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

Journal of Intelligent Information Systems manuscript No.
(will be inserted by the editor)

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

Michael Davis · Weiru Liu · Paul Miller

the date of receipt and acceptance should be inserted later

Abstract Many graph datasets are labelled with discrete and numeric attributes. Most frequent substructure discovery algorithms ignore numeric attributes; in this paper we show how they can be used to improve search performance and discrimination. 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 method is applicable to multi-dimensional numeric attributes; we outline how it can be extended for high-dimensional data. We support our findings with experiments on transaction graphs and single large graphs from the domains of physical building security and digital forensics, measuring the effect on runtime, memory requirements and coverage of discovered patterns, relative to the unconstrained approach.

Keywords graph mining · frequent substructure discovery · constraint-based mining · labelled graphs · numeric attributes · outlier detection

1 Introduction

Graph representations are useful in many domains where data has a natural structure which cannot be easily represented as a set of attributes. Graphs are commonly used to represent complex structures such as social, communication or computer networks and biological or ecological processes [32].

A common task in graph mining is to discover frequently-occurring substructures for concept learning, classification, clustering or anomaly detection. Frequent substructures are defined as those which pass some minimum support threshold [17, 27, 42] or in information-theoretic terms, as the patterns which can be used to maximally compress the input graph [8].

Michael Davis · Weiru Liu · Paul Miller
Centre for Secure Information Technologies (CSIT), School of Electronics, Electrical Engineering and Computer Science, Queen's University, Belfast, BT3 9DT, United Kingdom
Tel. +44 28 9097 1700, Fax +44 28 9097 1702
E-mail: mdavis05@qub.ac.uk, w.liu@qub.ac.uk, p.miller@ecit.qub.ac.uk

Most substructure discovery approaches assume discrete graph labels. However, many graph datasets also contain numeric labels or weights, representing counts, frequencies, or attributes such as size, distance or time. In this paper, we restrict our analysis to graphs with numeric attributes.

To count the support of each pattern, discovery algorithms must compare substructures for identity, or Graph Isomorphism (GI). GI is computationally complex to decide for the general case [13], but common special cases can be solved in polynomial time, using techniques such as sorting candidate substructures by their canonical labels [17, 27], organising discovered substructures into spanning trees [4, 42] or constraining the size of substructures to be compared [8]. Even so, GI tests remain the most expensive part of substructure discovery. Empirical studies show that most processing time is spent on operations which rely on GI tests: support computation or calculating embedding lists [41]. Thus it is always desirable to minimise the number of GI tests required. We address this problem with a guided pruning step prior to substructure discovery. Removing a small number of anomalous edges or vertices has the effect of significantly reducing the number of GI tests required.

Our work is based on the following observations:

- The number of GI tests performed during substructure discovery is based not only on the size of the input graph, but is also highly dependent on its structure and attributes, such as edge density and the number and distribution of label values.
- In real-world graphs, graph structure and label values are highly correlated, allowing us to exploit label information to determine the "best" substructures.
- We propose that the best substructures are those which are not only frequent, but also have the most normative numeric attributes.

Therefore, pruning away a few anomalous edges can dramatically reduce the complexity of the input graph while retaining the most descriptive patterns. Our main contributions are:

- We present our method, which is based on a pre-processing step to remove edges with anomalous numeric attributes before the substructure discovery step. The patterns which are discovered are those which have enough instances with normal numeric attributes to meet minimum support. We propose that these are the most descriptive substructures.
- Our experiments compare the coverage of discovered substructures using our method against the unconstrained approach. We show that the top substructures from the unconstrained approach are retained after the pruning step.
- The experiments show that our approach significantly reduces runtime and memory usage for substructure discovery on real-world graphs. By pruning unlikely candidates early, we focus computational resources on the most descriptive patterns. Thus many searches which are intractable using the standard algorithms become tractable using our constraint-based approach.

The paper is organised as follows: Sect. 2 is a brief survey of substructure discovery in graphs, constraint-based graph mining and numeric outlier detection. In Sect. 3, we discuss how the distribution of labels and attributes affects the complexity of the input graph and its effect on substructure discovery. We outline our method in Sect. 4, in which we use numeric outlier detection to prune the input graph. Sect. 5 describes the E-mail and Access Control datasets used in the experiments; results and discussion are in Sect. 6. Conclusions are in Sect. 7.

2 Related Work

2.1 Frequent Substructure Discovery

Early approaches to frequent substructure discovery were based on applying Apriori-style itemset mining to graph databases. AGM [17] and FSG [27] generate candidate substructures by growing them one vertex or one edge at a time, respectively. The *support* of each candidate is equal to the number of graph transactions which contain it; frequent substructures are those which exceed a specified minimum support threshold. Support is an anti-monotone property: the support of a graph cannot exceed that of its subgraphs. The main weakness of AGM and FSG is that candidate generation is expensive, as canonical labels must be calculated for a large number of redundant candidates.

gSpan [42] avoids the problem of generating redundant canonical labels by determining the minimum representation of vertex orderings as discovered by a DFS, then organising the canonical labels into a hierarchical spanning tree. Frequent substructures are discovered by traversing this tree and checking for substructures which exceed minimum support. CloseGraph [43] and SPIN [16] are refinements to gSpan which reduce redundancy by mining only the set of *closed* or *maximal* frequent subgraphs, *i.e.* those which are not part of any larger frequent subgraph.

GASTON [33] operates on the same principles as gSpan—DFS-based canonical ordering and testing for minimum support—but streamlines the search by searching first for sequences, then trees, then graphs. GASTON has been demonstrated to be more efficient than gSpan [41] as it avoids generating duplicate subgraphs, speeding up the support computation.

The frequent subgraph mining algorithms above cannot be applied to single large graphs, because support is calculated from the number of graph transactions which contain the pattern (rather than the number of instances of the pattern).

Subdue [8] represents a different class of substructure discovery algorithm, based on information theory. Instead of counting support, Subdue looks for the substructures which can be used to best compress the input graph based on the Minimum Description Length (MDL) principle. While the frequent subgraph mining approaches exhaustively search all substructures meeting the minimum support criterion, Subdue employs a greedy beam search strategy (a limited-length queue of the best few patterns found so far). This makes it possible to search large graphs; however, Description Length (DL) is not an anti-monotone property. Once the DL of an expanding substructure begins to increase, further expansion of the substructure *typically* does not yield a smaller DL [7], but this is not guaranteed. Thus MDL-based greedy search can miss some interesting patterns, especially if the beam width is too narrow.

Frequent subgraph mining and compression-based approaches are compared in [10]. Subdue and GASTON were able to discover the same patterns. For graph databases, GASTON was found to outperform Subdue; but Subdue was able to handle large and complex graphs which were not possible for GASTON to search.

None of these graph mining approaches make use of numeric attributes or weights, except in a very limited way (treating numeric values as categorical, or discretization based on binning or thresholding). In this paper we use numeric attribute values as a constraint on the search for frequent substructures.

2.2 Constraint-based Graph Mining

Graph mining approaches use *constraints* to prune away uninteresting patterns and focus on the most meaningful subgraphs. The most obvious example of a constraint from the preceding section is minimum support, but other constraints have been proposed.

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

[11] analyses edge weights in a post-processing step. First, graph transactions are mined using CloseGraph. Then entropy-based feature selection is applied to the result set, yielding an ordered list of edges. This approach improves the discriminitiveness of the results, but does not have any effect on search performance.

In [19], gSpan is extended by including edge weights into the support calculation and pruning low-weighted substructures from the search. Two of the proposed weighting schemes preserve the anti-monotone property, by thresholding the weight measure in addition to thresholding for minimum support. A third weighting scheme uses a heuristic and does not rely on anti-monotonicity. There are some similarities with our approach, but this method assumes that higher weights are more significant. This is usually the case where the weight indicates the count of an edge occurance, but not necessarily when the weight is an arbitrary continuous value. Our approach is more general, as it can make use of multi-dimensional numeric attributes and pruning is based on the concept of outlierness rather than low weights.

[40] propose a method to integrate constraints into frequent subgraph mining. They use anti-monotone constraints to prune the search space and monotone constraints to speed up the evaluation of further constraints. One of the proposed constraints is average weight, which is used to prune vertices and edges with outlier values from the input database as a pre-processing step.

Constraints on weighted graphs are considered within a general framework in [12]. The authors note that while weight-based constraints are not guaranteed to be anti-monotone, in practice there is frequently a correlation between graph structure and weights. Attribute-based constraints 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.

2.3 Numeric Outlier Detection

One of the most popular definitions of an outlier is "an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data." [3] Common approaches to outlier detection include statistical, cluster-based, distance-based or density-based methods [3, 6, 9]. Statistical thresholding or distance-based approaches detect outliers with respect to the global data distribution. Where data is comprised of multiple clusters of non-uniform density, it is more meaningful to detect local outliers [5]. An unsupervised local outlier approach is appropriate when analysing graph attributes, as each attribute may follow a different distribution and we do not know these distributions in advance [9].

The most established algorithm for unsupervised outlier detection in data of varying density is Local Outlier Factors (LOF) [5]. LOF computes the outlierness of each point with

respect to its local neighbourhood rather than the global data. A number of variants of LOF have been proposed. LOCI [35] defines separate neighbourhoods for counting and sampling, though LOF has been shown to outperform LOCI in terms of accuracy and runtime [18]. COF [37] uses a density measure based on "chaining distance" rather than Euclidean distance. INFLO [21] addresses the case where dense and sparse neighbourhoods are close to each other, by taking "reverse nearest neighbours" into account. The reverse nearest neighbours of a point p are the points which contain p as one of their k-nearest neighbours.

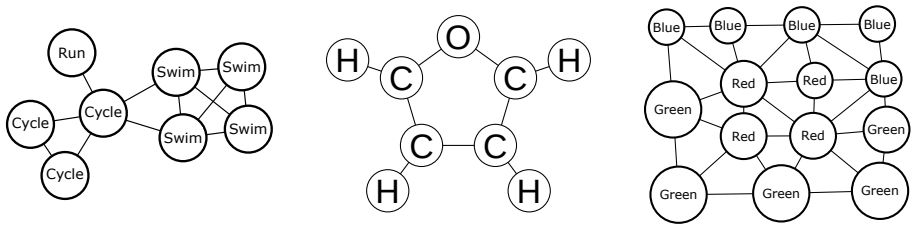
[36] presents a unified theoretical framework in which to compare density-based outlier detection methods. By generalising the notion of locality, LOF is compared to a number of specialised domain-specific algorithms. LOF is shown to be a good general method which can be applied to many different types of data; in most cases, specialised approaches are unjustified.

One weakness of LOF is high computational complexity for large datasets. The complexity of the nearest neighbour search in Naïve LOF is $O(n^2)$, but this can be reduced to $O(n \log n)$ using indexing [5]. If we are only interested in the top-n outliers, efficiency can be improved by avoiding the LOF calculation for objects deep within a cluster [20].

A second weakness is a drop in performance on very high dimensional datasets. [44] is a detailed survey on recent work on outlier detection in high-dimensional space. The "curse of dimensionality" is broken down into concrete phenomena: the distance concentration effect (as dimensionality increases, the relative contrast between near and far neighbours tends to degrade); the presence of irrelevant attributes (subspace approaches tackle this directly; other approaches do so indirectly or not at all); and efficiency issues. One interesting result is that it is not the concentration effect *per se* which impedes outlier detection in high dimensions: the presence of irrelevant attributes makes the concentration effect a problem.

Projected Index Nearest Neighbours (PINN) [39] solves the problem of efficiently discovering the k-Nearest Neighbours (k-NN) of each point in high-dimensional space. As data dimensionality grows, L_p-norm distance measures lose their discriminative ability, so indexing is no longer effective. PINN solves this by indexing in a lower-dimensional projected space, based on the observation that real-world datasets have high representational dimensionality but low intrinsic dimensionality [38]. The Random Projection used by PINN [2] is based on the Johnson-Lindenstrauss Lemma, which proves that a reduced dimensionality of $k = 2 \cdot \frac{\log n}{e^2}$ will preserve the pairwise Euclidean distances between data points within an error bound of $1 + e$ with high probability. A candidate k-NN set is determined in the projected space, then used as an index into the original space to calculate the LOFs. The PINN paper proves that LOF scores are preserved within the known error bounds of the Random Projection.

PINN does not explicitly solve the problem of irrelevant attributes. Irrelevant attributes can be explicitly handled by subspace feature selection [44]. Subspace Outlier Detection (SOD) [25] is a subspace LOF method which combines the tasks of finding relevant subspaces and detecting outliers, but the problem of indexing in high-dimensional space is not addressed. DB-CSC [15] combines graph clustering with subspace feature selection on the numeric attributes of the graphs, but experimental datasets only have synthetic data up to 20 dimensions, so it is not clear if this approach scales to higher dimensions. Subspace feature selection in high-dimensional data remains an active area for research.



(a) Social graph (b) Molecular structure (c) Scene Analysis

Fig. 1: Examples of labelled graphs

3 Graph Attributes and Structure

In its most abstract form, a graph G consists of a set of vertices V and a set of edges $E \subseteq V \times V$. When a graph represents a real-world network or phenomenon, the vertices or edges are labelled, as in Fig. 1. In Fig. 1a, the vertices represent people, labelled with their favourite sport; edges represent friendships. Fig. 1b is a graph of a molecular structure: vertices are atoms and edges are chemical bonds. Fig. 1c is a planar graph representing a frame in a video sequence where a red object is moving across a green and blue background; vertices are superpixels in the video frame and edges represent spatial relationships between adjacent superpixels.

Previous work on attribute-based constraints has assumed independence between the structure of a graph and its attributes [12, 19], but this is not the case in most real-world graphs. Our approach makes the opposite assumption: that the attribute values on each vertex and edge are dependent on its adjacent vertices, but are conditionally independent (CI) of the rest of the graph.

The example graphs in Fig. 1 illustrate the dependence of graph structure and attributes. In social networks (Fig. 1a), there are at least three processes which lead to dependencies [14]: *homophily*, the tendency to form relationships with people similar to ourselves; *induction*, the tendency to influence others in our social group; and *confounding*, where connected individuals are exposed to the same external influences (such as an economic downturn or living in the same neighbourhood). Attributes within social groups are correlated, and tend to become more homogenous over time. The social network in Fig. 1a has a discrete label indicating the favourite sport of the actors. We could also include numeric labels to indicate measures of physical activity; physical fitness (Body Mass Index, resting heart rate); or even emotional states (a "happiness index") [14].

In molecular structures (Fig. 1b), the vertex label (atom name) is conditionally dependent on its place in the molecular structure: the degree of each vertex is dependent on the number of free electrons of each atom. The lengths of the edges (atomic bonds) are dependent on the atomic weights of the vertices.

In Fig. 1c, the colour of each superpixel 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.

Recent research on graph generating algorithms assumes that dependencies exist between graph structure and attributes [23]. It is possible to learn "affinity matrices" from

real-world graphs [22], which encode the probability of an edge between each pair of vertices based on its attribute values.

Given that dependencies exist between graph labels and structure, what effect does this have on substructure discovery? To discuss this, we introduce some notation and the concept of vertex and edge partitions. Labelled graphs can have an arbitrary number of discrete and numeric labels on their vertices and edges:

Definition 1 A labelled graph G is a tuple $\langle V, E, L, \mathscr{L}_V, \mathscr{L}_E \rangle$. 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; $\mathscr{L}_V$ and $\mathscr{L}_E$ 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:

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

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

During substructure discovery, one important method of reducing the complexity of the GI test is vertex partitioning [13, 17, 27, 42]. 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:

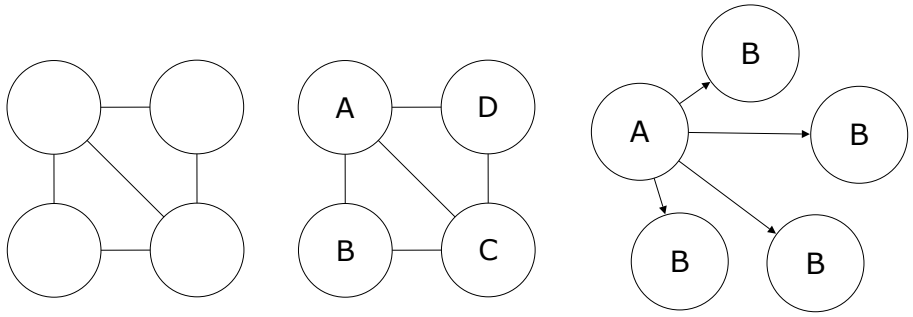
$$\forall v \in V_i \quad \forall w \in V_i \quad \forall l \in (L_V \cap L^D) : \quad \mathscr{L}_V(v, l) = \mathscr{L}_V(w, l)$$

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, $\langle v, w \rangle \Leftrightarrow \langle w, v \rangle$

There is an important relationship between the number of partitions and the complexity of GI and substructure discovery. Consider the unlabelled partial clique in Fig. 2a. This graph contains eight distinct substructures with two or more vertices; 27 instances in all. It will require 20 GI tests to determine which instances are isomorphic. The graph in Fig. 2b has the same structure, but each vertex is uniquely labelled. There are 27 distinct substructures with a single instance of each: each substructure has a distinct vertex and edge partition set.



(a) Unlabelled graph (b) Labelled graph (c) Possible substructure expansions

Fig. 2: Relationship between vertex partitions and number of substructures

As substructures with different partition sets cannot be isomorphic, we know that there are 27 distinct substructures without needing to do any GI tests.

A more subtle effect of vertex labelling is its effect on the number of possible expansions of partially-overlapping substructures. Sometimes multiple instances of the same subgraph overlap within one graph. Consider the example in Fig. 2c, which has only one instance of vertex A, but contains four instances of the substructure defined by $E = \{\langle A,B\rangle\}$. As the size of the candidate substructures is expanded, the number of instances can go up or down: there are six instances of $\{\langle A,B\rangle, \langle A,B\rangle\}$ and four instances of $\{\langle A,B\rangle, \langle A,B\rangle, \langle A,B\rangle\}$. In other words, where instances of the same substructure are allowed to overlap, the count of instances violates the anti-monotone property. In general, overlapping instances do not contribute independently to the support of that substructure[1], but it is necessary to maintain a record of each instance for the next phase of expansion, so it has an impact on the runtime and memory requirements of the search.

In practice, most real-world graphs lie somewhere between the extremes of no labels and a unique labelling for every vertex. Fig. 3 shows the relationship between the number of vertex partitions and the complexity of substructure discovery: the number of distinct substructures rises linearly with the number of partitions (Fig. 3a), but the number of instances of each substructure falls, leading to an exponential reduction in the number of GI tests required (Fig. 3b).

In the next section, we exploit the dependencies between vertex labels to define a constraint based on numeric attribute values. For frequency-based search algorithms, the property of anti-monotonicity is preserved.

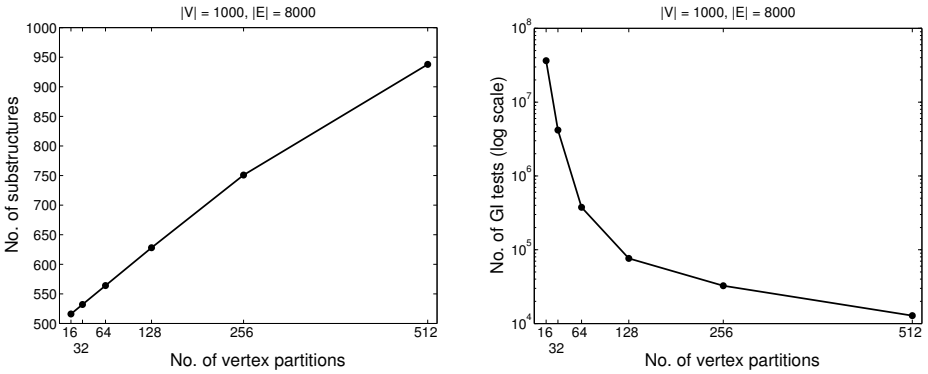
4 Frequent Substructure Discovery with Numeric Attribute Constraints

Formally, we define a constraint as follows:

Definition 5 A constraint c is a Boolean predicate which any subgraph $G_0 \subseteq G$ must fulfil. c is said to be anti-monotone if it satisfies the downward closure property:

$$\forall G_1 \subset G_0 : c(G_0) \implies c(G_1)$$

[1] Sometimes overlapping substructures can contribute independently: [34] discusses alternative strategies for handling overlapping substructures in Subdue.



(a) Unique substructures (b) Graph Isomorphism tests

Fig. 3: Effect of no. of vertex partitions on complexity of substructure discovery[2]

An example of an anti-monotone constraint is minimum support: the support (frequency count) of a graph g cannot exceed the support of any of its subgraphs g'.

The anti-monotone condition holds for frequency-based approaches such as gSpan, but not for the compression-based approach employed by Subdue. Subdue evaluates each discovered substructure by its compression ratio, given as:

$$\frac{DL(G)}{DL(S)+DL(G|S)}$$

where $DL(G)$ is the description length of the input graph, $DL(S)$ is the description length of the substructure and $DL(G|S)$ is the description length of G after compression with S. As the compression ratio is a function of both the size and frequency of the substructure, it is possible for a graph g to have a higher compression ratio than its subgraphs g'.

We have argued that the most descriptive substructures are 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:

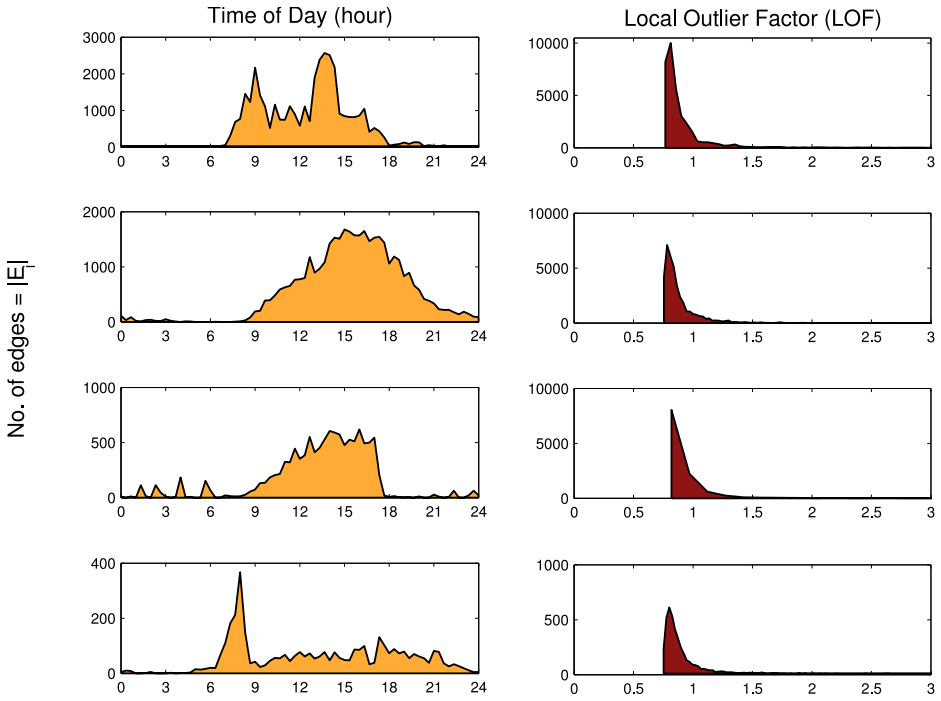
Definition 6 We define a numeric outlier function $\mathscr{O}$ on a dataset D as:

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

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 $\mathscr{O}$ depend on the specific choice of outlier detection function. $\mathscr{O}$ must be applicable to any general data set: we cannot assume prior knowledge of how numeric attribute values are distributed. Each attribute may be generated by a separate process, more likely multiple processes. We also expect that different generating processes act on different parts of the graph structure.

[2] Fig. 3 shows the results for R-MAT random graphs, with $0, 1, \ldots, 9$ binary labels, *i.e.* 1–512 vertex partitions. The label values were assigned independently from a uniform distribution. Our experiments on 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.



(a) Empirical data (b) LOFs

Fig. 4: Distribution of data and Local Outlier Factors (LOFs)

Fig. 4 shows an example of one numeric attribute from the Access Control dataset of Sect. 5.1. The edges of the graph have a Time of Day attribute, which records the time that an individual moved between a pair of door sensors (vertices). The processes which generate Time of Day are dependent on the edge partition; Fig. 4a shows the distribution of Time of Day for four sample partitions. For each partition, the mixture of processes is different: in a lab, the people who work there tend to come in in the morning and around lunchtime; others may visit the lab for short periods; security staff show up periodically for short intervals in the middle of the night. The patterns of behaviours in a laboratory are very different from those in a lift. Our method requires an unsupervised outlier detection function which can perform reliably over a wide range of data distributions and mixtures.

To calculate outliers over a range of unknown data distributions, we use LOF. The LOF score of a sample p is a measure of its outlierness with respect to its local neighbourhood:

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

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 $\mathrm{LOF}(d) \lesssim 1$. Outliers have LOF values several times larger. Thus LOF satisfies the property given in Def. 6: $\mathrm{LOF}(d) \approxeq 1$ for normal values of d and $\mathrm{LOF}(d) \gg 1$ for anomalous values.

The distributions of LOF scores for the sample edge partitions are shown in Fig. 4b. Although the data distributions are very different, the distributions of LOF scores are very similar, with normal values ≤ 1 and anomalous values stretching out in a long tail to the right. LOF is therefore well-suited to unsupervised learning, as we can reliably detect outliers without making any assumptions about the underlying distribution of the data or the sizes and densities of clusters.

Definition 7 For each vertex $v \in V_i$, we define d_v as a multi-dimensional feature vector across all the numeric attributes of v: $d_v = \mathscr{L}_V(v, L^N)$. The outlier factor $\mathscr{O}(d_v)$ is calculated relative to the dataset defined by its vertex partition:

$$\mathscr{O} : D_i \to \mathbb{R} \qquad \forall d_v \in D_i : v \in V_i$$

Outlier factors for edges are defined analagously, relative to their edge partition.

Definition 8 A vertex $v \in V_i$ is normal if $\mathscr{O}(\mathscr{L}_V(v, L^N)) \leq q_0$, anomalous otherwise. An edge $e \in E_i$ is normal if $\mathscr{O}(\mathscr{L}_E(e, L^N)) \leq q_0$, anomalous otherwise.

To use $\mathscr{O}$ as a constraint on substructure discovery, structural elements are classified as normal or anomalous by Def. 8. Anomalous vertices and edges are pruned from the graph during pre-processing, before generating all frequent 1- and 2-vertex subgraphs. After the pruning step, substructure discovery proceeds as normal. (Details are in the Appendix.)

As we only consider elements with "normal" numeric values to be part of normative substructures, this constraint dramatically reduces the number of GI tests required, without significantly affecting which substructures are discovered. We validate this experimentally in Sect. 6.

4.1 High-dimensional Numeric Attributes

LOF works well for datasets with low to moderate dimensionality. At higher dimensions, we face additional challenges to handling complexity. The most expensive part of LOF is discovering the k-NN of each point. At low dimensionality, the complexity of the k-NN search is $\mathrm{O}(mn \log n)$ (for n data points with dimensionality m, using Indexed LOF). As data dimensionality grows and L_p-norm distance measures lose their discriminative ability, indexing is no longer effective. The alternative—a sequential search for each point's neighbourhood—has complexity $\mathrm{O}(n^2 m)$. We address this problem by using PINN, where the candidate k-NN set is determined in the projected space, with complexity $\mathrm{O}(nm \log n)$.

Using RP + PINN + LOF makes LOF tractable for large, high-dimensional data. We evaluate this scheme for graphs with high-dimensional numeric attributes in Sect. 6.3.

5 Datasets

There are applications of frequent substructure mining in almost every domain: the graphs in Fig 1 are from social network analysis, predictive toxicology and video scene analysis. We evaluate our method on datasets from two application areas:

Physical Building Security. Many secure buildings—such as airports, hospitals and power stations—are equipped with Access Control Systems (ACS) based on door sensors and electronic locks. Authorised users gain access by presenting credentials, typically an ID card. The system authenticates users and records all movements in a database. Frequent substructure mining can be used to identify "normal" and "suspicious" movement patterns. We apply our frequent substructure mining approach to the ACS transaction database of a large University campus (Sect. 5.1).

Digital Forensics. During the 2011 News International phone hacking scandal, police in Scotland Yard obtained evidence for their arrests from a database of 300 million e-mails[3]. Mining a "who communicates with whom" e-mail network can focus attention on the most important or most unusual communications. Ten years previously, e-mail evidence played an important role in the indictments following the Enron scandal. One of the problems facing investigators was the deliberate deletion of e-mails [30]. Inference based on frequent patterns can be used to identify anomalous "holes" in the network where e-mails may have been deleted. We apply our graph mining approach to the Enron e-mail database organised as a social network (Sect. 5.2) and as a bipartite network of senders and messages (Sect. 5.3).

5.1 Access Control System

The graph in Fig. 5a is from the building Access Control System (ACS) for a university campus. 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 mining the typical paths through the network, we organised the data as a graph transaction database, where each subgraph represents the movement of an individual within a 12-hour time period (Fig. 5b). If a user fails to swipe a sensor (*e.g.* someone holds open a door for them), there will be a missing vertex in the graph. We compensated for missing vertices by including forward edges, turning each chain graph into a directed clique (Fig. 5c). Mining cliques is substantially more complex than mining chain graphs.

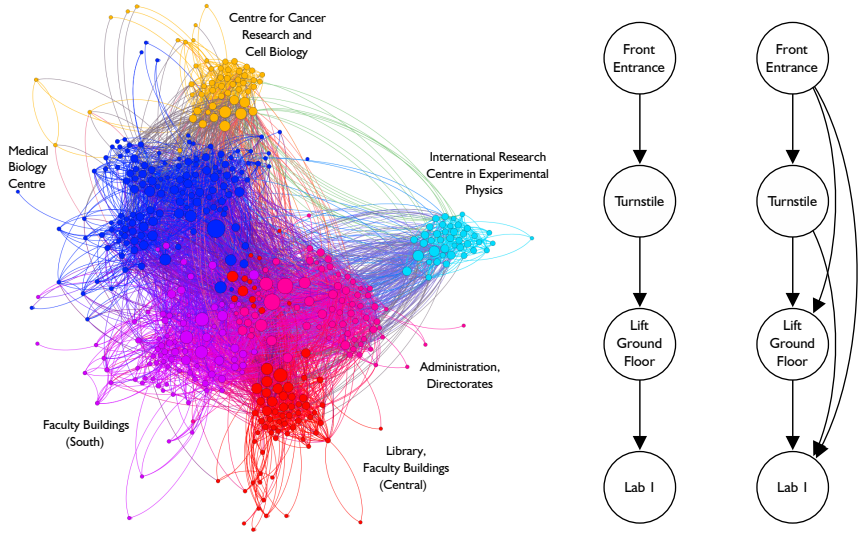
The characteristics of the ACS graph transaction database are given in Table 1. We labelled the graph edges with two numeric attributes: *Absolute Time*, the time of day (seconds since midnight) when the user presented their ID card to the door sensor at the end of the path segment; and *Elapsed Time*, the difference in seconds between the absolute time at the current sensor and the absolute time at the previous sensor. This dataset is used to evaluate LOF + gSpan (Sect. 6.1).

5.2 Enron Social Graph

During the investigation following the collapse of Enron [30], the US Federal Energy Regulatory Commission ordered that the e-mails of over 150 senior employees be made public. See [24] for an overview of the Enron e-mail corpus. For our experiments, we used all e-mail messages where the sender and at least one of the recipients was one of the 159 identifiable individuals in the dataset[4].

[3] `http://www.bbc.co.uk/news/uk-15679784`

[4] Graphs are created from the August 21, 2009 version of the Enron corpus, `https://www.cs.cmu.edu/~enron/`. Identification of individuals and job roles are from "Ex-employee Status Report", `http://`



(a) Single large graph. (Colour version available online.)

(b) Chain graph

(c) Clique

Fig. 5: Graph of Access Control Transactions on a University campus

Graph type	Graph Transaction Database	
No. of graph transactions	70,595	
Edge type	Directed, with self-cycles	
	Vertices	Edges
	Discrete:	Numeric:
Labels	Door Sensor ID	Absolute Time, Elapsed Time
Total number of elements	554,661	5,952,974
Maximum per transaction	340	57,630
Average per transaction	7.8	84.3
Number of partitions	468	339

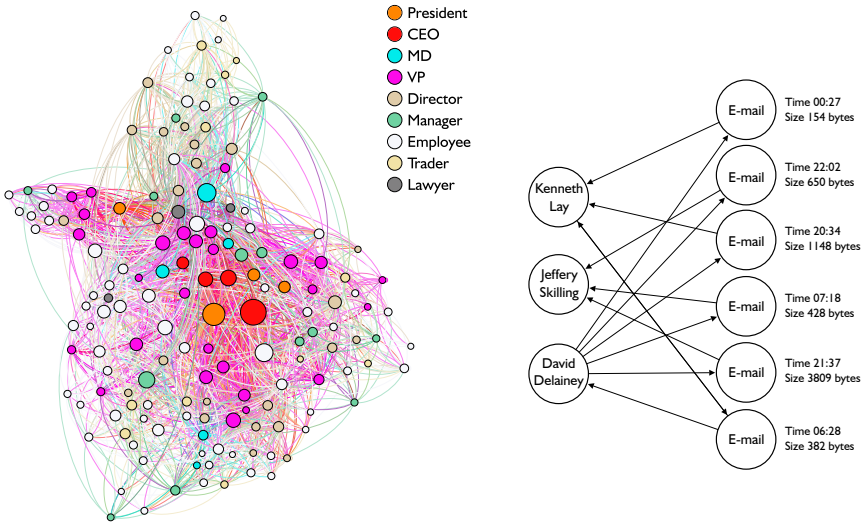
Table 1: Characteristics of the Access Control System Graph Database (Fig. 5)

Our first Enron graph is a social graph showing who commnicated with whom (Fig. 6a). Vertices are people; weighted edges represent the volume of communication. The detailed characteristics of this graph are given in Table 2. As this is a single large graph, we use it to evaluate LOF + Subdue (Sect. 6.2).

5.3 Enron Bipartite Graph

Our third graph is a more detailed view of the Enron data, including a vertex for each message as well as for senders and recipients. This is more interesting for data mining. We labelled each message vertex with two numeric attributes: the *Size* of the message in bytes and the *Time* it was sent (seconds since midnight). Fig. 6b shows an excerpt of a few mes-

`www.isi.edu/~adibi/Enron/Enron.htm`. This spreadsheet contains 161 names, but two of them appear to be duplicates.



(a) Social graph of who communicates with whom. (Colour version available online.)

(b) Fragment of bipartite graph of actors and messages

Fig. 6: Graph representations of Enron e-mail corpus

Graph type	Single large graph; weighted egonet	
Structure	Unipartite	
Edge type	Directed, with self-cycles	
	Vertices	Edges
Labels	Discrete: Employee type	Numeric: Weight
Number of elements	159	2767
Number of partitions	9	58

Table 2: Characteristics of the Enron Social Graph (Fig. 6a)

sages and Table 3 gives the characteristics of the graph. This graph is used to evaluate LOF + Subdue (Sect. 6.2).

5.4 High-dimensional Numeric Attributes

Common high-dimensional data objects include text or images. To evaluate the performance of our approach on graphs with very high-dimensional numeric attributes, we used the bipartite graph structure from Sect. 5.3, but replaced the low-dimensional numeric attributes of Table 3 with a high-dimensional numeric representation of the message. Each message vertex was labelled with a “bag of words”, a feature vector representing the occurances in the message of the most common 1,000 words in the corpus. The experiments were conducted using RP + PINN + LOF + Subdue, and were repeated for longer feature vectors with up to 4,000 dimensions. The results are in Sect. 6.3.

Graph type		Single large graph	
Structure		Bipartite	
Edge type		Directed	
	Vertices (Actors)	Vertices (Messages)	Edges
Labels	Discrete: Name	Numeric: Size, Time Sent (Bag of Words)	Discrete: FROM/ TO/CC/BCC
Number of elements	159	31,396	96,286
Number of partitions	159	1	636

Table 3: Characteristics of the Enron Bipartite Graph (Fig. 6b)

6 Experiments and Results

Our experiments evaluate the performance and accuracy of our method using two well-established graph mining approaches. For graph transaction databases, we evaluate against gSpan. For single graphs, we evaluate against Subdue. For each dataset, we measure the coverage of the result set compared to the unconstrained approach; and the the computation time and memory usage for substructure discovery.

Implementation notes. Our implementation of Subdue was written in C++, using the graph isomorphism test from the Boost Graph Library (BGL)[5]. We used the gSpan implementation from the gboost toolkit[6] as it can handle graphs with directed edges and self-cycles. We implementated LOF in C++, and validated against the "official" implementation in the ELKI toolkit[7]. The RP + PINN + LOF implementation was from ELKI. Graphs were stored in GEXF format[8], which allows an arbitrary number of discrete and numeric labels to be attached to vertices and edges. Our experiments were conducted on an Intel Xeon 2.67 GHz CPU with 100 Gb of main memory, running 64-bit Debian GNU/Linux 6.0 ("Squeeze").

6.1 Graph Transaction Database

Fig. 7 shows the runtime and memory performance of substructure discovery for the ACS graph transaction database with the unconstrained search and constrained search, calculating LOFs on absolute time, elapsed time and on a two-dimensional feature vector of absolute + elapsed time.

With unconstrained search, the lowest possible minimum support threshold was 1.3%; below this, memory was exhausted and the search became intractable. Using constrained search, it was possible to lower the minimum support to 0.4% within the same time and memory limits.

At minimum support 1.3%, constrained search with one numeric attribute ran ≈ 100 times faster than unconstrained search. Using two numeric attributes ran 183 times faster.

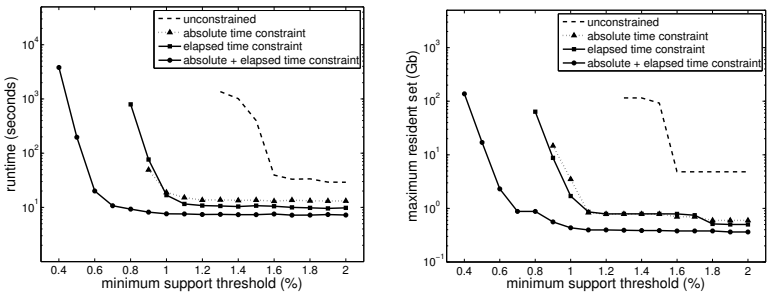
Fig. 8a shows the number of patterns discovered using unconstrained search compared with the constrained searches. The maximum number of patterns that could be discovered

[5] `http://www.boost.org/doc/libs/1_53_0/libs/graph/doc/isomorphism.html`

[6] `http://www.nowozin.net/sebastian/gboost/`

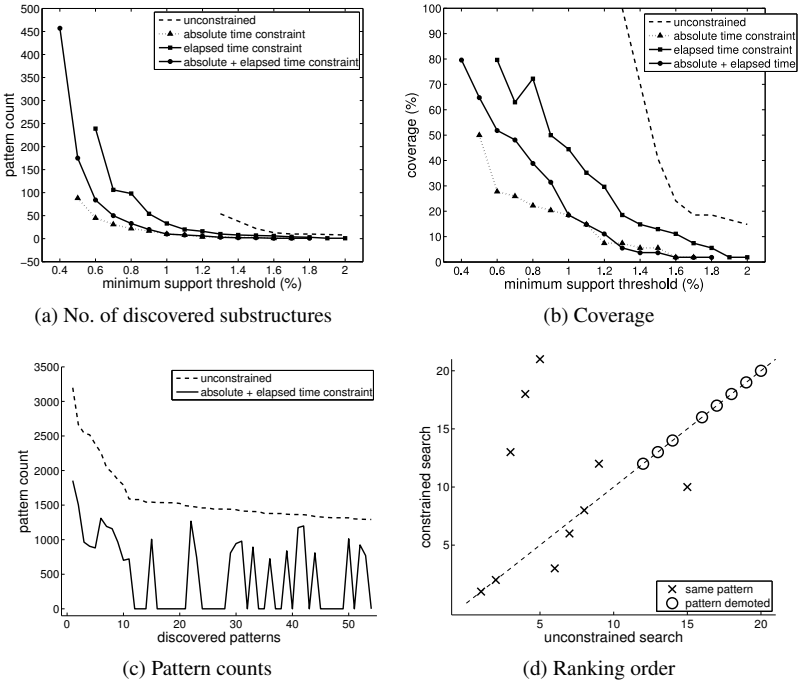
[7] Environment for Developing KDD Applications Supported by Index Structures [1], `http://elki.dbs.ifi.lmu.de/`

[8] Graph Exchange XML Format, `http://gexf.net/format/`



(a) Computation time (b) Memory requirements

Fig. 7: Performance of constrained and unconstrained substructure discovery in the Access Control System graph database



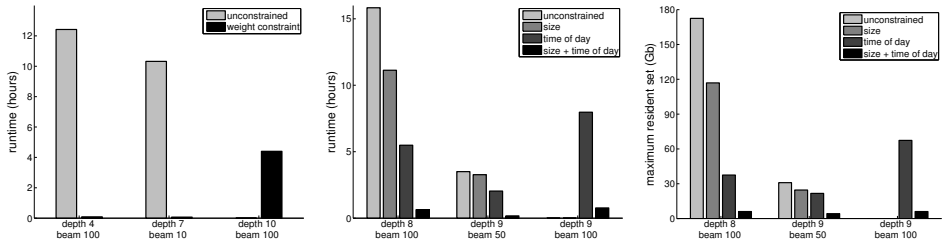
(a) No. of discovered substructures (b) Coverage

(c) Pattern counts (d) Ranking order

Fig. 8: Analysis of subgraphs returned by constrained and unconstrained substructure discovery in the Access Control System graph database

using the unconstrained search was 54, at 1.3% minimum support. With the constrained search, it is possible to find more patterns by lowering the minimum support threshold.

As with all approximation approaches, our method does not return a complete set of patterns. Fig. 8b shows the coverage of patterns discovered by the 1.3% baseline minimum support for unconstrained search. At 1.3% minimum support, the elapsed time constraint returns 20% of the patterns. More patterns can be recovered by reducing the threshold;



(a) Computation time (Enron Social graph)

(b) Computation time (Enron Bipartite graph)

(c) Memory requirements (Enron Bipartite graph)

Fig. 9: Performance of substructure discovery in the Enron graphs

at minimum support 1.0% we recover around 45% of the patterns with a small additional computation and memory cost (around 10%).

Figs. 8c and 8d give some insights into which patterns are retained at minimum support 1.3%. The unconstrained search with minimum support 1.3% (54 patterns; 1,363 secs) is compared to the constrained search with minimum support 0.7% (50 patterns; 10 secs); we chose the threshold for constrained search which returns approximately the same number of patterns as the maximum number for unconstrained search. The top 10 patterns (those with the highest support) are all retained. The support for each substructure is reduced in different proportions, depending on the how normal or anomalous the numeric attributes of each instance are.

These results show a beneficial trade-off between improved performance and reduced coverage. We have argued that the patterns which are retained are the most descriptive ones, as they satisfy both minimum support and the normal numeric attribute constraint. This conclusion could be strengthened by applying our approach to a supervised learning task on a dataset with ground truth.

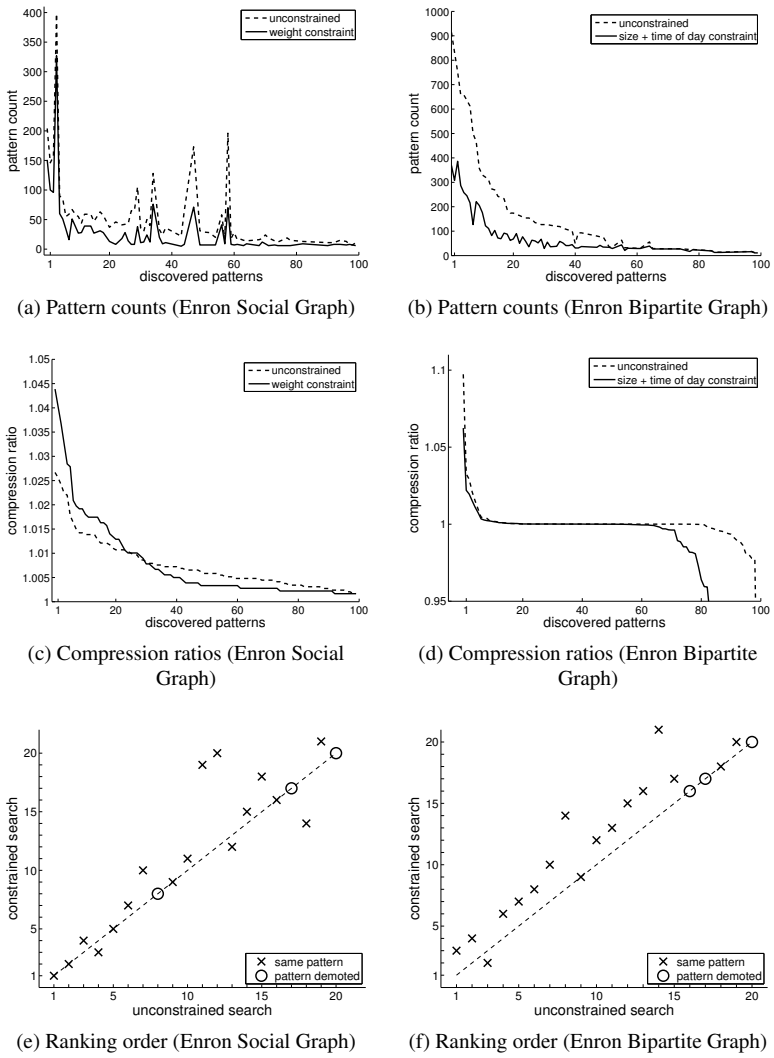
6.2 Single Large Graphs

The results for single large graphs (LOF + Subdue) are similar to those for graph transaction databases. Fig. 9 shows the results on the Enron graphs for a range of Subdue parameter settings: *depth* controls how large discovered substructures are allowed to grow before the search terminates; *beam width* controls the "greediness" of the search.

The Enron Social graph (Fig. 9a) has a single numeric attribute (edge weight). At depth 4, the constrained search was 142 times faster. Increasing to depth 7, the unconstrained search was intractable; narrowing the beam width to 10, the constrained search was 155 times faster. At depth 10, the unconstrained search was intractable even at beam 10, but the constrained search could still complete in reasonable time, even at beam 100.

The Enron Bipartite graph (Fig. 9b) has two numeric attributes: as in Sect. 6.1, calculating LOFs across the two-dimensional feature vector gives better performance than using either attribute in isolation. Memory requirements are similarly reduced (Fig. 9c); using more numeric attributes makes the search more tractable. These results demonstrate that using our approach, we can relax the greediness of the Subdue search while keeping processing time tractable.

As with the transaction graphs, the constrained search returns a very similar set of patterns to the unconstrained search (Fig. 10). As Subdue uses a greedy search heuristic, it



(a) Pattern counts (Enron Social Graph)

(b) Pattern counts (Enron Bipartite Graph)

(c) Compression ratios (Enron Social Graph)

(d) Compression ratios (Enron Bipartite Graph)

(e) Ranking order (Enron Social Graph)

(f) Ranking order (Enron Bipartite Graph)

Fig. 10: Analysis of subgraphs returned by substructure discovery in the Enron graphs

is not meaningful to compare coverage. The relative counts of the top patterns are shown in Figs. 10a–10b. As Subdue ranks patterns by their compression ratio, the most frequent patterns are not necessarily ranked highest. While the total number of instances of each pattern have been reduced by the constraint, the relative count of each pattern is approximately preserved.

The compression ratios of the top patterns are compared in Figs. 10c–10d. The shape of the curve is approximately preserved for the constrained and unconstrained search. In Fig. 10c, the gradient of compression ratios is steeper using the constraint. By removing anomalous edges, we have increased the discrimination between "good" patterns and less good patterns. In the Bipartite graph (Fig. 10d), there are a smaller number of interesting

patterns to begin with (patterns with a compression ratio of 1.0 or less are not interesting). The compression ratios of the interesting patterns are preserved very well.

The ranking of the top 20 substructures discovered by the constrained and unconstrained searches is shown in Figs. 10e–10f. For both graphs, 17 of the top 20 results are the same, and the ordering is very similar. Thus our constraint-based approach provides order-of-magnitude improvements in discovery time with little impact on which substructures are discovered.

6.3 High-dimensional Numeric Attributes

Our experiments using RP + PINN + LOF to calculate outliers on the Enron bipartite graph (Fig. 11) are preliminary, but suggest some interesting directions for future research.

Fig. 11a shows the time to compute outliers on the full Enron bipartite graph of 300,000 vertices, each with numeric attributes of 1,000–4,000 dimensions[9]. This verifies that the PINN approach is tractable for processing the high-dimensional attributes of large graphs.

Fig. 11b shows the distribution of LOF scores at the lower end of the scale (values above 6 in the long tail are not shown). Fig.s 11c–11d show the full distribution of LOF scores for 1,000 and 4,000 dimensions. The shape of the curve in Fig. 11b is somewhat different to the characteristic shape of LOF for low-dimensional attributes (Fig. 4). At first glance, the threshold at which a point is an outlier seems to be around ≈ 2.5 rather than 1. As dimensionality increases, the distribution of normal values changes only slightly (Fig. 11b), but the distribution of outliers becomes spread out over a wider range (Fig.s 11c–11d).

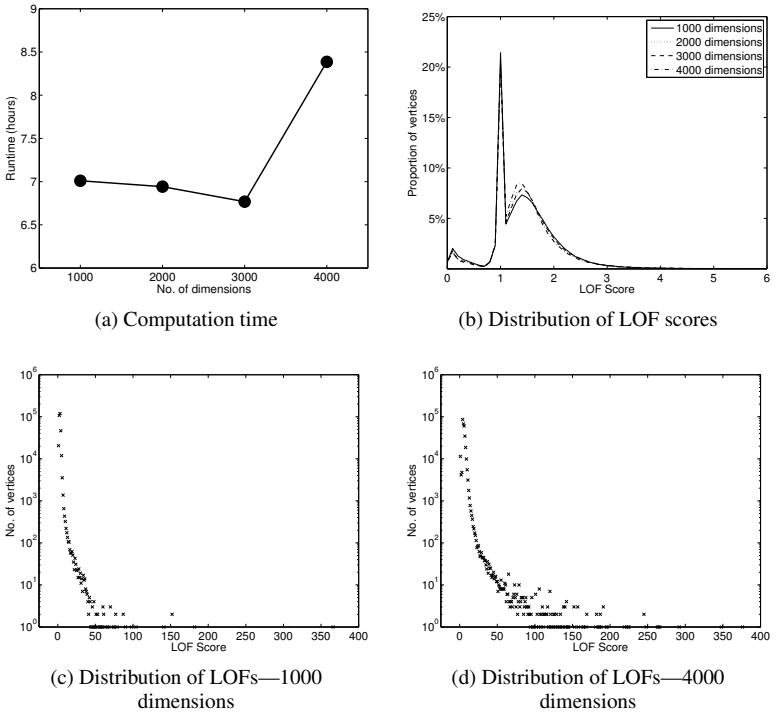
These initial results are encouraging, but further investigation is required. The L_p-norm distance measures used by LOF are valid on dense feature vectors (as long as each dimension adds information [44]) but we need to establish that they remain meaningful on sparse feature vectors such as our bag of words. If the distances are valid, it may be necessary to rescale the LOF scores, for example using one of the methods presented in [26].

6.4 Comparison with Random Removal

To verify that performance improvements are not simply because the graph is smaller, we compared our constraint-based approach to random removal of graph edges. For this experiment, we randomly deleted 10–90% of the edges in the ACS graph database before searching for frequent substructures (Fig. 12). It is necessary to remove almost 50% of the graph in order to achieve similar performance improvements to our constraint-based approach.

Fig. 12b shows how random removal affects the frequency of the top 10 substructures, compared to the constraint-based approach. Randomly removing edges increases the entropy of the graph: the shape of the curve becomes flatter as more of the graph is removed. After 20% edge removal, discrimination between substructures 4–10 has degraded and by 50% edge removal, the order is random. Randomly deleting graph edges does not preserve information about the "goodness" of discovered substructures.

[9] We were unable to calculate RP + PINN + LOF for larger datasets due to memory constraints. The random projection [2] used by PINN is designed to be "database-friendly"; by changing the indexing method it would be possible to create a PINN implementation which creates its index on disk in order to process larger or higher-dimensional datasets.



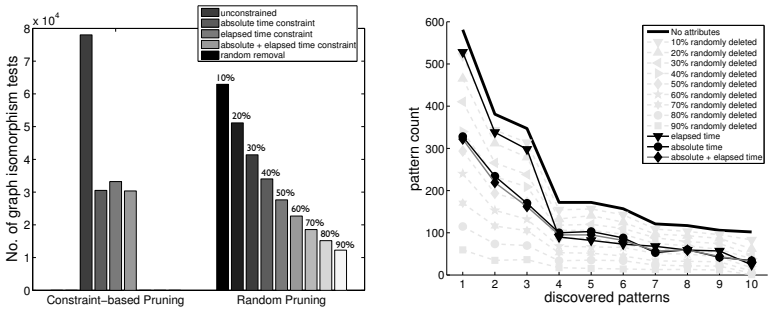
(a) Computation time

(b) Distribution of LOF scores

(c) Distribution of LOFs—1000 dimensions

(d) Distribution of LOFs—4000 dimensions

Fig. 11: Random Projection (RP) + PINN + LOF on "bag of words" feature vectors on Enron bipartite graph



(a) Number of GI tests required

(b) Frequency of top 10 substructures

Fig. 12: Effect of randomly removing graph edges on substructure discovery

Our results show that our constraint-based approach can find frequent subgraphs more efficiently than an unconstrained approach, without having an adverse effect on which substructures are discovered. 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.

7 Conclusions

In this paper we have 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.

Our method reduces the complexity of substructure discovery at both the instance mining and substructure grouping steps, resulting in order of magnitude improvements in discovery time and memory requirements, without significantly altering which patterns are discovered. In our experiments, we observed performance improvements of a factor of 100–200 compared to unconstrained search. An important consequence is that substructure discovery remains tractable for larger graphs than is possible with the standard algorithms.

While constraints based on a single numeric attribute provided significant improvements, the best results were observed when using multiple attributes. We also presented an outline of how our method could be applied to graph objects with very high-dimensional features (*e.g.*, image or text attributes); this area is the subject of future research.

7.1 Future work

To further improve performance, we propose to develop a fast parallel implementation of our method.

Our experiments showed that 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. With slight modification, our method could be applied to a supervised learning task on graphs labelled with numeric attributes. By constructing a one-class classifier, we will be able to objectively measure the prediction accuracy of our method against the standard algorithms.

Our method has been successfully applied to graphs with a low or moderate number of numeric attributes. We believe that it will also work for dense high-dimensional numeric attributes. For graphs with a sparse high-dimensional representation, such as the Enron bag of words, further investigation is indicated to determine whether L_p-norm distances are still discriminitive. If L_p-norm distances are not appropriate, further research is required to determine whether there exists a "database-friendly" random projection for other more suitable distance measures.

Where there is a high-dimensional representation with many irrelevant attributes, 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.

Acknowledgements We would like to thank Erich Schubert at Ludwig-Maximilians Universität München for assistance with verifying our LOF implementation and providing us with the RP + PINN + LOF implementation ahead of its official release in ELKI.

References

1. Achert E, Kriegel HP, Schubert E, Zimek A (2013) Interactive data mining with 3D-parallel-coordinate-trees. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD)

2. Achlioptas D (2003) Database-friendly random projections: Johnson-Lindenstrauss with binary coins. Journal of Computer and System Sciences 66(4):671–687, DOI 10.1016/S0022-0000(03)00025-4
3. Barnett V, Lewis T (1994) Outliers in Statistical Data, 3rd edn. Wiley Series in Probability & Statistics, Wiley
4. Borgelt C (2006) Canonical forms for frequent graph mining. In: Decker R, Lenz HJ (eds) GfKl, Springer, Studies in Classification, Data Analysis, and Knowledge Organization, pp 337–349
5. Breunig MM, Kriegel HP, Ng RT, Sander J (2000) LOF: Identifying density-based local outliers. In: Chen W, Naughton JF, Bernstein PA (eds) SIGMOD Conference, ACM, pp 93–104
6. Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: A survey. ACM Computing Surveys 41(3):15:1–15:58, DOI 10.1145/1541880.1541882
7. Cook DJ, Holder LB (1994) Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research 1(1):231–255
8. Cook DJ, Holder LB (2000) Graph-based data mining. IEEE Intelligent Systems 15:32–41
9. Davis M, Liu W, Miller P, Redpath G (2011) Detecting anomalies in graphs with numeric labels. In: Macdonald C, Ounis I, Ruthven I (eds) Proceedings of the 20th ACM Conference on Information and Knowledge Management (CIKM 2011), ACM, pp 1197–1202
10. Eberle W, Holder L (2011) Compression versus frequency for mining patterns and anomalies in graphs. In: Ninth Workshop on Mining and Learning with Graphs (MLG 2011), SIGKDD, San Diego, CA, at the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2011)
11. Eichinger F, Böhm K, Huber M (2008) Mining edge-weighted call graphs to localise software bugs. In: Daelemans W, Goethals B, Morik K (eds) ECML/PKDD (1), Springer, Lecture Notes in Computer Science, vol 5211, pp 333–348
12. Eichinger F, Huber M, Böhm K (2010) On the usefulness of weight-based constraints in frequent subgraph mining. In: ICAI 2010, BCS SGAI, pp 65–78, DOI 10.1007/978-0-85729-130-1_5
13. Fortin S (1996) The graph isomorphism problem. Tech. rep., Univ. of Alberta
14. Fowler JH, Christakis NA (2008) Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the Framingham Heart Study. BMJ 337, DOI 10.1136/bmj.a2338
15. Günnemann S, Boden B, Seidl T (2011) DB-CSC: a density-based approach for subspace clustering in graphs with feature vectors. In: Proceedings of the 2011 European conference on Machine Learning and Knowledge Discovery in Databases, Springer, ECML PKDD'11, vol Part I, pp 565–580
16. Huan J, Wang W, Prins J, Yang J (2004) SPIN: Mining maximal frequent subgraphs from graph databases. In: KDD 2004, ACM, pp 581–586
17. Inokuchi A, Washio T, Motoda H (2000) An Apriori-based algorithm for mining frequent substructures from graph data. In: PKDD 2000, Springer, pp 13–23
18. Janssens J, Flesch I, Postma E (2009) Outlier detection with one-class classifiers from ML and KDD. In: International Conference on Machine Learning and Applications (ICMLA '09), pp 147–153, DOI 10.1109/ICMLA.2009.16
19. Jiang C, Coenen F, Zito M (2010) Frequent sub-graph mining on edge weighted graphs. In: Proceedings of the 12th international conference on Data Warehousing and Knowledge Discovery, Springer-Verlag, Berlin, Heidelberg, DaWaK'10, pp 77–88

20. Jin W, Tung AKH, Han J (2001) Mining top-n local outliers in large databases. In: Lee D, Schkolnick M, Provost FJ, Srikant R (eds) KDD, ACM, pp 293–298
21. Jin W, Tung AKH, Han J, Wang W (2006) Ranking outliers using symmetric neighborhood relationship. In: Ng WK, Kitsuregawa M, Li J, Chang K (eds) PAKDD, Springer, Lecture Notes in Computer Science, vol 3918, pp 577–593
22. Kim M, Leskovec J (2011) Modeling social networks with node attributes using the Multiplicative Attribute Graph model. In: Cozman FG, Pfeffer A (eds) UAI, AUAI Press, pp 400–409
23. Kim M, Leskovec J (2012) Multiplicative Attribute Graph model of real-world networks. Internet Mathematics 8(1–2):113–160
24. Klimt B, Yang Y (2004) The Enron corpus: A new dataset for email classification research. In: Boulicaut JF, Esposito F, Giannotti F, Pedreschi D (eds) ECML, Springer, Lecture Notes in Computer Science, vol 3201, pp 217–226
25. Kriegel HP, Kröger P, Schubert E, Zimek A (2009) Outlier detection in axis-parallel subspaces of high dimensional data. In: Theeramunkong T, Kijsirikul B, Cercone N, Ho TB (eds) PAKDD, Springer, Lecture Notes in Computer Science, vol 5476, pp 831–838
26. Kriegel HP, Kröger P, Schubert E, Zimek A (2011) Interpreting and unifying outlier scores. In: SDM, SIAM/Omnipress, pp 13–24
27. Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: ICDM 2001, IEEE, pp 313–320
28. Lewis DD (1998) Naïve (Bayes) at forty: The independence assumption in information retrieval. In: Nedellec C, Rouveirol C (eds) ECML, Springer, Lecture Notes in Computer Science, vol 1398, pp 4–15
29. McKay BD, Piperno A (2013) Practical graph isomorphism, II. Tech. rep., URL `http://arxiv.org/abs/1301.1493v1`
30. McLean B, Elkind P (2003) The Smartest Guys in the Room: The Amazing Rise and Scandalous Fall of Enron. Penguin Group USA
31. Moser F, Colak R, Rafiey A, Ester M (2009) Mining cohesive patterns from graphs with feature vectors. In: SDM, SIAM, pp 593–604
32. Newman M (2010) Networks: An Introduction. Oxford University Press, Inc., New York, NY, USA
33. Nijssen S, Kok JN (2004) A quickstart in frequent structure mining can make a difference. In: Kim W, Kohavi R, Gehrke J, DuMouchel W (eds) KDD, ACM, pp 647–652
34. Palacio MAP (2005) Spatial data modeling and mining using a graph-based representation. PhD thesis, Department of Computer Systems Engineering, University of the Americas, Puebla
35. Papadimitriou S, Kitagawa H, Gibbons PB, Faloutsos C (2003) Loci: Fast outlier detection using the local correlation integral. In: Dayal U, Ramamritham K, Vijayaraman TM (eds) ICDE, IEEE Computer Society, pp 315–326
36. Schubert E, Zimek A, Kriegel HP (2012) Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection. Data Mining and Knowledge Discovery DOI 10.1007/s10618-012-0300-z
37. Tang J, Chen Z, Fu AWC, Cheung DWL (2002) Enhancing effectiveness of outlier detections for low density patterns. In: Cheng MS, Yu PS, Liu B (eds) PAKDD, Springer, Lecture Notes in Computer Science, vol 2336, pp 535–548
38. Tenenbaum JB, de Silva V, Langford JC (2000) A Global Geometric Framework for Nonlinear Dimensionality Reduction. Science 290(5500):2319–2323, DOI 10.1126/science.290.5500.2319

39. de Vries T, Chawla S, Houle M (2010) Finding local anomalies in very high dimensional space. In: ICDM 2010, IEEE, pp 128–137
40. Wang C, Zhu Y, Wu T, Wang W, Shi B (2005) Constraint-based graph mining in large database. In: Zhang Y, Tanaka K, Yu JX, Wang S, Li M (eds) APWeb, Springer, Lecture Notes in Computer Science, vol 3399, pp 133–144
41. Wörlein M, Meinl T, Fischer I, Philippsen M (2005) A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston. In: Jorge A, Torgo L, Brazdil P, Camacho R, Gama J (eds) PKDD, Springer, Lecture Notes in Computer Science, vol 3721, pp 392–403
42. Yan X, Han J (2002) gSpan: Graph-based substructure pattern mining. In: ICDM 2002, IEEE, pp 721–724
43. Yan X, Han J (2003) CloseGraph: Mining closed frequent graph patterns. In: KDD 2003, ACM, pp 286–295
44. Zimek A, Schubert E, Kriegel HP (2012) A survey on unsupervised outlier detection in high-dimensional numerical data. Statistical Analysis and Data Mining 5(5):363–387

Appendix. The substructure discovery algorithms referred to throughout the paper are included here for convenience. The original references are [42] for gSpan and [8] for Subdue.

A gSpan

To apply our method to gSpan, we calculate numeric anomalies for all vertices and edges by Def. 8 as a pre-processing step. Then amend step 2 to *Remove infrequent and anomalous vertices and edges*.

Algorithm 1 GRAPHSET_PROJECTION: search for frequent substructures

Require: Graph Transaction Database $\mathbb{D}$, *minSup*
1: Sort the labels in $\mathbb{D}$ by their frequency
2: Remove infrequent vertices and edges
3: Relabel the remaining vertices and edges
4: $\mathbb{S}^1 \leftarrow$ all frequent 1-edge graphs in $\mathbb{D}$
5: Sort $\mathbb{S}^1$ in DFS lexicographic order
6: $\mathbb{S} \leftarrow \mathbb{S}^1$
7: **for all** edge $e \in \mathbb{S}^1$ **do**
8: Initialise s with e, set $s.D$ to graphs which contain e
9: Subgraph_Mining($\mathbb{D}$, $\mathbb{S}$, s)
10: $\mathbb{D} \leftarrow \mathbb{D} - e$
11: **if** $|\mathbb{D}| <$ *minSup* **then**
12: **break**
13: **return** Discovered Subgraphs $\mathbb{S}$

B Subdue

To apply our method to Subdue, calculate numeric anomalies for all vertices and edges by Def. 8 as a pre-processing step (as above). Prune all anomalous vertices and edges from the graph before step 5.

Algorithm 2 SUBDUE: search for frequent substructures

Require: Graph, BeamWidth, MaxBest, MaxSubSize, Limit
1: **let** ParentList = {}
2: **let** ChildList = {}
3: **let** BestList = {}
4: **let** ProcessedSubs = 0
5: Create a substructure from each unique vertex label and its single-vertex instances; insert the resulting substructures in ParentList
6: **while** ProcessedSubs $\leq$ Limit and ParentList is not empty **do**
7: **while** ParentList is not empty do **do**
8: **let** Parent = RemoveHead(ParentList)
9: Extend each instance of Parent in all possible ways
10: Group the extended instances into Child substructures
11: **for all** Child **do**
12: **if** SizeOf(Child) $\leq$ MaxSubSize **then**
13: Evaluate the Child
14: Insert Child in ChildList in order by value
15: **if** Length(ChildList) $>$ BeamWidth **then**
16: Destroy the substructure at the end of ChildList
17: **let** ProcessedSubs = ProcessedSubs + 1
18: Insert Parent in BestList in order by value
19: **if** Length(BestList) $>$ MaxBest **then**
20: Destroy the substructure at the end of BestList
21: Switch ParentList and ChildList
22: **return** BestList