reduces $|I|$ as discussed above. There is little effect on the frequent substructures in S (see Fig. 7b, 8b), but infrequent substructures will lose support and will be pruned earlier, so we expect some reduction in $|S|$.

Cost of Calculating Numeric Anomalies LOF has complexity $\mathrm{O}(n^2 d)$, where d is the number of numeric attributes. For a graph with N vertex partitions (Def. 4), the complexity of calculating LOF for all vertices is $\mathrm{O}(n_1^2 d + \ldots + n_N^2 d)$. Thus, the cost of calculating LOF is typically orders of magnitude smaller than the cost of substructure discovery, but rises significantly when the input graph exhibits low entropy (as there are fewer partitions with more vertices in each).

Where each n_i is small (not more than a few thousand), LOF's $\mathrm{O}(n^2 d)$ complexity is acceptable. For very large or very regular graphs, the complexity can be reduced by replacing LOF with an approximation algorithm such as aLOCI [16] or PINN [17], which has sub-quadratic complexity, $\mathrm{O}(dn \log n)$.

In summary, our approach reduces the complexity of substructure discovery at both the instance mining and substructure grouping phases, and the cost of calculating the numeric anomaly scores is typically orders of magnitude less than the cost savings. The resulting performance improvement means that substructure discovery remains tractable for larger graphs than is possible with the standard algorithms.

6 Conclusions

In this paper, we presented a method of using numeric outliers as a constraint on the search for frequent substructures in graphs. Our thesis is that the "best" substructures are those which are not only the most frequent, but which are also normative in terms of their numeric attributes.

Previous work on attribute-based constraints has assumed independence between graph structure and attributes, but this assumption does not hold for real-world graphs. Our outlier-based constraint and algorithm for generating random attributes on graphs assume conditional independence between graph structure and attributes.

Our experiments on random graphs demonstrate that in many cases where the input graph is intractable with an unconstrained approach, our approach allows the graph to be processed. In experiments on real-world data, we find similar substructures to an unconstrained search, with around 30% fewer graph isomorphism tests. Where discovered substructures are of similar frequency, we are better able to discriminate between them, because we give greater weight to substructures with normal numeric attributes.

Future Work The algorithm for generating random attributes (Sect. 5) must be provided with prior and posterior distributions of attribute values. We plan to analyse real-world datasets representing different kinds of graph to learn these distributions. This will allow us to generate different types of random graph which share the characteristics of real-world graphs.

Instead of hard-pruning anomalies, the measure $\mathcal{O}$ could be used in the calculation of how much support each subgraph instance contributes. Instances with normal numeric values would contribute a support of 1, whereas instances with anomalous numeric values would contribute a support of less than 1. We plan to conduct further experiments to compare this alternative support measure with the pruning approach.

We used LOF as the measure function $\mathcal{O}$. As discussed in Sect. 5.2, where the vertex or edge partitions are very large, LOF's complexity may be unacceptable. This could be addressed by replacing LOF with aLOCI [16] or PINN [17].

We tested our approach on graphs with a moderate number of numeric attributes (up to 10). If there are very many attributes, the numeric feature vectors become very sparsely distributed in high-dimensional space. In this case, LOF's ability to discriminate between normal and anomalous values is diminished. However, not all attributes are of equal importance to all clusters. For high-dimensional data, we could amend our approach to detect numeric anomalies in subspaces rather than in full space by choosing only locally-relevant attributes on which to calculate the outlier score [12].

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
2. Borgelt, C.: Canonical forms for frequent graph mining. In: 30th Annual Conf. German Classification Society (GfKl 2006). pp. 337–349. Springer (2006)
3. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: Identifying density-based local outliers. SIGMOD Rec. 29(2), 93–104 (2000)
4. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. In: SDM 2004. SIAM (2004)
5. Cook, D.J., Holder, L.B.: Graph-based data mining. IEEE Intelligent Systems 15, 32–41 (March 2000)
6. Davis, M., Liu, W., Miller, P., Redpath, G.: Detecting anomalies in graphs with numeric labels. In: CIKM 2011. pp. 1197–1202. ACM (2011)
7. Eichinger, F., Huber, M., Böhm, K.: On the usefulness of weight-based constraints in frequent subgraph mining. In: ICAI 2010. pp. 65–78. BCS SGAI (Dec 2010)
8. Fortin, S.: The graph isomorphism problem. Tech. rep., Univ. of Alberta (1996)

9. Huan, J., Wang, W., Prins, J., Yang, J.: SPIN: Mining maximal frequent subgraphs from graph databases. In: KDD 2004. pp. 581–586. ACM (2004)
10. Inokuchi, A., Washio, T., Motoda, H.: An Apriori-based algorithm for mining frequent substructures from graph data. In: PKDD 2000. pp. 13–23. Springer (2000)
11. Jiang, C., Coenen, F., Zito, M.: Frequent sub-graph mining on edge weighted graphs. In: DaWaC 2010. Springer (2010)
12. Kriegel, H.P., Kröger, P., Zimek, A.: Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Transactions on Knowledge Discovery in Data 3(1), 1:1–1:58 (March 2009)
13. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: ICDM 2001. pp. 313–320. IEEE (2001)
14. McKay, B.D., Piperno, A.: Practical graph isomorphism, II (Jan 2013), `http://arxiv.org/abs/1301.1493v1`
15. Moser, F., Colak, R., Rafiey, A., Ester, M.: Mining cohesive patterns from graphs with feature vectors. In: SDM. pp. 593–604 (2009)
16. Papadimitriou, S., Kitagawa, H., Gibbons, P., Faloutsos, C.: Loci: fast outlier detection using the local correlation integral. In: Data Engineering, 2003. Proceedings. 19th International Conference on. pp. 315 – 326 (March 2003)
17. de Vries, T., Chawla, S., Houle, M.: Finding local anomalies in very high dimensional space. In: ICDM 2010. pp. 128–137. IEEE (2010)
18. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: ICDM 2002. pp. 721–724. IEEE (2002)
19. Yan, X., Han, J.: CloseGraph: Mining closed frequent graph patterns. In: KDD 2003. pp. 286–295. ACM (2003)