Infrastructure Pattern Discovery in Configuration Management Databases via Large Sparse Graph Mining

Pranay Anchuri*, Mohammed J. Zaki*, Omer Barkol†, Ruth Bergman†, Yifat Felder†, Shahar Golan† and Arik Sityon†
*Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
†HP Labs, Technion City, Haifa 32000, Israel
*{anchupa, zaki}@cs.rpi.edu
†{omer.barkol, ruth.bergman, yifat.felder, shahar.golan, arik.sityon}@hp.com

***Abstract*—A configuration management database (CMDB) can be considered to be a large graph representing the IT infrastructure entities and their inter-relationships. Mining such graphs is challenging because they are large, complex, and multi-attributed, and have many repeated labels. These characteristics pose challenges for graph mining algorithms, due to the increased cost of subgraph isomorphism (for support counting), and graph isomorphism (for eliminating duplicate patterns). The notion of pattern frequency or support is also more challenging in a single graph, since it has to be defined in terms of the number of its (potentially, exponentially many) embeddings. We present *CMDB-Miner*, a novel two-step method for mining infrastructure patterns from CMDB graphs. It first samples the set of maximal frequent patterns, and then clusters them to extract the representative infrastructure patterns. We demonstrate the effectiveness of CMDB-Miner on real-world CMDB graphs.**

***Keywords*-single graph mining; frequent subgraphs; configuration management databases**

I. Introduction

A configuration management database (CMDB) is used to manage and query the IT infrastructure of an organization. It stores information about the so-called configuration items (CIs) – servers, software, running processes, storage systems, printers, routers, etc. As such it can be considered to be a *single* large multi-attributed graph, where the nodes represent the various CIs and the edges represent the connections between the CIs (e.g., the processes on a particular server, along with starting and ending times). Fig. 1 shows a snippet from a real-world CMDB graph, displaying only type labels. A CMDB provides a wealth of information about the largely undocumented IT practices of a large organization, and thus mining the CDMB graph for frequent subgraph patterns can reveal the de facto infrastructure patterns. Once mined, these patterns can be used to either set the default IT policies, or refine them if found unsatisfactory. Thus, the discovery of infrastructure patterns is an important real-world application of subgraph mining in IT domain.

Mining a CMDB graph comes with several challenges. The CMDB graph is a massive, multi-attributed, and complex graph. There are various types and sub-types of CIs, which may be hierarchically related. CIs further have various associated attributes and metadata elements. There are a lot of repetitive labels, namely, the CIs and their various attributes. For example, there can be hundreds and thousands of running processes of the same type, running on (and thus connected to) a single server. The vast majority of frequent graph mining algorithms assume that the database consists of many different graphs, so that the support or frequency can be computed by counting how many graphs in the database *contain* a given pattern. The containment is defined in terms of subgraph isomorphism, i.e., the node mapping, also called an *embedding*, corresponding to isomorphism between a pattern and some subgraph of the database graph. As long as there exists an embedding of the pattern in a database graph, the support can be incremented by one. On the other hand, the support of a pattern in a single graph usually involves finding all possible pattern embeddings (or node/edge disjoint embeddings), which can potentially be exponential in the size and order of the graph. Furthermore, the subgraph isomorphism problem is rendered more expensive due to the repetitive CIs. Simply mining the frequent subgraph patterns from the CMDB graph is not enough to discover the infrastructure patterns. As is well known in frequent pattern mining, there can be a huge number of mined patterns, with many of them being small variations of one another, what is required is to summarize the patterns and to select only the most representative ones as the infrastructure patterns.

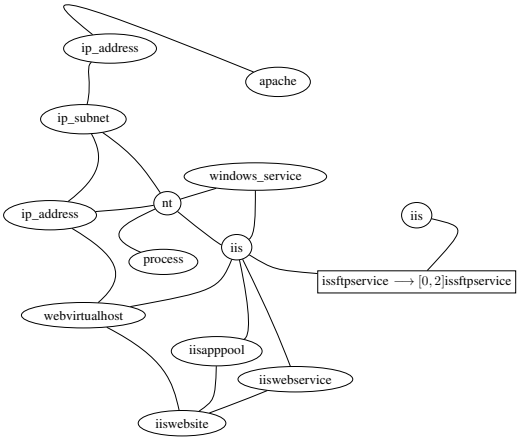


Figure 1. Snippet from a CMDB graph

This work was supported in part by an HP Labs Innovation Award and NSF Grant EMT-0829835.

In this paper, we propose an effective approach to mine representative patterns from a large multi-attributed graph, with special focus on discovering infrastructure patterns from CMDB graphs. Our approach consists of two main steps: i) Mining a sample of the maximal frequent subgraphs from a single large database graph. ii) Clustering the mined patterns via spectral graph clustering, and extracting the representative infrastructure patterns. There are several novel contributions in this paper:

- We propose a new approach for mining/sampling maximal subgraph patterns from a single database graph. In particular, we propose a new network-flow based definition of graph support which is an upper-bound on the number of edge-disjoint embeddings, and which allows us to prune patterns the moment they become infrequent. We further propose a fast filter-based approach for eliminating isomorphic (i.e., duplicate) patterns.
- We propose a new diffusion-based graph similarity method to compute the pair-wise similarities between two labeled graphs. The method takes into account both the structure and labels of the graphs. Given the pair-wise similarity matrix, we use spectral graph clustering to extract groups of related patterns. We then select the representative patterns via a coverage-based approach.

We evaluate our approach on several real-world CMDB graphs with millions of nodes and edges, and we demonstrate that our method, called *CMDB-Miner*, can find meaningful IT infrastructure patterns. Even though our focus is on CMDB graphs, our approach is generic, and can be applied in many other real-world applications with similar characteristics, namely single large graph database, multiple attributes on the nodes and many repeated labels.

II. Background

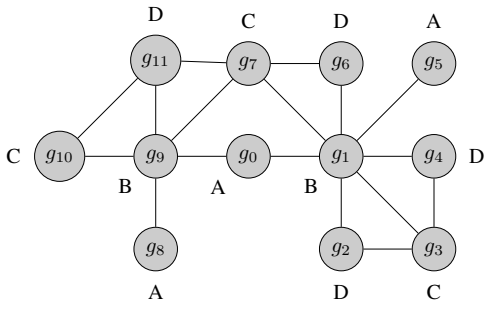
A. Preliminary Concepts

Let Σ denote a given set of labels. A *labeled graph* is a triple $G = (V, E, L)$, where V is the set of vertices or nodes, $E \subseteq V \times V$ is the set of (unordered) edges, and L is the labeling function for both nodes and edges, so that $L(v)$ is the label of a node v, and $L(e) = L(a, b)$ is the label of an edge $e = (a, b)$. The *order* of the graph is the number of nodes $|V|$, and the *size* of the graph is the number of edges $|E|$.

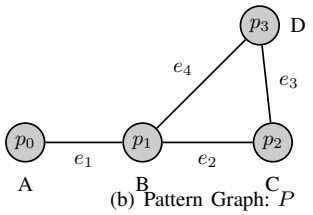
We say that $G' = (V', E', L')$ is a *subgraph* of $G = (V, E, L)$, denoted $G' \subseteq G$, if there exists a 1-1 mapping $\pi : V' \to V$, such that $(v_i, v_j) \in E'$ implies $(\pi(v_i), \pi(v_j)) \in E$. Further, π must preserve vertex and edge labels, i.e., $L'(v_i) = L(\pi(v_i))$ for all $v_i \in V'$, and $L'(v_i, v_j) = L(\pi(v_i), \pi(v_j))$ for all edges $(v_i, v_j) \in E'$. The mapping π is called a *subgraph isomorphism* from G' to G. If $G' \subseteq G$ we also say that G *contains* G'. If $G' \subseteq G$ and $G \subseteq G'$, we say that G and G' are *isomorphic*.

Let $G = (V, E, L)$ be a single large database graph, and let $P = (V', E', L')$ be a *candidate* pattern, whose support we want to compute. Let π be a subgraph isomorphism from P to G. The sequence $\pi(v_1), \pi(v_2), \ldots, \pi(v_n)$ over all $v_i \in V'$ is called an *embedding* of G' in G. For an edge $e_i = (a_i, b_i) \in E'$, define $\pi(e_i) = \pi(a_i, b_i) = (\pi(a_i), \pi(b_i)) \in E$. The sequence $\pi(E') = \pi(e_1), \pi(e_2), \ldots, \pi(e_m)$ over all edges $e_i \in E'$ is called an *edge mapping* of P in G. For example, given the database graph G in Fig. 2(a), and the candidate pattern P in Fig. 2(b), the subgraph isomorphism π_3 from P to G specified by the mapping $p_0 \to g_0$, $p_1 \to g_9$, $p_2 \to g_7$, $p_3 \to g_{11}$, corresponds to the embedding $0, 9, 7, 11$, and the edge mapping $(0, 9), (9, 7), (7, 11), (11, 9)$. Since π uniquely specifies the embedding and edge mapping, we use these terms interchangeably.

There are several ways to compute the number of occurrences, called the *support*, of P in G. The most straightforward definition is to define the support of P as the number of possible embeddings of P in G, denoted $sup_a(P)$. Figure 2(c) shows all the possible embeddings of P in G. There are ten embeddings of P in G for this example, thus $sup_a(P) = 10$. Unfortunately, there can be exponentially many embeddings of a pattern in the database graph. For example, if $G = P = K_n$, where K_n is the complete graph on n nodes, with all node and edge labels being the



(a) Database Graph: G



(b) Pattern Graph: P

π_0	$\{0, 1, 3, 2\}$
π_1	$\{0, 1, 3, 4\}$
π_2	$\{0, 1, 7, 6\}$
π_3	$\{0, 9, 7, 11\}$
π_4	$\{0, 9, 10, 11\}$

π_5	$\{5, 1, 3, 2\}$
π_6	$\{5, 1, 3, 4\}$
π_7	$\{5, 1, 7, 6\}$
π_8	$\{8, 9, 7, 11\}$
π_9	$\{8, 9, 10, 11\}$

(c) All Embeddings: Π

π_1	$\{0, 1, 3, 4\}$
π_4	$\{0, 9, 10, 11\}$
π_7	$\{5, 1, 7, 6\}$

(d) Edge-Disjoint

π_0	$\{0, 1, 3, 2\}$
π_8	$\{8, 9, 7, 11\}$

(e) Node-Disjoint

Figure 2. (a) A database graph. (b) A pattern graph. (c) All embeddings of P. (d) and (e): edge and node disjoint embeddings of P.

same, then there are $n!$ distinct embeddings of P in G. Unfortunately, due to the label multiplicities in the CMDB graphs, this is a real problem in this application. To avoid the combinatorial blowup, support can also be defined as the maximum number of node or edge disjoint embeddings of P in G, denoted $sup_n(P)$ and $sup_e(P)$, respectively. Let Π be the set of all possible embeddings of P in G. We say that two embeddings $\pi, \pi' \in \Pi$ are *node disjoint*, if $\pi(v_i) \neq \pi'(v_j)$ for all nodes $v_i, v_j \in V'$. We say that π and π' are *edge disjoint* if $(\pi(v_i), \pi(v_j)) \neq (\pi'(v_a), \pi'(v_b))$ for all edges $(v_i, v_j), (v_a, v_b) \in E'$. Figures 2(d) and 2(e) show examples of a maximum set of edge and node disjoint embeddings, respectively. These sets are not unique; for example, the embedding set $\{\pi_0, \pi_3, \pi_9\}$ is also edge-disjoint. However, the edge-disjoint support of P is $sup_e(P) = 3$, and the node disjoint support is $sup_n(P) = 2$. Finding the maximum number of edge (or node) disjoint embeddings is equivalent to finding the maximum independent set (MIS) in an *embeddings graph*, where each embedding is a node, and there exists an edge between two embeddings if they share an edge (or node). Unfortunately, the MIS problem is known to be NP-hard, and thus both the edge and node disjoint embeddings are expensive to compute. One of the novel contributions of this paper is that we approximate the edge-disjoint support via a network-flow based approach. We prefer edge-disjointness, since node-disjointness is more constrained (every node-disjoint embedding is also an edge-disjoint embedding, but not vice-versa).

B. Related Work

Many different methods have been developed for frequent subgraph mining [1]–[5]. Recently, methods that sample and summarize subgraph patterns have gained more traction [6]–[10]. However, these methods assume that the database contains many different graphs, and cannot be directly applied when the database is just a single large graph. This is because they define pattern support to be the number of graphs in the database that contain the pattern. As long as a single embedding is found, the support can be incremented by one, and as such these methods do not have to deal with the problem of enumerating all the embeddings, or computing the maximum number of edge (or node) disjoint embeddings. Also, pattern support, as defined for a database of many graphs, is *anti-monotonic*, i.e., a supergraph cannot have support more than any of its subgraphs. This property allows for fast pruning of candidate patterns during pattern search, since we can prune a pattern (and all of its extensions), when its support falls below a user specified minimum support threshold, $minsup$. However, the number of embeddings is clearly not anti-monotonic. For example, let $minsup = 3$, and let the database graph comprise a node labeled A, connected to two nodes labeled B, and further, let each of the B nodes be connected to three nodes labeled C. In this database graph, the edge A–B has two embeddings (below $minsup$), but the pattern A–B–C has six embeddings (above $minsup$). The lack of anti-monotonicity is clearly a problem for support computation.

Support in a single graph: Several recent approaches have been proposed to tackle the challenges in mining a single graph. Kuramochi and Karypis [11] proposed a support counting measure that is anti-monotonic. They proposed three different formulations for mining a single graph. The first is based on an exact maximum independent set (MIS) of the overlap graph, which gives the exact set of edge disjoint embeddings. The other two approaches are based on approximate MIS, which provide a subset and superset of the edge disjoint embeddings. However, they require enumerating all the embeddings and then discarding the ones that overlap. Since the total number of possible embeddings is exponential, it makes these methods incapable of finding bigger patterns. Further, instead of the MIS, we propose a network-flow based approach. Fidler and Borgelt [12] gives a formal proof that maximum independent set based support counting is anti-monotonic. We also prove that our flow-based upper-bound on the number of edge disjoint embeddings leads to an anti-monotonic pruning criteria. Bringmann and Nijssen [13] proposed an *image-based* support of a pattern, defined as the minimum number of mappings from a vertex in the pattern to a vertex in the graph. Our flow-based approach yields a tighter upper bound compared to the image-based support. Li et al. [14] proposes a method to compute edge disjoint support to find frequent dense subgraphs in a single graph. This method is not suitable for CMDB graphs, since infrastructure patterns are not very dense. Besemann and Denton [15] tackles graphs in which nodes have multiple attributes. The edge disjoint support is computed by constructing a bipartite graph with the original node set (V) and a node for every attribute (U). Edge disjointness is imposed on V, allowing for overlap in the bipartite edges that connect a vertex in V to one of its attributes in U. Recent theoretical work has focused on proving necessary and sufficient conditions for anti-monotonicity of edge overlap based graph support measures [16], and in other generalizations such as homomorphisms and isomorphisms, for labeled and unlabeled, directed and undirected graphs [17].

III. CMDB-Miner: Mining CMDB Graphs

CMDB-Miner has three main steps. Given the particular characteristics of CMDB graphs, first we pre-process them to extract the relevant attributes for each configuration item, and summarize the graph. Second, we perform random walks in the pattern space to extract a sample of the maximal frequent patterns. In the third step, we cluster the maximal patterns (since many of them may be similar) and extract a set of representative patterns from each cluster. The latter constitute the infrastructure patterns presented to the IT practitioners to help manage and set the IT configuration policies throughout the organization. The details of each step are given below.

A. Graph Pre-processing

CMDB graphs have many different types of composite items, and each CI may have many possible attributes (with

various values). Furthermore, there are many degree one nodes, called *leaf nodes* in CMDB graph. Before mining these graphs, we preprocess them in two ways to aid in interpretation and mining. First, we prune attributes based on their entropy, and second, we summarize the multiplicities among the leaf nodes.

Entropy-based Attribute Pruning: Based on the distribution of values for each attribute, we observed that across the various instances of the same CI type, some of the attributes either have a single value, or they have all distinct values. Let $p_v = \frac{m_v}{m}$ be the probability of observing value v for an attribute a of a given CI type, where m is the total number of occurrences of attribute a, and m_v is the number of times a has value v. The entropy of a is defined as $E(a) = -\sum_v p_v \log p_v$. We prune the uninformative attributes, namely those that have very low or very high entropy, by discarding the tails of the entropy distribution (e.g., discarding attributes within the bottom 5% and top 5% of entropy). This results in a significant reduction in the number of attributes.

Summarizing Leaf Nodes: A peculiarity of CMDB graphs is that a vast majority of nodes are *leaf nodes*, defined as those with degree one. Further, an *internal node*, defined as a node with degree more than one, can be connected to many of the same types of leaf nodes, a characteristic we call *node multiplicity*. Some of the CI types like *process*, *ip_address*, etc., have a wide range of multiplicities. CMDB-Miner employs leaf level summarization, which reduces the size of the CMDB graphs significantly, and aids interpretation of the mined infrastructure patterns. For each leaf node u with CI type t, we define its same label siblings as: $Sib(u,t) = \{x | L(x) = t, x \text{ is a leaf node}, x \text{ and } u \text{ have common neighbor}\}$.
For every CI type t, we define its multiplicities as: $Mult(t) = \{m | \exists u,\ u \text{ is a leaf}, L(u) = t, |Sib(u,t)| = m\}$. In other words, the multiplicities of CI type t, is the multiset comprising the number of its occurrences at the leaf level with common internal neighbors. We discretize the multiplicities $Mult(t)$ using equi-width binning. For each internal node connected to leaf nodes, we then attach a new label of the form $x \longrightarrow [l,u]y$, which is interpreted as internal node x having between l and u occurrences of CI type y as a leaf. Fig. 1 shows an example of such a label, namely, *issftpservice* $\longrightarrow [0,2]$ *issftpservice*, meaning that *issftpservice* is connected to up to two other leaf nodes with CI type *issftpservice*.

B. Sampling Maximal Patterns

The goal of this step is to extract a sample of maximal frequent subgraphs from a large, sparse CMDB graph. We do this via random walks in the pattern space, starting from the empty graph, and extending the current candidate pattern by a random edge. After each extension we ensure that the pattern is frequent, according to our new network-flow based approach (described below). Thus random pattern extension and support computation form the two sub-steps for each candidate.

Random Walks in Pattern Space: CMDB-Miner takes as input a parameter k, specifying the number of walks to perform. Each random walk begins with the empty graph, and extends the patterns via random edge extensions. If a random extension yields a frequent pattern, based on the flow-based support described below, it is accepted. Otherwise, the extension is rejected, and we try another random extension. If none of the possible extensions yield a frequent pattern, we are guaranteed that the current pattern is maximal, and we add it to the set of maximal patterns M. It is important to note that, unlike other graph mining approaches that check for isomorphism during pattern growth (to eliminate duplicates), CMDB-Miner does not check for isomorphism until all the k walks finish. This way we pay the price for isomorphism only for patterns that are maximal, and not at each extension. This strategy confers significant efficiency.

Within a given walk, we assume that the edges (and nodes) in P are numbered in the order in which they are added to generate P, starting from an empty graph. Edge ordering automatically leads to node ordering as well. For example, Fig. 2(a) shows a database graph G, and Fig. 2(b) shows a candidate pattern graph P. $e_1 = (p_0, p_1)$ is the first, $e_2 = (p_1, p_2)$ is the second, and $e_3 = (p_2, p_3)$ is the third edge to be added to P. All of these are examples of *forward* edges, i.e., an edge that introduces at least one new node to P. The nodes are ordered from p_0 to p_3. Due to node ordering, a forward edge is implicitly directed from a lower to a higher numbered node. The last edge to be added to complete P is $e_4 = (p_3, p_1)$, and is an example of a *backward* edge, defined to be an edge between existing nodes. A backward edge is implicitly directed from a higher to lower numbered node. This direction information is used in our flow-based support detailed next.

Network-Flow Based Pattern Support: Recall that a *flow network* $G = (V, E)$ is a *directed graph* with two distinguished vertices – source s and sink t. Every ordered edge $(u, v) \in E$ has a capacity $c(u, v) \geq 0$. A flow in this network is a function $f : E \to \mathbb{R}$ that satisfies the following properties: i) capacity constraint: $f(u, v) \leq c(u, v)$, and ii) flow conservation: $\sum_{u \in V} f(u, v) - \sum_{u \in V} f(v, u) = 0$, for all $v \in V \setminus \{s, t\}$. The *value* of a flow is defined as $|f| = \sum_{v \in V} f(s, v)$, and *maximum flow* is a flow with the maximum value. It is known that if all the edge capacities $c(u, v)$ are integers, then there exists a maximum flow with only integer flows on the edges. A *path* from node u to v in a flow network $G = (V, E)$ is a sequence of *distinct* vertices $(v_1, v_2, \ldots, v_k)$ such that $u = v_1$, $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq k - 1$, and $v_k = v$. The *length* of this path is $k - 1$. A path from s to t is also called a s-t path.

We now describe the construction of a flow network in which the maximum flow corresponds to an upper bound on the edge disjoint support of the pattern. The main idea is that any embedding of a pattern can be viewed as a path

from s to t in the flow network, and edge disjointness can be imposed by using unit capacities on the edges.

Consider a pattern $P = (V', E', L')$, and a database graph $G = (V, E, L)$. Let $E' = \{e_1, e_2, \ldots, e_m\}$ be an ordering of the edges in P (e.g., the order in which pattern P was obtained). Recall that each edge is oriented, i.e., it is a forward or backward edge. Let Π_i denote the set of all embeddings in G for a single edge $e_i \in E'$. For example, Fig. 3(a) shows the embeddings for each oriented edge in P. The flow network $F = (V_F, E_F)$ is constructed from the set of embeddings by setting V_F to be the set of distinct nodes over all the edge embeddings Π_i, and by adding the directed edge (a_j, b_j), with capacity $c(a_j, b_j) = 1$, for each embedding $a_j, b_j \in \Pi_i$, with $1 \leq j \leq |\Pi_i|$ and $1 \leq i \leq m$. Further, we add an edge (s, u) for each distinct u such that $(u, v) \in \Pi_1$, with capacity $c(s, u) = n_u$, where n_u is the number of time node u appears in Π_1. Finally, we add an edge (v, t) for each distinct v such that $(u, v) \in \Pi_m$, with capacity $c(v, t) = n_v$, where n_v is the number of times v appears in Π_m. Fig. 3(b) shows the flow network obtained from the embeddings of each edge in P. For instance, since $0, 1 \in \Pi_1$, we add the edge $(0, 1)$ in F with capacity 1. Likewise, since $2, 1 \in \Pi_4$ we add the edge $(2, 1)$ in F with capacity 1. The same is done for all embeddings in Π_j, for $1 \leq j \leq 4$. There are three distinct start nodes in Π_1, namely $\{0, 5, 8\}$, thus we add three edges from the source: $(s, 0)$ with capacity 2, $(s, 5)$ with capacity 1, and $(s, 8)$ with capacity 1. Finally, there are two distinct end nodes in Π_4, namely $\{1, 9\}$, thus we add two edges to the sink: $(1, t)$ with capacity 3, and $(9, t)$ with capacity 1.

Definition. *The flow-based support of a pattern P, denoted $sup_f(P)$, is defined as the maximum flow in the flow network for P.*

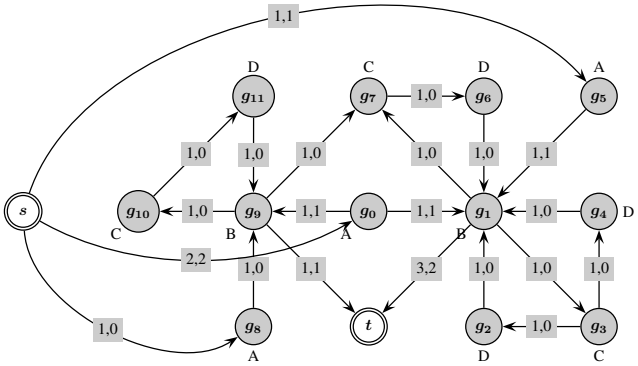
There are several efficient implementations of maximum flow. We use Dinic's algorithm [18] which is based on blocking flows. In special cases where all the edges have a unit capacity it has complexity $O(\min(V^{2/3}, E^{1/2}) \cdot E)$. Fig. 3(b) shows that the maximum flow value is 3, thus $sup_f(P) = 3$. Fig. 3(c) shows the three disjoint edge mappings for P, corresponding to the three disjoint embeddings in Fig. 2(d).

We now prove that the flow-based support of P is an upper-bound on the edge disjoint support. Let $G = (V, E, L)$, and $P = (V', E', L')$ with $|V'| = n$ and $|E'| = m$. We make the following observations:

- LEMMA 1: If π is an embedding of P in G, then there exists a corresponding s-t path in the flow network F. This follows immediately from the facts that: i) P is connected, ii) for each edge $e_i = (a_i, b_i) \in E'$, there is an edge in the flow network corresponding to the edge mapping for e_i, namely $\pi(e_i) = (\pi(a_i), \pi(b_i)) \in \Pi_i$, iii) there exists an edge from s to each start node in Π_1 (for edge $e_1 \in P$), and from each end node in Π_m (for $e_m \in P$) to t. Note that the path length can be less than $m + 2$. It is $m + 2$ when all edges in P lie on some path from s to t.

A–B $e_1(p_0, p_1)$	*B–C* $e_2(p_1, p_2)$	*C–D* $e_3(p_2, p_3)$	*D–B* $e_4(p_3, p_1)$
Π_1	Π_2	Π_3	Π_4
$0, 1$	$1, 3$	$3, 2$	$2, 1$
$0, 9$	$1, 7$	$3, 4$	$4, 1$
$5, 1$	$9, 7$	$7, 6$	$6, 1$
$8, 9$	$9, 10$	$7, 11$	$11, 9$
		$10, 11$	

(a) Edge Embeddings



(b) Flow Network for P

A–B, *B–C*, *C–D*, *D–B* (p_0, p_1), (p_1, p_2), (p_2, p_3), (p_3, p_1)
$(5, 1)$, $(1, 7)$, $(7, 6)$, $(6, 1)$
$(0, 1)$, $(1, 3)$, $(3, 4)$, $(4, 1)$
$(0, 9)$, $(9, 10)$, $(10, 11)$, $(11, 9)$

(c) Disjoint Edge Mappings

Figure 3. Flow Network and Maximum Flow: (a) Edge embeddings for P. (b) Flow network for P. Boxes show capacity and flow on each edge. Maximum flow has value 3. (c) A possible set of three edge disjoint embeddings corresponding to the maximum flow of 3.

- LEMMA 2: If π_1, and π_2 are two edge disjoint embeddings of P in G, then the $s-t$ corresponding paths are disjoint, ignoring the out-edges of s and in-edges of t (which may be shared).
- LEMMA 3: If $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$ is a set of edge-disjoint embeddings of P in G, then the maximum flow is at least k. Let n_u embeddings have the same start vertex u, and let n_v embeddings have the same end vertex v. From Lemma 2, we know that ignoring s and t, there are k disjoint paths in the flow network k, corresponding to each of the k embeddings in Π. If $f(e) = 1$ for all edges e on these paths, and if $f(s, u) = n_u$ and $f(v, t) = n_v$, then we can see that the resulting flow has value at least k.

Theorem 1. *The maximum flow in the flow network F for pattern P is an upper bound on the number of edge disjoint embeddings of P in G.*

Proof: In Lemma 3, if Π is the set of all possible edge-disjoint embeddings of P, with $|\Pi| = k$, then the maximum flow in F is at least k, and $sup_f(P) \geq k = sup_e(P)$. Let Q

be an extension of pattern P, i.e., $P \subseteq Q$. Since every edge disjoint embedding of Q is also an edge disjoint embedding of P it immediately implies that $sup_e(Q) \leq sup_e(P) \leq sup_f(P)$. ■

The fact that $sup_f(P)$ is an upper-bound on the edge disjoint support allows us to prune any extension (an immediate supergraph) of pattern P if $sup_f(P) < minsup$. This follow immediately from the theorem above, since $sup_f(P) < minsup \implies sup_e(Q) < minsup$, and thus we can guarantee that no extension of P can be frequent according to edge disjoint support.

It is worth noting that the edge-disjoint support of P is equal to the maximum number of edge disjoint $s-t$ paths of length $m+2$ in the flow network. However, [19] proved that finding the maximum disjoint paths with constraints on the length is NP-Complete. For this reason our formulation does not place any restrictions on the length of the paths, and thus we obtain an upper-bound on the edge-disjoint support. It is important to note that Dinic's algorithm finds the shortest $s-t$ paths that are saturated. Thus the flow-based support is close to the actual support if the shortest $s-t$ path length in the flow network is close to the number of edges in the candidate pattern. While it may not be a beneficial strategy for general (dense) patterns, our formulation is very effective for CMDB graphs, which are sparse, and thus the mined patterns are also sparse. For such patterns the flow-based support is generally close to the edge disjoint support.

Pruning Isomorphic Patterns: Given a minimum support threshold $minsup$, and given k, the number of random walks, CMDB-Miner performs k random walks in the pattern space, to yield a set M of exactly k maximal frequent subgraphs, using flow-based support. However, since the walks are random, they may yield isomorphic maximal patterns. Such isomorphic patterns have to be discarded before the infrastructure pattern extraction step. Unfortunately, while graph isomorphism is in NP, it is not known whether it is NP-complete or is in P [20].

Instead of checking for isomorphism between every pair of maximal patterns in M, we use a sequence of polynomial-time filters to create equivalence classes of possibly isomorphic patterns. Thus, the worst-case exponential time algorithm for graph isomorphism method has to be applied to only pairs of graphs within the same equivalence class. Initially M comprises a single equivalence class. We then apply the following filters:

- NODE MULTISET: Given a pattern $P = (V', E', L')$, define $\rho_V(P) = \{L(v_i) : v_i \in V\}$ to be the *multiset* of node labels in P. It is easy to see that two patterns P and P' cannot be isomorphic if $\rho_V(P) \neq \rho_V(P')$. In this case P and P' are put into different equivalence classes, and never have to be checked for isomorphism.
- EDGE MULTISET: Given pattern $P = (V', E', L')$, for each edge $e_i = (a_i, b_i) \in E'$, define a *composite edge label* to be the triple $\mathcal{L}(e_i) = (L'(a_i), L'(b_i), L'(e_i))$, with $a_i < b_i$. Define the filter $\rho_E(P) = \{\mathcal{L}(e_i) : e_i \in E'\}$ to be the *multiset* of composite edge labels for P. Two patterns P and P' cannot be isomorphic if $\rho_E(P) \neq \rho_E(P')$.
- LAPLACIAN SPECTRUM: Let A be the adjacency matrix for pattern P, i.e., $A(v_i, v_j) = 1$ if $(v_i, v_j) \in E'$, and $A(v_i, v_j) = 0$, otherwise. Let D be the diagonal degree matrix for P, defined as $D(v_i, v_i) = \sum_{v_j} A(v_i, v_j)$, and $D(v_i, v_j) = 0$ for all $v_i \neq v_j$. Define the *normalized Laplacian matrix* of P as follows: $N = D^{-1/2} \cdot (D - A) \cdot D^{1/2}$. N is a $n \times n$ positive semi-definite matrix, and thus N has n (not necessarily distinct) real, positive eigen-values: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Define the Laplacian spectrum of P as the multiset $\rho_S(P) = \{\lambda_i : 1 \leq i \leq n\}$. It is known that two isomorphic patterns are iso-spectral, i.e., they have the same Laplacian spectrum [20]. Thus, P and P' cannot be isomorphic if $\rho_S(P) \neq \rho_S(P')$

After applying the above filters, the set M is partitioned into smaller equivalence classes of possibly isomorphic graphs. For each pair of graphs in the same class, we perform full isomorphism checking using the VF2 [21] algorithm. The output of this step is the final set M of non-isomorphic maximal frequent patterns in G. Note that at this stage we can find the actual edge disjoint support of all the maximal patterns by using the maximal independent set approach proposed in [11].

IV. INFRASTRUCTURE PATTERN EXTRACTION

Given a set of non-isomorphic maximal patterns M, CMDB-Miner clusters them into groups of similar patterns, and then selects a representative set of infrastructure patterns from each cluster. There are three main steps: i) defining pair-wise similarities between patterns, ii) graph clustering, iii) infrastructure pattern extraction.

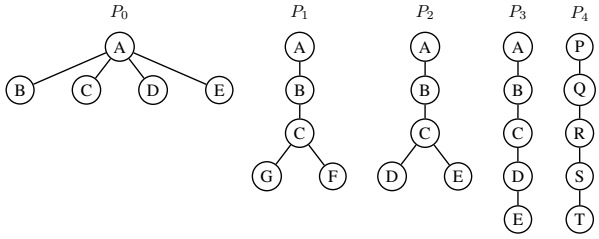


Figure 4. Sample Maximal Patterns

A. Pattern Similarity

Before clustering the maximal patterns, we have to define a similarity measure between patterns, that takes into account both the structure and label information. Graph edit distance based methods [22] are a popular approach to compute the similarity, however, the vast majority of these methods focus mainly on the structure. For example, a purely structure based method would consider P_3 and P_4 in Fig. 4 to be highly similar. Methods that consider labels include [23], [24]. We propose a novel pattern similarity approach based on diffusion kernels [25], which works well for CMDB graphs. As such the clustering method is

independent of the similarity measure, and thus any of the attributed graph similarity measures can also be used.

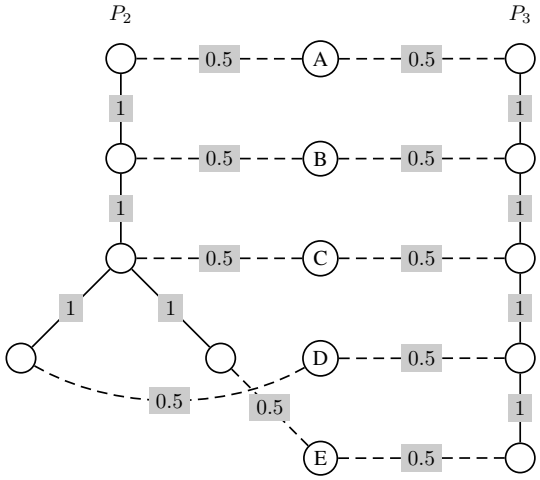
We define similarity between two patterns $P = (V_P, E_P, L_P)$ and $Q = (V_Q, E_Q, L_Q)$, as

$$Sim(P, Q) = Jaccard(P, Q) \times Diffusion(P, Q) \quad (1)$$

Here $Jaccard(P, Q) = \frac{|L_P \cap L_Q|}{|L_P \cup L_Q|}$ is the Jaccard coefficient between the label sets for P and Q. The more the labels in common, the higher the Jaccard similarity. $Diffusion(P, Q)$ is the diffusion kernel based similarity between P and Q that considers both the structure and the label information, as described below.

Following a procedure similar to that in [26], given P and Q, we first create an augmented weighted graph $R = (V_R, E_R, W_R)$. Here $V_R = V_P \cup V_Q \cup \{l | \exists v \in V_P, L_P(v) = l\} \cup \{l | \exists v \in V_Q, L_Q(v) = l\}$, i.e., R contains both structural nodes (those in P and Q) and attribute nodes (labels for nodes in P and Q). $E_R = E_P \cup E_Q \cup \{(v, l) : v \in V_P, L_P(v) = l\} \cup \{(v, l) : v \in V_Q, L_Q(v) = l\}$. In other words, E_R contains both the structural edges (the original edges between vertices in both P and Q), as well as the attribute edges (between a node in P and Q, and its label). Finally, $W_R : E_R \to \mathbb{R}$ is a function that assigns a weight to each edge. The weights on structural edges are set to 1, i.e., $W(u, v) = 1.0$ for all $(u, v) \in E_P \cup E_Q$. The weights on attribute edges are set as follows: $W(v, l) = \frac{1}{n_l}$, where n_l is the number of neighbors of node l in R. In the augmented graph, two structural vertices that have the same label l, are both neighbors of the attribute node l. To avoid inflating the similarity purely due to labels (which has already been accounted for by $Jaccard(P, Q)$), we assign the fractional weight on attribute edges. Fig. 5(a) shows the augmented weighted graph for P_2 and P_3 from Fig. 4.

To compute $Diffusion(P, Q)$ for each pair of patterns, we use the diffusion kernel approach [25] over their augmented graph. A diffusion kernel mimics the physical process of diffusion where heat, gases, etc., originating from a point diffuse with time. On graphs, it is the local similarity that diffuses via continuous time random walks (i.e., with an infinite number of infinitesimally small steps). Given the augmented graph $R = (V_R, E_R, W_R)$, the matrix W_R is taken to be the weighted adjacency matrix of R. Further, define the diagonal degree matrix as $D(v_i, v_i) = \sum_{v_j} W_R(v_i, v_j)$, and $D(v_i, v_j) = 0$ when $i \neq j$. The *Laplacian matrix* of R is then defined as: $N = D - W_R$. Finally, the diffusion kernel matrix is defined as $K = e^{\beta L} = \sum_{k=0}^{\infty} \frac{\beta^k}{k!} L^k$, where β is a real-valued diffusion parameter, and $e^{\beta L}$ is the matrix exponential (with $L^0 = I$ and $0! = 1$). Since L is positive semi-definite, it has $|V_R| = n$ real and positive eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Let $\boldsymbol{u}_i$ be the eigenvector corresponding to eigenvalue λ_i. Then the diffusion kernel can easily be computed as the spectral sum [25]: $K = \sum_{i=1}^{n} \boldsymbol{u}_i e^{\beta \lambda_i} \boldsymbol{u}_i^T$. The eigenvalues and eigenvectors of K can be computed in $O(n^3)$ time, where $n = |V_R|$.



(a) Augmented Weighted Graph

$\searrow$	P_0	P_1	P_2	P_3	P_4
P_0	**0.47**	0.10	0.26	0.25	0.0
P_1	0.10	**0.40**	0.06	0.09	0.0
P_2	0.26	0.06	**0.40**	0.33	0.0
P_3	0.25	0.09	0.33	**0.37**	0.0
P_4	0.0	0.0	0.33	0.0	**0.47**

(b) Similarity Matrix

Figure 5. Augmented Graph and Pattern Similarity: (a) shows the augmented weighted graph for P_2 and P_3 in Fig. 4. Structural edges are solid, whereas attribute edges are shown dashed. (b) shows the pair-wise similarity matrix between all five patterns in Fig. 4.

The kernel matrix entry $K(v_i, v_j)$ gives the diffusion-based similarity between any two vertices in the augmented graph R for patterns P and Q. In particular, we are interested in those entries $K(u, v)$ where $u \in V_P$ and $v \in V_Q$. We define the diffusion similarity between P and Q as follows: If $L_P \cap L_Q = 0$, then we set $Diffusion(P, Q) = 0$, otherwise

$$Diffusion(P, Q) = \min_{l \in L_P \cap L_Q} \left\{ \max_{\substack{u \in V_P, v \in V_Q \\ (u,l),(v,l) \in E_R}} \{K(u, v)\} \right\}$$

In other words, the diffusion similarity between P and Q is defined as the least label similarity over all labels l, such that the label similarity is the maximum kernel similarity over pairs of nodes u, v that share a given label l. Fig. 5(b) shows the pairwise similarities between all the patterns in Fig. 4, based on Eq. (1), that combines both the $Jaccard()$ and $Diffusion()$ values.

B. Clustering

We employ graph clustering to cluster the set of maximal patterns M. In particular, given the similarity matrix $S(i, j) = Sim(P_i, P_j)$ between any two patterns $\in M$, we can think of S as the weighted adjacency matrix of a *similarity graph*, where each maximal pattern is a node, and any two maximal patterns are linked with weight $S(i, j)$. Clustering of the patterns is then equivalent to clustering the

nodes in the similarity graph. While many algorithms have been proposed for graph clustering [27], we use the Markov clustering (MCL) [28] approach as opposed to spectral methods [29], since MCL does not require the number of clusters as input.

Let D be the diagonal degree matrix corresponding to the weighted similarity matrix S. Let $N = D^{-1}S$ be the normalized adjacency matrix for the similarity graph. The matrix N is a row-stochastic or Markov matrix that specifies the probability of jumping from node P_i to any other node P_j. N is thus that transition matrix for a Markov random walk on the similarity graph. As such, the k-th power of N, namely N^k, specifies the probability of transitioning from P_i to P_j in a walk of k steps. MCL [28] takes successive powers of N to *expand* the influence of a node. However, it damps the extent of a nodes' influence, by an *inflation* step, whose goal is to enhance higher and diminish lower transition probabilities. Given transition matrix N, define the inflation operator Υ, given as follows: $\Upsilon(N, r) = \left\{\frac{N(i,j)^r}{\sum_{a=1}^{n} N(i,a)^r}\right\}_{i,j=1}^{n}$. In essence, Υ takes each element of N to the r-th power, and then re-normalizes the rows to make the matrix row-stochastic.

Given the initial N matrix, and an inflation parameter r, MCL is an iterative matrix algorithm consisting of two main steps: i) expansion: $N = N^2$, followed by ii) inflation: $N = \Upsilon(N, r)$. The method converges to a doubly idempotent matrix, and the strongly connected components in the corresponding induced graph yield the final node clusters [28]. The only parameter in MCL is the inflation value r that controls the granularity. Higher values lead to more, smaller clusters, whereas smaller values lead to fewer, larger clusters. MCL runs in $O(tn^3)$ time, where $|M| = n$, and t is the number of iterations until convergence.

C. Infrastructure Pattern Extraction

Given a set of clusters C_i, $1 \leq i \leq k$ obtained via the MCL approach, the final step in CMDB-Miner is to extract the so-called infrastructure patterns, i.e., representative members from each cluster. Given a similarity threshold θ, from each cluster C_i we aim to extract as subset of the patterns $R_i \subseteq C_i$, such that for each $P_j \in C_i$, there exists a pattern $P \in R_i$ with $Sim(P_j, P) \geq \theta$. The task is to find a minimal set of representative patterns for each cluster. However, this problem is equivalent to smallest set cover, an NP-Complete problem, which nevertheless has a greedy $\Theta(\log n)$ approximation algorithm [30]. The greedy heuristic iteratively chooses the pattern that covers or represents the largest number of remaining elements in a cluster, until all the cluster members are covered.

V. Experimental Evaluation

In this section we evaluate CMDB-Miner on real-world CMDB graphs for two multi-national corporations, company A and B (names not revealed due to non-disclosure issues), from HP's Universal Configuration Management Database (UCMDB). We also conduct experiments to validate some of the design choices in the implementation of CMDB-Miner. All experiments were performed on a machine with 2.67GHz Intel i7 processor with 4GB of memory running Ubuntu Linux version 10.04.

Table I
CMDB Graphs A,B: before and after preprocessing

	A		B	
Property	Before	After	Before	After
$\|V\|$	443192	11363	455012	57525
$\|E\|$	480143	20978	523415	149229
Avg. Deg.	2.16	3.68	2.3	5.16

Table II
Biggest pattern extracted

Database	# Vertices	# Edges
A	24	41
B	54	55

A. Preprocessing

The raw CMDB graph of company A contains 443,192 vertices and 480,143 edges. Company B also contains a similar number of nodes and edges, and is shown in Table I. We discard uninformative attributes for each composite item, by discarding both high and low entropy attributes. This results in a significant reduction in the number of attributes, as shown in Figure 6 for some of the common CI types in the CMDB graph of company A (similar results are obtained for B too). More than 75% of the attributes are pruned in this stage. Further, collapsing leaf nodes reduces the total number of vertices to 11,363. Table I shows the graph order and size, as well as average degree, both before and after preprocessing. As pointed out earlier, these two preprocessing steps also aid in better interpretation of the final infrastructure patterns.

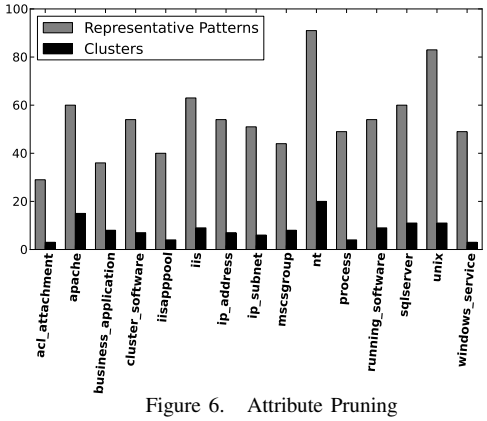
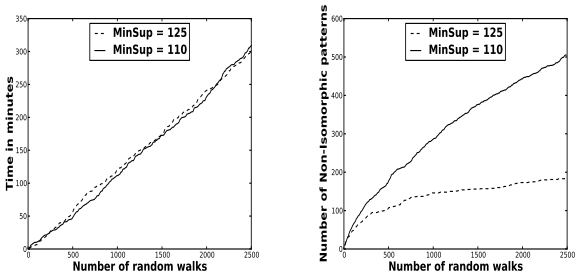


Figure 6. Attribute Pruning

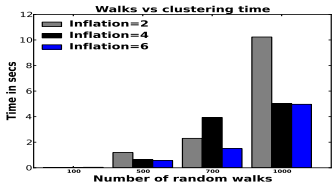
B. Sampling Maximal Patterns

Fig. 7(a) shows the time for sampling maximal patterns versus number of random walks, for two different (absolute) values of minimum support for company A. We can see that as expected time is linear in the number of walks. Fig. 7(b) shows the number of distinct or non-isomorphic maximal

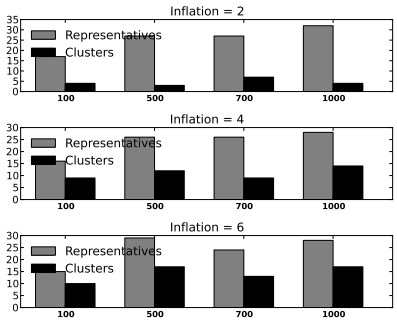


(a) Time (b) Distinct Patterns

Figure 7. Maximal and Non-Isomorphic Patterns for Company A: (a) sampling time and (b) number of distinct maximal patterns, versus number of random walks.



(a) Clustering Time



(b) Clusters and Representative Patterns

Figure 8. (a) shows time to extract clusters, and (b) shows the number of clusters and representative infrastructure patterns, for different values of inflation parameter.

Table III
ISOMORPHISM CHECKING FILTERS (TIME IN SEC)

Walks	Non-Isomorphic	Filtering	VF2
100	89	0.17	2.05
500	361	5.26	82.07
700	464	3.14	47.57
1000	671	10.96	148.66

patterns versus number of random walks. We can see that for $minsup = 125$ the fraction of distinct maximal patterns decreases with the number of walks, indicating convergence to the "true" set of maximal patterns. The convergence has not yet been reached for $minsup = 110$ within 2500 walks. These curves suggest an automated method to stop sampling, namely, when the fraction of distinct patterns versus number of walks falls below some threshold. Similar results were obtained for company B (not shown here due to space constraints), though Table II shows the order/size of the largest maximal pattern for both A and B.

Table III shows the time to detect the number of distinct maximal patterns. We compare the time taken by our filter based approach versus the cost of running the VF2 algorithm on each pair of patterns in M. It is clear that the sequence of filters is very effective in reducing the running time by over an order of magnitude.

C. Infrastructure Pattern Extraction

For company A, Fig. 8 shows the clustering time, and the number of clusters and representative patterns versus the given number of walks, for different values of the inflation parameter $r = 2, 4, 6$. We used $\theta = 0.9$ (threshold for a pattern to represent another pattern), and $\beta = 2$ (the diffusion kernel parameter). Clustering time is negligible compared to the time to sample the set of maximal patterns. The number of clusters increases with increase in the inflation parameter, as expected. Also, in most cases the number of representative patterns remains the same. This is due to the characteristics of CMDB graphs, where each CI type is connected to only a limited number of other CI types. Thus, most of the maximal patterns either contain very similar or very different node labels. As the similarity measure is based on the attributes, these patterns tend to remain in the same cluster or different clusters, respectively. The small number of representative patterns shows the effectiveness of CMDB-Miner in summarizing large and sparse CMDB graphs into a small set of infrastructure patterns.

D. Example Infrastructure Patterns

Fig. 9 shows three mined maximal patterns. which are a partial view of a general construct that is known in CMDB, and is defined as a standard Topological Query – *Node = nt, sqlserver = running_software*. Fig. 10 shows another infrastructure pattern mined by CMDB-Miner. This pattern is a representative for several other patterns in a cluster. In order to choose a representative for each cluster we currently choose a member of the cluster that maximizes the overall similarity to other members of the cluster. Alternatively, we can consider trimmed similarity (to say only the closest 80% of the members), or we could aim for a description of a family of graphs that describes a large majority of the cluster. Exploring these options is part of future work.

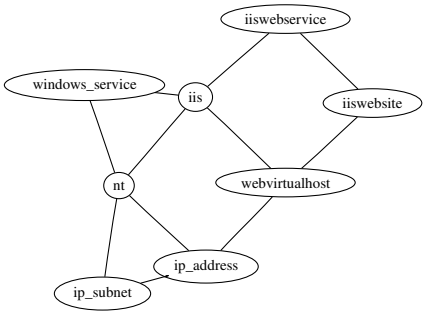
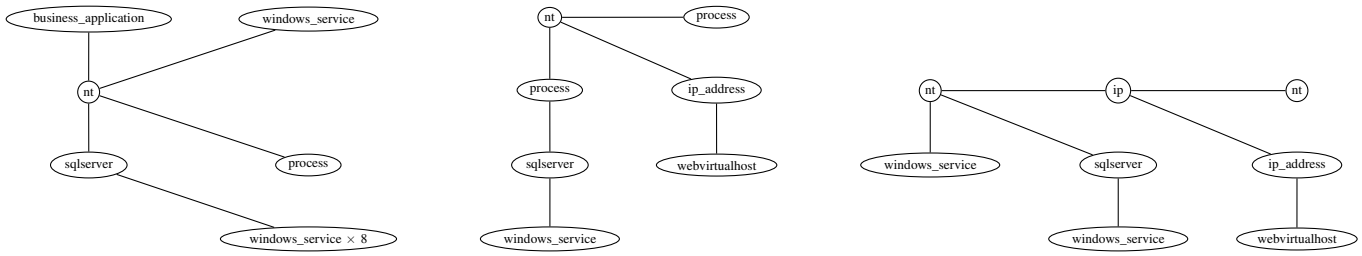


Figure 10. Infrastructure Pattern

VI. CONCLUSIONS

We have demonstrated that CMDB-Miner is an effective algorithm for mining real-world CMDB graphs. It makes use



(a) (b) (c)

Figure 9. Mined Maximal Patterns

of the characteristics of such graphs (e.g., label multiplicities, sparsity) to speed up the mining process. It performs random walks without having to check for isomorphism, which is only performed on the final set of maximal patterns via a filter-based approach. Further, we proposed a new flow-based upper-bound on the edge-disjoint support that allows for effective pattern pruning, and which avoids the exponential blowup in the number of possible embeddings that plague many previous methods. To extract the infrastructure patterns we proposed a new diffusion kernel based similarity that takes into account both the structure and label information. We show that CMDB-Miner is able to extract meaningful infrastructure patterns. In terms of future work, we plan to parallelize the approach for better scalability, and to extend the approach to mining graphs with multiple attributes on the nodes and edges. We would also like to mine approximate patterns, with possibly mismatched nodes and edges.

REFERENCES

[1] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *1st IEEE Int'l Conf. on Data Mining*, Nov. 2001.

[2] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *IEEE Int'l Conference on Data Mining*, 2002.

[3] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *IEEE Int'l Conf. on Data Mining*, 2003.

[4] A. Inokuchi, T. Washio, and H. Motoda, "Complete mining of frequent patterns from graphs: Mining graph data," *Machine Learning*, vol. 50, no. 3, pp. 321–354, 2003.

[5] V. Chaoji, M. A. Hasan, S. Salem, and M. J. Zaki, "An integrated, generic approach to pattern mining: data mining template library," *Data Mining and Knowledge Discovery*, vol. 17, no. 3, pp. 457–495, Dec. 2008.

[6] M. A. Hasan and M. J. Zaki, "Output space sampling for graph patterns," *Proceedings of the VLDB Endowment (35th International Conference on Very Large Data Bases)*, vol. 2, no. 1, pp. 730–741, 2009.

[7] C. Chen, C. Lin, X. Yan, and J. Han, "On effective presentation of graph patterns: a structural representative approach," in *17th ACM conference on Information and knowledge management*, 2008.

[8] Y. Liu, J. Li, and H. Gao, "Summarizing graph patterns," *IEEE Transactions on Knowledge and Data Engineering*, p. 10.1109/TKDE.2010.48 (online early access), 2011.

[9] S. Zhang, J. Yang, and S. Li, "Ring: An integrated method for frequent representative subgraph mining," in *9th IEEE International Conference on Data Mining*, 2009.

[10] V. Chaoji, M. A. Hasan, S. Salem, J. Besson, and M. J. Zaki, "ORIGAMI: A Novel and Effective Approach for Mining Representative Orthogonal Graph Patterns," *Statistical Analysis and Data Mining*, vol. 1, no. 2, pp. 67–84, Jun. 2008.

[11] M. Kuramochi and G. Karypis, "Finding Frequent Patterns in a Large Sparse Graph*," *Data Mining and Knowledge Discovery*, vol. 11, no. 3, pp. 243–271, 2005.

[12] M. Fiedler and C. Borgelt, "Support computation for mining frequent subgraphs in a single graph," in *5th International Workshop on Mining and Learning with Graphs*, 2007.

[13] B. Bringmann and S. Nijssen, "What is frequent in a single graph?" in *12th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2008.

[14] S. Li, S. Zhang, and J. Yang, "Dessin: mining dense subgraph patterns in a single graph," in *Scientific and Statistical Database Management Conference*, 2010.

[15] C. Besemann and A. Denton, "Mining edge-disjoint patterns in graph-relational data," in *Workshop on Data Mining for Biomedical Informatics (at SDM)*, 2007.

[16] N. Vanetik, S. Shimony, and E. Gudes, "Support measures for graph data," *Data Mining and Knowledge Discovery*, vol. 13, no. 2, pp. 243–260, 2006.

[17] T. Calders, J. Ramon, and D. Van Dyck, "All normalized anti-monotonic overlap graph measures are bounded," *Data Mining and Knowledge Discovery*, vol. 10.1007/s10618-011-0217-y (online first), 2011.

[18] Y. Dinitz, "Dinitz algorithm: The original version and even's version," *Theoretical Computer Science*, pp. 218–240, 2006.

[19] A. Itai, Y. Perl, and Y. Shiloach, "The complexity of finding maximum disjoint paths with length constraints," *Networks*, vol. 12, pp. 277–286, 1982.

[20] D. Cvetkovic, P. Rowlinson, S. Simic, and N. Biggs, *Eigenspaces of graphs*. Cambridge University Press Cambridge, UK, 1997.

[21] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 10, pp. 1367–1372, 2004.

[22] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern recognition letters*, vol. 19, no. 3-4, pp. 255–259, 1998.

[23] D. Hidovic and M. Pelillo, "Metrics for attributed graphs based on the maximal similarity common subgraph," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 18, no. 3, pp. 299–313, 2004.

[24] M. Neuhaus, K. Riesen, and H. Bunke, "Fast suboptimal algorithms for the computation of graph edit distance," *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 163–172, 2006.

[25] R. Kondor and J.-P. Vert, "Diffusion kernels," in *Kernel Methods in Computational Biology*, B. Scholkopf, K. Tsuda, and J.-P. Vert, Eds. The MIT Press, 2004.

[26] Y. Zhou, H. Cheng, and J. Yu, "Graph clustering based on structural/attribute similarities," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 718–729, 2009.

[27] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, August 2007.

[28] S. V. Dongen, "Graph clustering via a discrete uncoupling process," *SIAM J. on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 121–141, 2004.

[29] J. Shi and J. Malik, "Normalized cuts and image segmentation." *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, August 2000.

[30] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, pp. 233–235, 1979.