Vertex Unique Labelled Subgraph Mining for Vertex Label Classification

Wen Yu, Frans Coenen, Michele Zito, and Subhieh El Salhi

Department of Computer Science, University of Liverpool,
Ashton Building, Ashton Street, Liverpool, L69 3BX, UK
{yuwen,coenen,michele,hsselsal}@liverpool.ac.uk

Abstract. A mechanism is presented to classify (predict) the values associated with vertices in a given unlabelled graph or network. The proposed mechanism is founded on the concept of Vertex Unique Labelled Subgraphs (VULS). Two algorithms are presented. The first, the minimal Right-most Extension VULS Mining (minREVULSM) algorithm, is used to identify all minimal VULS in a given graph or nework. The second, the Match-Voting algorithm, is used to achieve the desired VULS based classification (prediction). The reported experimental evaluation demonstrates that by using the minimal VULS concept good results can be obtained in the context of a sheet metal forming application used for evaluation purposes.

Keywords: Data mining, Graph mining, Vertex unique labelled subgraph mining, Classification

1 Introduction

This paper introduces a novel classification system founded on the concept of Vertex Unique Labelled Subgraphs (VULS) coupled with a Match-Voting algorithm to predict the unknown vertex labelling in a given graph or network G. The basic idea is that, given a suitably defined training set, if we can identify subgraphs that have unique vertex labelings associated with them we can use this knowledge to predict the vertex labelings for previously unencountered graphs. We refer to such subgraphs as Vertex Unique labelled Subgraphs (VULS). There are two elements to the proposed mechanism: (i) the process for identifying individual VULS in a vertex labelled training graph, and (ii) the process for matching the identified VULS to vertex-unlabelled testing graph so that vertex labels can be predicted. There are two challenges with respect to the first process:

1. The need to identify a sufficiently comprehensive set of VULS (a collection of VULS that will ensure good "coverage" with respect to unseen data).
2. The large number of potential VULS that can be contained in a reasonably sized graph (the identification process thus needs to be efficient).

With respect to the second challenge the issue is how to address the situations where, with respect to a specific vertex in unseen data, either: (i) several competing VULS can be used to label the vertex, or (ii) no appropriate VULS can

be found. With respect to the first process the minimal Right-most Extension VULS Mining (minREVULSM) algorithm is presented to find all minimal VULS in a given graph or network. With respect to the second process we present the Match-Voting algorithm.

The proposed VULS based prediction mechanism has a variety of applications. For example given a social network we can use the VULS concept to identify unique structures in the network and consequently to classify the "type" of specific nodes in the network that are covered by VULS. However, to act as a focus for the work described in this paper, a sheet metal forming application is considered. More specifically Asymmetric Incremental Sheet Forming (AISF) [1–5]. An issue with sheet metal forming processes, such as AISF, is that distortions (referred to as "springback") are introduced as a result of the application of the process. These distortions are non-uniform across the "shape" but tend to be related to local geometries. The intuition is that by utilising a system, such as the proposed VULS based systems, to predict springback in a proposed shape (prior to its manufacture) knowledge of this springback can be used to formulate some form of mitigating error correction to the shapes definition.

The rest of this paper is organised as follows. In section 2 we provide a formalism for the concept of VULS together with some examples. The minREVULSM algorithm is then presented in section 3, and the Match-Voting algorithm in Section 4. An experimental analysis of the proposed algorithms, in the context of sheet metal forming, is presented in section 5. Section 6 then presents some conclusions and summarises the work and main findings.

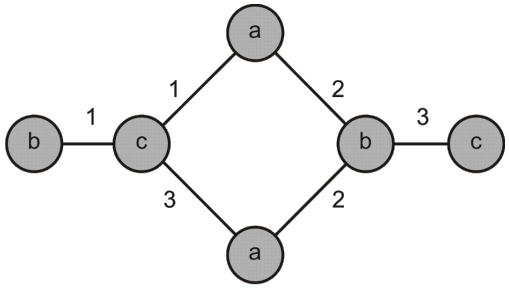


Fig. 1. Example input graph G_1

2 Formalism

A graph G is usually defined in terms of (V, E, L_V, L_E, F), where V is a set of n vertices,such that $V = \{v_1, v_2, \ldots, v_n\}$; E is a set of m edges, such that $E = \{e_1, e_2, \ldots, e_m\}$. The vertices are labelled according to a set of p vertex labels $L_V = \{l_{v_1}, l_{v_2}, \ldots, l_{v_p}\}$. The edges are labelled according to a set of q edge labels $L_E = \{l_{e_1}, l_{e_2}, \ldots, l_{e_q}\}$. F is some function that maps the vertices and edges onto the labels. A graph G can also be conceptualised as comprising k one-edge subgraphs: $G = \{P_1, P_2, \ldots, P_k\}$, where P_i is a pair of vertices linked by an edge, thus $P_i = \langle v_a, v_b \rangle$ (where $v_a, v_b \in V$). The size of a graph G ($|G|$) can thus be defined in terms of its one edge sub-graphs, we refer to 1-edge subgraphs, 2-edge subgraphs and so on up to k-edge subgraphs. We use the

notation $P_i.v_a$ and $P_i.v_b$ to indicate the vertices v_a and v_b associated with a particular vertex pair P_i, and the notation $P_i.v_a.label$ and $P_i.v_b.label$ to indicate the labels associated with $P_i.v_a$ and $P_i.v_b$ respectively. We indicate the sets of labels which might be associated with $P_i.v_a$ and $P_i.v_b$ using the notation $L_{P_i.v_a}$ and $L_{P_i.v_b}$ ($L_{P_i.v_a}$, $L_{P_i.v_b} \in L_V$). We indicate the edge label associated with P_i using the notation $P_i.label$ ($P_i.label \in L_E$). We also define a function, $getVertexLabels$, that returns the potential list of labels S that can be assigned to the vertices in G_2 according to G: $getVertexLabels(G_2) \rightarrow S$, where $G_2 = \{P_1, P_2, \ldots, P_k\}$ and $S = [[L_{P_1.v_a}, L_{P_1.v_b}], [L_{P_2.v_a}, L_{P_2.v_b}], \ldots, [L_{P_k.v_a}, L_{P_k.v_b}]]$ (recall that $L_{P_i.v_a}$ and $L_{P_i.v_b}$ are the sets of potential vertex labels for vertex v_a and v_b associated with a one-edge subgraph P_i).

We typically define the size of G in terms of the number of edges $|E|$. Given a vertex labelled graph $G_1 = (V_1, E_1, L_{V_1}, L_{E_1}, F_1)$ and a vertex unlabelled subgraph $G_2 = (V_2, E_2, L_{V_2}, L_{E_2}, F_2)$ such that $|E_2| \leq |E_1|$, $L_{E_2} \subseteq L_{E_1}$, $L_{V_1} \neq \{\}$ and $L_{V_2} = \{\}$, we can attempt to match G_2 to G_1 so as to populate L_{V_2}. There may be zero, one or more matches. If we have only one match then G_2 is a VULS. A VULS is a minimal VULS if none of its sub-graphs are also VULS.

Table 1. 1-edge VULS

Num.	Candidate VULS (G_2)	Populated Candidate VULS
1	? 1 ?	a 1 c; b 1 c
2	? 2 ?	a 2 b
3	? 3 ?	a 3 c; b 3 c

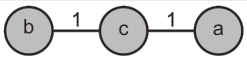
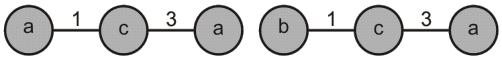
Table 2. 2-edge VULS

Num.	Candidate VULS (G_2)	Populated Candidate VULS
1	? 1 ? 1 ?	b 1 c 1 a
2	? 1 ? 2 ?	c 1 a 2 b
3	? 1 ? 3 ?	a 1 c 3 a; b 1 c 3 a
4	? 2 ? 2 ?	a 2 b 2 a
5	? 2 ? 3 ?	a 2 b 3 c
6	? 3 ? 3 ?	

Considering the example graph G_1 presented in Figure 1. We can identify a number of potential VULS in this graph. Considering only 1-edge VULS we can identify three potential VULS as shown in the left-hand column of Table 1. Populating these candidate VULS, using the vertex labels in G_1, we get the labelings shown in the right-hand column of Table 1. Note that the second 1-edge subgraph, featuring edge 2 is a VULS because it has only one possible vertex labelling associated with it, the remainder have more than one vertex labellings. If we now consider the 2-edge candidate VULS we can identify six such candidate VULS as listed in the left-hand column of Table 2. When we attempt to match these to G_1 we find that the first five exist in G_1 and the sixth does not. Note also that candidate VULS one, two, four and five are all VULS because they have unique vertex labeling, while candidate VULS three is not a VULS because it has more than one possible vertex labellings. No vertex labelling could be matched with the sixth candidate VULS so by default it is not a VULS.

Returning to Table 1 note that the identified one edge VULS (graph 2) is in fact a minimal VULS because it has no sub-graphs that are also VULS. If we only expand the non-minimal 1-edge VULS listed in the left-hand column of Table 1 we get the 2-edge minimal VULS candidates listed in the left-hand column of Table 3. The possible labelings are shown in the right-hand column of Table 3 (the first candidate is also a minimal VULS). Note that with respect to coverage, using only minimal VULS does not necessarily produce as good a coverage as in the case when all VULS are used (inspection of Figure 1 and Tables 1 and 3 indicates that the right hand edge of G_1 is not covered when only minimal VULS are considered, while when using all VULS a coverage of 100% is obtained). However, finding only minimal VULS clearly requires less computational resource. The effect of the trade off between coverage and efficiency is one of the issues that the work presented in this paper seeks to identify.

Table 3. 2-edge minimal VULS when 1-edge minimal VULS are not expanded

Num.	Candidate minimal VULS (G_2)	Populated Candidate minimal VULS
1		
2		
3		

3 The minREVULSM algorithm

The proposed minREVULSM algorithm is presented in this section. The algorithm is partially influenced by the well known gSpan algorithm [7] in that the graph representation, candidate generation process (right most extension and isomorphism testing) are similar. The pseudo code for minREVULSM is presented in Algorithms 1 and 2. Algorithm 1 presents the high level control structure while Algorithm 2 the detail for determining whether a specific subgraph is a VULS or not. Considering Algorithm 1 first, the algorithm comprises

one main procedure (*main*) and a sub-procedure ($genMinVULS$). The algorithm commences with an input graph G_{input} and a parameter max that defines the maximum size for a desired minimal VULS. If we do not limit the size of the searched-for VULSs the entire input graph may ultimately be identified as a minimal VULS which in the context of the target application will not be useful. The output is a set of minimal VULS R. Note that all graphs are encoded using Minimal Depth First Search (DFS) lexicographical ordering (as also used in gSpan [7]). The global variable G (line 7 in Algorithm 1) is the part of the graph G_{input} which is not covered by any of the identified minimal VULS so far. The global variable *coverage* (line 8) is employed to determine whether G_{input} is covered completely by the minimal VULS identified so far (if so the algorithm stops). The *coverage* is the percentage of the number of vertices "covered" by the detected minimal VULS so far compared to the total number of vertices in the input graph G_{input} (Equation 1) (as will become apparent later in this paper, when using VULS for classification purposes, high coverage is desirable). The global variable T_k (line 9) is the set of k-edge non-VULS which will be extended further to produce $(k+1)$-edge candidate VULS. Note that at the start of the procedure, G will be equal to G_{input} and coverage will be 0. We proceed in a breadth first manner starting with one-edge subgraphs ($k = 1$), then two edge sub-graphs ($k = 2$), and so on. We continue in this manner until either: (i) $k = max$ or (ii) the coverage is equal to 100%. On each iteration the $genMinVULS$ procedure is called (line 15).

$$coverage = \frac{num.\ vertices\ covered\ by\ VULS}{num.\ vertices\ in\ G_{input}} \times 100 \tag{1}$$

The $genMinVULS$ procedure takes as input the current graph size k (where k is the number of edges) and the set of k-edge sub-graphs contained in the set G_k as pruned so far. The procedure returns the set of k-edge minimal VULS. On each call the procedure $genMinVULS$ loops through the input set of k-edge subgraphs and (line 23) for each sub-graph g determines whether it is a VULS or not by calling Algorithm 2 which is described in detail below. If g is a VULS it is added to the set R (line 24). We then (line 25) calculate the coverage so far, if this has reached 100% we have found the complete set of minimal VULS and we exit (line 27). Note that if coverage is equal to 100% the input set G will now be empty. Otherwise, if the coverage is not 100%, we continue processing and (line 29) remove g from the global set G. If g is not a VULS we add it to T_k (line 31), T_k is the set of k-edge sub-graphs which we will eventually be extended to form G_{k+1}, the set of $(k+1)$-edge sub-graphs, ready for the next level of processing. Eventually all g in G_k will have been processed. If, at this stage T_k is empty there will be no more sub-graphs that can be generated and the process will exit (line 35). Otherwise control will return to the *main* procedure and the set of $(k+1)$-edge sub-graphs will be generated from T_k (the set of k-edge subgraphs that have not been found to be VULS) using a right most extension technique coupled with isomorphism checking to establish which $(k+1)$-edge subgraphs are contained in G as processed so far (line 16). This part of the algorithm is not

Algorithm 1 minREVULSM

Input:
G_{input} = Training input graph
max = Max subgraph size
Output:
R = Set of minimal VULS
Global variables:
$G = G_{input}$ (Part of training input graph not covered by minimal VULS)
coverage = 0
T_k = the set of k-edge subgraphs which are not VULS

procedure $main(G_{input}, max)$
    $k = 1$
    G_k = the set of k-edge subgraphs in G
    $R = \emptyset$
    while $(k < max)$ **do**
        $R = R \cup genMinVULS(k, G_k)$
        G_{k+1} = Set of $(k+1)$-edge subgraphs in G (found by applying right most extension to each subgraph in T_k)
        $k = k + 1$
    end while
end procedure

procedure $genMinVULS(k, G_k)$
    $T_k = \emptyset$
    for all $g \in G_k$ **do**
        if $isaVULS(g, G_k) == true$ (Algorithm 2) **then**
            $R = R \cup g$
            $coverage$ = compute coverage using Equ. 1
            if $coverage == 100\%$ **then**
                exit
            end if
            $G = G - g$
        else
            $T_k = T_k \cup \{g\}$
        end if
    end for
    if $T_k == \emptyset$ **then**
        exit
    end if
    return R
end procedure

presented here because it is similar to that found in more traditional sub-graph mining algorithms (such as gSpan). The generated minimal VULS set R is then returned (line 37) back to the main process ready for the next iteration (unless the maximum value for k has been reached).

Algorithm 2 presents the pseudo code for identifying whether a given sub-graph g is a VULS or not with respect to the current set of k-edge sub-graphs G_k from which g has been removed. The algorithm returns $true$ if g is a VULS and $false$ otherwise. The process commences (line 8) by generating the potential list of vertex labels S that can be matched to g according to the content of G_k (see previous section for detail). The list S is then processed and tested. If there exists a vertex pair whose possible labelling is not unique (has more than one possible labelling that can be associated with it) g is not a VULS and the procedure returns $false$, otherwise g is a VULS and the procedure returns $true$. Thus, as the minREVULSM algorithm proceeds, the input graph G will be continuously pruned with respect to the identified VULS. As a result G can become disconnected, any disconnected sub graph of size less than the current value of k can not therefore contain any k-edge VULS. Although not shown in Algorithm 1 any disconnected sub-graphs of size less than k are discounted so as to speed up the overall process.

Algorithm 2 Identify VULS

```
1: Input:
2: g = a single k-edge subgraph (potential VULS)
3: G_k = a set of k-edge subgraphs to be compared with g
4: Output:
5: true if g is a VULS, false otherwise

6: procedure isaVULS(g, G_k)
7:     isVULS = true
8:     S = the list of vertex label pairs of edges that may be assigned to g
9:     for all vertex label pairs of each edge [L_i, L_j] ∈ S do
10:        if either size of start vertex labels |L_i| ≠ 1 or size of end vertex labels
   |L_j| ≠ 1 then
11:            isVULS = false
12:            break
13:        end if
14:    end for
15:    return isVULS
16: end procedure
```

4 The Match-Voting algorithm

Once we have identified a set of minimal VULS we can use the set of identified VULS to predict vertex labels with respect to further (unseen) graphs. The basic idea is to match subgraphs contained in the new graph with the collection of identified VULS. The matching is conducted using the Match-Voting Algorithm. The algorithm comprises one main procedure ($main$) and a sub-procedure ($matchVoteVULS$). The algorithm takes as input a collection of VULS and a

new graph G_{new} which has edge labels but no known vertex labels. The algorithm also utilises a parameter max set to the same value as that used to generate the VULS so that the algorithm does not continue to try and find matches beyond the maximum size of the VULS presented as part of the input. The output is the graph G_{new} with vertices labelling. We match each VULS from VULSmodel to each of the potential edge lists in G_{new}. Whenever a match is found we attach the labels from the VULS to the vertices associated with the identified edge list. We continue in this manner for each VULS in our input set of VULSmodel. On completion at least some of vertices in G_{input} will have vertex labels associated with them (the more the better). In some cases an edge list in G_{new} may be matched to more than one VULS in which case the associated vertices will have a number of potential labelings. In this case a voting mechanism is used to select the "best" labelling (lines 15-18). We proceed in a breadth first manner starting with one-edge VULS ($k = 1$), then two edge VULS ($k = 2$), and so on. We continue in this manner until $k = max$ is reached. On each iteration the $matchVoteVULS$ procedure is called (lines 11 to 14).

5 Experiments and Performance Study

This section describes the evaluation of the proposed VULS classification system in terms of effectiveness and efficiency. Because of the novelty of the proposed VULS concept there are no alternative algorithms that can be used for comparison purposes. However, we were able to analyse the effectiveness of the prediction using standard classification performance measures (accuracy and AUC), and compare the effectiveness of using only minimal VULS with using all VULS. The rest of this section is organised as follows. Section 5.1 presents the data sets used for the evaluation and briefly describes the application domain. Section 5.2 reports on classification performance with respect to both minimal VULS and all VULS, and in Section 5.3 we consider the "runtime" of the proposed approach. Note the the REVULSM algorithms were implemented using the JAVA programming language; all experiments were conducted using a 2.7 GHz Intel Core i5 with 4 GB 1333 MHz DDR3 memory, running OS X 10.8.1 (12B19).

5.1 Data sets

The data used for the reported experiments was taken from a real sheet metal forming application, namely the fabrication of flat topped pyramid shapes out of sheet steel. More precisely the application of Asymmetric Incremental Sheet Forming (AISF), a particular kind of sheet metal forming. A pyramid shape was chosen as it is frequently used as a benchmark shape for conducting experiments in the context of AISF [6]. The pyramid was manufactured twice so that we had one dataset to use as a training set and another to use as a test set. The required input labelled training and test graphs were extracted from the "before" and "after" grids describing the geometry of the piece to be manufactured and the resulting piece actually produced. Each grid square centre point was defined in terms of a Euclidean (X-Y-Z) coordinate scheme. Each before grid square centre point was then considered to represent a vertex and had a springback

Algorithm 3 Match-Voting Algorithm

Input:
G_{new} = Edge labelled testing input graph
max = Max subgraph size (consistent with minREVULSM in training procedure)
$VULSmodel$ = a set of VULS or minimal VULS
Output:
G_{new} with vertices labelling

procedure $main(G_{new}, max, VULSmodel)$
    $G_{new} = \{P_1, P_2, \ldots, P_i\}$ (where i is the total number of edges in G_{new})
    $k = 1$
    $VULS_k$ = the set of k-edge VULS or minimal VULS in $VULSmodel$
    while $(k < max)$ **do**
        matchVoteVULS($VULS_k$)
        $k = k + 1$
    end while
    for all $p \in G_{new}$ **do**
        $p.v_a.label$ =most frequently voted vertex label in Vote ($p.v_a$)
        $p.v_b.label$ =most frequently voted vertex label in Vote ($p.v_b$)
    end for
end procedure

procedure $matchVoteVULS(VULS_k)$
    for all $vuls \in VULS_k$ **do**
        $S \leftarrow getVertexLabels(vuls)$
        if $\exists\{P_x, \ldots, P_y\} \in G_{new}$ (where $y - x = k$) and $\{P_x.label = L_1.label, P_(x+1).label = L_2.label, \ldots, P_y.label = L_k.label\}$ (where $[L_1, L_2, \ldots, L_k] \in S$) **then**
            Vote($P_x.v_a.label = L_{v_{a1}}, P_x.v_b.label = L_{v_{b1}}, \ldots, P_y.v_a.label = L_{v_{ak}}, P_y.v_b.label = L_{v_{bk}}$)
        end if
    end for
end procedure

error value associated with it calculated by measuring the distance along the normal from the before surface at the grid centre to where the normal cut the after surface. Each vertex (except at the edges and corners) was then connected to each of its four neighbours by a set of four edges (one per neighbour) labelled with a "slope" value calculated from the absolute difference in Z value between each centre point and its neighbour. Finally the vertex and edge values were discretised, according to a set of labels L_V and L_E respectively, so that they were represented by nominal values (otherwise every edge pair was likely to be unique). A number of different grid sizes were also considered each related to a different grid size d.

Table 4. Number of VULS Comparison

max	$\|L_E\| \times \|L_V\|$	All VULS			min VULS		
		$d = 10$	$d = 12$	$d = 14$	$d = 10$	$d = 12$	$d = 14$
4	2×2	25	37	24	16	17	18
	3×2	137	135	112	41	35	25
	4×2	286	299	303	70	54	76
5	2×2	139	144	110	50	40	76
	3×2	611	689	573	53	113	37
	4×2	1466	1564	1628	112	113	84
6	2×2	665	696	526	118	46	204
	3×2	2732	3366	2886	87	277	42
	4×2	7137	7423	8392	205	173	89

Table 5. Accuracy Comparison

max	$\|L_E\| \times \|L_V\|$	All VULS			min VULS		
		$d = 10$	$d = 12$	$d = 14$	$d = 10$	$d = 12$	$d = 14$
4	2×2	29.00	9.03	43.37	38.00	26.39	83.67
	3×2	39.00	13.89	37.24	70.00	38.19	78.06
	4×2	37.00	22.92	42.35	68.00	72.22	90.82
5	2×2	30.00	11.81	51.02	70.00	65.28	88.78
	3×2	49.00	17.36	33.67	70.00	58.33	82.65
	4×2	57.00	23.61	15.31	67.00	93.75	91.33
6	2×2	24.00	15.97	47.45	70.00	72.92	92.86
	3×2	31.00	22.22	35.20	70.00	89.58	91.33
	4×2	52.00	20.83	12.76	67.00	77.78	91.33

5.2 Prediction Performance

For the prediction performance evaluation we used a number of different parameter settings for: $|L_E| \times |L_V|$, d and max. With respect to the evaluation results reported here the following were used: $|L_E| \times |L_V| = \{2 \times 2, 3 \times 2, 4 \times 2\}$, $d = \{10, 12, 14\}$ and $max = \{4, 5, 6\}$. Use of these parameters all gave a 100% coverage (in the context of the sheet metal forming application 100% coverage is desirable). The results obtained using the above parameters are presented in

Table 6. AUC Comparison

max	$\|L_E\| \times \|L_V\|$	All VULS			min VULS		
		$d = 10$	$d = 12$	$d = 14$	$d = 10$	$d = 12$	$d = 14$
4	2×2	44.90	52.88	58.35	34.92	61.87	75.09
	3×2	47.86	7.19	33.70	50.32	19.78	50.72
	4×2	44.41	21.51	31.17	49.93	37.41	49.72
5	2×2	44.58	54.32	59.88	50.32	33.81	72.56
	3×2	52.87	8.99	31.74	50.32	30.22	53.24
	4×2	53.40	12.23	19.03	48.20	48.56	50.00
6	2×2	31.18	56.47	60.58	50.32	37.77	72.13
	3×2	37.13	11.51	32.58	50.32	46.40	57.99
	4×2	44.78	10.79	20.29	48.20	40.29	50.00

Tables 5 and 6[1]. From Table 5 it can be observed that using minimal VULS produces much better accuracy results than when using all VULS. This is because when minimal VULS are used there are much fewer competing label predictions for each vertex and consequently the Match Voting algorithm is much more effective. Best results were obtained using $max = 6$ and $d = 14$. Inspection of Table 6 indicates that in terms of the Area Under the receiver operating Curve (AUC) $|L_E| \times |L_V| = 2 \times 2$ and $d = 14$ produced the best results. Overall the results indicate that the proposed VULS mechanism can be successfully applied in the context of sheet metal forming for the purpose of error (springback) prediction.

5.3 Runtime Analysis

The runtime results obtained, using the same parameters as above, are presented in Table 7. From the table it can be observed (as anticipated) that the runtime for mining minimal VULS is significantly less than when mining all VULS (gains with respect to memory usage are all realised). Thus we can conclude that finding only minimal VULS is more efficient than when finding all VULS while at the same time resulting in an improved accuracy.

6 Conclusions and further Study

In this paper we have introduced the concept of minimal VULS mining and presented the minREVULSM algorithm for identifying all minimal VULSM in a given graph or network. One application for the VULS concept is classification and we have thus also presented the Match-Voting algorithm for applying the VULS concept to the vertex label classification problem. The work has been illustrated, and evaluated, using a sheet metal forming application (AISF) where we wish to predict the "springback" error as a result of metal forming. The results produced indicate that minimal VULS mining is both efficient and effective (at

[1] When $d = 10$ is used the number of vertices is $d \times d = 10 \times 10 = 100$, so the percentage coverage values are whole numbers.

Table 7. Runtime Comparison (seconds)

max	$\|L_E\| \times \|L_V\|$	All VULS			min VULS		
		$d = 10$	$d = 12$	$d = 14$	$d = 10$	$d = 12$	$d = 14$
4	2×2	0.34	0.52	0.50	0.27	0.32	0.37
	3×2	0.48	0.66	0.74	0.28	0.36	0.41
	4×2	0.59	0.82	0.87	0.40	0.38	0.41
5	2×2	0.7	1.07	0.93	0.35	0.45	0.55
	3×2	1.35	1.53	1.41	0.39	0.75	0.47
	4×2	1.50	1.69	1.64	0.42	0.60	0.46
	4 v 3	1.66	1.69	2.38	0.76	0.99	1.04
6	2×2	1.62	1.81	1.54	0.53	0.63	0.95
	3×2	1.91	2.78	3.24	0.46	1.15	0.56
	4×2	4.03	4.88	5.84	0.42	0.80	0.68

least in the context of the sheet metal forming application used for the evaluation). For future work the authors intend to investigate more sophisticated ways of conducting VULS based classification and to consider alternative application domains such as social network mining.

Acknowledgments. The research leading to the results presented in this paper has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 266208.

References

1. Cafuta, G., Mole, N., tok, B.: An enhanced displacement adjustment method: Springback and thinning compensation. Materials and Design 40, 476–487 (2012)
2. Firat, M., Kaftanoglu, B., Eser, O.: Sheet metal forming analyses with an emphasis on the springback deformation. Journal of Materials Processing Technology 196(1-3), 135–148 (2008)
3. Jeswiet, J., Micari, F., Hirt, G., Bramley, A., andJ. Allwood, J.D.: Asymmetric single point incremental forming of sheet metal. CIRP Annals Manufacturing Technology 54(2), 88–114 (2005)
4. Liu, W., Liang, Z., Huang, T., Chen, Y., Lian, J.: Process optimal ccontrol of sheet metal forming springback based on evolutionary strategy. In: In Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress. pp. 7940–7945 (June 2008)
5. Nasrollahi, V., Arezoo, B.: Prediction of springback in sheet metal components with holes on the bending area, using experiments, finite element and neural networks. Materials and Design 36, 331–336 (2012)
6. Salhi, S., Coenen, F., Dixon, C., Khan, M.: Identification of correlations between 3d surfaces using data mining techniques: Predicting springback in sheet metal forming. In: Proceedings Proc. AI 2012. Springer, Cambridge. pp. 391–404. (2012)
7. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: Proceedings of the 2002 International Conference on Data Mining. p. 721 (2002)