Vertex Unique Labelled Subgraph Mining

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

Abstract With the successful development of efficient algorithms for Frequent Subgraph Mining (FSM), this paper extends the scope of subgraph mining by proposing Vertex Unique labelled Subgraph Mining (VULSM). VULSM has a focus on the local properties of a graph and does not require external parameters such as the support threshold used in frequent pattern mining. There are many applications where the mining of VULS is significant, the application considered in this paper is error prediction with respect to sheet metal forming. More specifically this paper presents a formalism for VULSM and an algorithm, the Right-most Extension VULS Mining (REVULSM) algorithm, which identifies all VULS in a given graph. The performance of REVULSM is evaluated using a real world sheet metal forming application. The experimental results demonstrate that all VULS (Vertex Unique Labelled Subgraphs) can be effectively identified.

1 Introduction

A novel research theme in the context of graph mining [7, 15, 8, 16], Vertex Unique labelled Subgraph Mining (VULSM), is proposed in this paper. Given a particular sub-graph g in a single input graph G; this subgraph will have a specific structure, and edge and vertex labelling associated with it. If we consider only the structure and edge labelling there may be a number of different compatible vertex labellings with respect to G. A Vertex Unique Labelled Subgraph (VULS) is a subgraph with a specific structure and edge labelling that has a unique vertex labelling associated with it. This paper proposes the

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

Right-most Extension Vertex Unique Labelled Subgraph Mining Algorithm (REVULSM) to identify all VULS. REVULSM generates subgraphs (potential VULS candidates) using Right Most Extension [3], in a DFS manner, as first proposed in the context of gSpan [14]; and then identifies all VULS using a level-wise approach (first proposed by Agrawal and Srikant in the context of frequent item set mining [1, 2, 9]). VULSM is applicable to various types of graph; however, in this paper we focus on undirected graphs.

VULSM has relevance with respect to a number of domains. The application domain used to illustrate the work described in this paper is error prediction in sheet metal forming. More specifically error prediction in Asymmetric Incremental Sheet Forming (AISF) [4, 6, 10, 12, 13]. In this scenario the piece to be manufactured is represented as a grid, each grid centre point is defined by a Euclidean (X-Y-Z) coordinate scheme. The grid can then be conceptualised as a graph (lattice) such that each vertex represents a grid point. Each vertex (except at the edges and corners) can then be connected to its four neighbours by a sequence of edges, which in turn can be labelled with "slope" values. An issue with sheet metal forming processes, such as AISF, is that distortions 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 proposed graph representation captures such geometries in terms of sub-graphs, particular sub-graphs are associated with particular local geometries (and by extension distortion/error patterns). Given before and after shapes we can create a training set by deriving the error associated with each vertex in the grid. This training data, in turn, can then be used to train a predictor or classifier of some sort. There are various ways that such a classifier can be generated; but one mechanism is to apply VULSM, as proposed in this paper, to identify sub-graphs that have unique error patterns associated with them that can then be used for error prediction purposes (some form of mitigating error correction can then be formulated).

A simple example grid and corresponding graph are given in Figure 1. The grid (lefthand side of Figure 1) comprises six grid squares. Each grid centre is defined by a X-Y-Z coordinate tuple. Each grid centre point is associated with a vertex within the graph (right-hand side of figure 1). The edges, as noted above, are labelled with "slope" values, the difference in the Z coordinate values associated with the two end vertices. Each vertex will be labelled with an error values (e_1 to e_3 in the figure) describing the expected distortion at that vertex as obtained from a "training set" (derived from "before and after" grid data). Identified VULS will describe local geometries each with a particular associated error pattern. This knowledge can then be used to predict errors in "unseen" grids so that some form of mitigating error correction can be applied.

The rest of this paper is organised as follows. In Section 2, we define the basic concepts of VULS together with an illustrative example. The REVULSM algorithm is then described in detail in Section 3. An experimental analysis

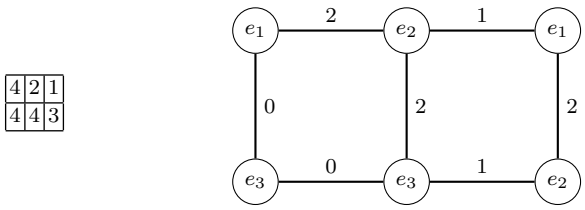

Fig. 1: Grid representation (left) with corresponding graph/lattice (right) featuring "slope" labels on edges

of the approach is presented in Section 4, and Section 5 summarises the work and the main findings, and presents some conclusions.

2 The problem formulation

This section presents a formal definition of the concept of a VULS. Assume a connected labelled graph G comprised of a set of n vertices V, such that $V = \{v_1, v_2, \ldots, v_n\}$; and a set of m edges E, 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 $L_E = \{l_{e_1}, l_{e_2}, \ldots, l_{e_q}\}$. A graph G can thus 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. For undirected graphs, the edge $\langle v_a, v_b \rangle$ is equivalent to $\langle v_b, v_a \rangle$ (in this paper we assume undirected 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 notaion $P_i.label$ ($P_i.label \in L_E$). We can use this notation with respect to any subgraph G_{sub} of G ($G_{sub} \subseteq G$).

For training purposes the graphs of interest are required to be labelled. However, we can also conceive of edge only labelled graphs and subgraphs. Given some edge only labelled subgraph ($G_{subedgelab}$) of some fully labelled graph G ($G_{subedgelab} \subseteq G$) comprised of k edges, there may be many different vertex labelings that can be associated with such a subgraph according to the nature of G. We thus define a function, $getVertexLabels$, that returns the potential list of labels S that can be assigned to the vertices in $G_{subedgelab}$ according to G:

$$getVertexLabels(G_{subedgelab}) \rightarrow S$$

where $G_{subedgelab} = \{P_1, P_2, \ldots, P_k\}$ 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). Thus each element in S comprises two sub-sets of labels associated respectively with the start and end vertex for each edge in $G_{subedgelab}$; there is a one to one correspondence between each element (pair of label sets) in S with each element in $G_{subedgelab}$, hence they are both of the same size k (recall that k is the number of edges). We also assume that some canonical labelling is adopted.

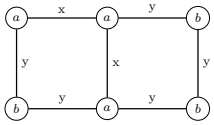

Fig. 2: Undirected example lattice

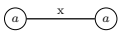

Fig. 3: One edge VULS generated from lattice in Figure 2

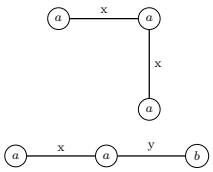

Fig. 4: Two edge VULS generated from lattice in Figure 2

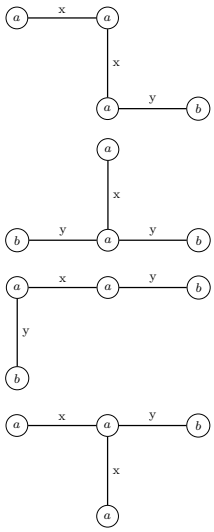

Fig. 5: Three edge VULS generated from lattice in Figure 2

According to the above, the formal definition of the concept of a VULS is as follows. Given: (i) a k-edge edge labelled subgraph $G_{subedgelab} = \{P_1, P_2, \ldots, P_k\}$ ($G_{subedgelab} \subseteq G$), (ii) a list of labels that may be associated with the vertices in $G_{subedgelab}$, $S = [[L_{P_1.v_a}, L_{P_1.v_b}], [L_{P_2.v_a}, L_{P_2.v_b}], \ldots, [L_{P_k.v_a}, L_{P_k.v_b}]]$, and (iii) the proviso that $G_{subedgelab}$ is connected. If $\forall [L_i, L_j] \in S, |L_i| = 1, |L_j| = 1$ then $G_{subedgelab}$ is a k-edge VULS with respect to G.

So as to provide for a full and complete comprehension of the concept of VULS an example lattice is presented in Figure 2. The VULS that exist in this lattice are itemized in Figures 3 to 5. If we consider one-edge subgraphs first, there are two possibilities: (i) graphs featuring edge x, and (ii) graphs featuring edge y. The list of possible vertices S associated with the first, obtained using the $getVertexLabels$ function, is $[[\{a\}, \{a\}]]$, while the list associated with the second is $[[\{a, b\}, \{b\}]]$ (this can be verified by inspection of Figure 2). Considering edge x first, $\forall [L_i, L_j] \in S$, $|L_i| = 1$ and $|L_j| = 1$,

so this is a VULS; however, considering edge y, $\forall[L_i, L_j] \in S$, $|L_i| \neq 1$ and $|L_j| = 1$ hence this is not a VULS. We now consider the two edge subgraphs by extending the one edge subgraphs. We can not enumerate all two edge subgraphs here due to space limitations but the two edge VULS are shown in Figure 4. Taking the first VULS in Figure 4, $\{P_1, P_2\}$, as an example, here $P_1.v_a = a$, $P_1.v_b = a$, $P_2.v_a = a$ and $P_2.v_b = a$, furthermore the edge labels associated with P_1 and P_2 are $P_1.label = x$ and $P_2.label = x$ respectively. In this case $S = [[a, a], [a, a]]$ thus $\forall[L_i, L_j] \in S, |L_i| = 1, |L_j| = 1$ therefore this is a *two*-edge VULS with respect to G.

Algorithm 1 REVULSM

```
1: Input:
2: G_input = Input graph
3: Max=Max subgraph size
4: Output:
5: R = Set of VULS
6: Global variables:
7: G = set of subgraphs (VULS candidates) in G_input

8: procedure REVULSM(G_input, Max)
9:     R = ∅
10:    G = ∅
11:    G = Subgraph_Mining(G_input) (Algorithm 2)
12:    k=1
13:    while (k < Max) do
14:        for all G_sub ∈ G_k (where G_k is the set of k-edge subgraphs in G) do
15:            if IdentifyVULS(G_sub, G_k) == true (Algorithm 3) then
16:                R = R ∪ G_sub
17:            end if
18:        end for
19:        k++
20:    end while
21:    Return R
22: end procedure
```

3 The REVULSM algorithm

The proposed REVULSM algorithm is defined in this section. The algorithm is founded on the VULS properties presented above and makes use of a graph representation technique "borrowed" from gSpan.

The pseudo code for REVULSM is presented in Algorithms 1, 2 and 3. Algorithm 1 presents the high level control structure, while Algorithm 2 presents the detail for generating all subgraphs (VULS candidates), and Algorithm 3 the detail for determining whether a specific sub-graph is a VULS or not.

Algorithm 2 Subgraph_Mining

```
1: Input:
2: G_input = Input graph
3: Output:
4: G = set of subgraphs in G_input
5: Global variables:
6: G_temp=set of subgraphs generated so far

7: procedure Subgraph_Mining(G_input)
8:     G = ∅
9:     G_temp = ∅
10:    G_1=the set of one-edge subgraphs in G_input
11:    sort G_1 in DFS lexicographic order
12:    for each edge e ∈ G_1 do
13:        G_temp = Subgraph(e,1,Max)
14:        G = G ∪ G_temp
15:        G_input = (G_input − e) (remove e from G_input)
16:    end for
17:    Return G
18: end procedure

19: procedure Subgraph(e, size, Max)
20:    if size > Max then
21:        return ∅
22:    end if
23:    generate all e's potential extension subgraphs c in G_input with one edge growth by
       right most extension
24:    for each c do
25:        if c is minimal DFSCode then
26:            G_temp = G_temp ∪ c
27:            Subgraph(G_temp,size+1,Max)
28:        end if
29:    end for
30:    Return G_temp
31: end procedure
```

Considering Algorithm 1 first, the algorithm commences with an input graph G_{input} and a parameter Max that defines the maximum size of the VULS. If we do not limit the size of the searched-for VULSs the entire input graph may ultimately be identified as a VULS which, in the context of the sheet metal forming target application, will not be very useful. The output is a set of VULS R (the set R may include overlaps). Note that all graphs are encoded using a Depth First Search (DFS) lexicographical ordering (as used in gSpan [14]). The global variable G (line 7 in Algorithm 1) is the set of all subgraphs in G_{input}. At the start of the REVULSM procedure, the sets G and R will be empty. We proceed in a depth first manner to generate all subgraphs (VULS candidates) G by calling algorithm 2 (line 11). Then we identify VULS from all subgraphs G starting from one-edge subgraphs ($k = 1$), then two edge sub-graphs ($k = 2$), and so on. We continue in this manner until $k = Max$

Algorithm 3 IdentifyVULS

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

(line 13-20). On each iteration algorithm 3 is called (line 15) to determine whether G_{sub} is a VULS or not with respect to the k-edge subgraphs G_k. If it is VULS, it will be added to the set R.

Algorithm 2 comprises two procedures. The first, $Subgraph\_Mining(G_{input})$, is similar to that found in gSpan. We are iteratively finding all subgraphs, up to a size of Max. We commence (line 10-11) by sorting all the one-edge subgraphs, contained in input graph G_{input}, into DFS lexicographic order and storing them in G_1. Then (lines 12-16), for each one edge subgraph e in G_1 we call the $Subgraph$ procedure (line 13), which finds all super graphs for each one edge graph e up to size Max in a DFS manner, and stores the result in G_{temp}; which is then added to G (line 14). Finally, we remove e from G_{input} (line 15) to avoid generating again any duplicate subgraphs containing e.

The $Subgraph(e, size, Max)$ procedure generates all the super graphs of the given one edge subgraph e by growing e by adding edges using the right most extension principle. For each potential subgraph c, if c is described by a minimal DFSCode (line 25) the process is repeat (in a DFS style) so as to generate all the super graphs of e (line 27). The process continues in this recursive manner until the number of edges in the super graphs to be generated ($size$) is greater than Max, or no more graphs can be generated.

Algorithm 3 presents the pseudo code for identifying whether a given subgraph 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 $true$.

4 Experiments and Performance Study

This section describes the performance study that was conducted to analyse the generation and application of the concept of VULS. The reported experiments were all applied to a real application of sheet metal forming, more specifically the application of AISF [11, 5] to the fabrication of flat-topped pyramid shapes manufactured out of sheet steel. This shape was chosen as it is frequently used as a benchmark shape for conducting experiments in the context of AISF (although not necessarily with respect to error prediction). Nine graphs were generated from this data using three different grid sizes and different numbers of edge and vertex labels; in addition a range of values were used for the Max parameter. The rest of this sub-section is organised as follows. The performance measures used with respect to the evaluation are itemised in Section 4.1, more detail concerning the data sets used for the evaluation is given in Section 4.2, and the obtained results are presented and discussed in Section 4.3.

4.1 Experimental performance measurement

Four performance measures were used to analyse the effectiveness of the proposed REVULSM : (i) run time (seconds), (ii) number of VULS identified, (iii) discovery rate and (iv) coverage rate. The last two merit some further explanation. The discovery rate is the ratio of VULS discovered with respect to the total number of subgraphs of size less than Max (Equation 1). The coverage rate is the ratio of the number of vertices covered by the detected VULS compared to the total number of vertices in the input graph (Equation 2); with respect to the sheet steel forming example application high coverage rates are desirable.

$$discovery\ rate\ (\%) = \frac{number\ of\ VULS}{number\ of\ subgraphs} \tag{1}$$

$$coverage\ rate\ (\%) = \frac{number\ of\ vertices\ covered\ by\ VULS}{number\ of\ vertices\ in\ input\ graph} \tag{2}$$

4.2 Data sets

The data sets used for the evaluation consisted of before and after "coordinate clouds"; the first generated by a CAD system, the second using an optical measuring technique. These were transformed into grid representations, referenced using a X-Y-Z coordinate system, such that the before grid could be correlated with the after grid and error measurements obtained. A fragment of the before grid data, with associated error values (mm), is presented in Table 1. The before grid data was then translated into a graph such that each grid square was represented by a vertex linked to each of its neighbouring squares by an edge. Each vertex was labelled with an error value while the edges were labelled according to the difference in Z of the two end vertices (the "slope" connecting them). Furthermore, the vertex and edge labels were discretised so that they were represented by nominal values (otherwise every edge pair was likely to be unique). This was then the input into the REVULSM algorithm.

Table 1: Format of raw input data

x	y	z	error
0.000	0.000	0.000	0.118
1.000	0.000	0.000	0.469
2.000	0.000	0.000	0.469
3.000	0.000	0.000	0.472
0.000	1.000	0.000	0.471
1.000	1.000	-1.402	0.088
2.000	1.000	-4.502	1.308
3.000	1.000	-4.676	1.907
...	...	...	...

As noted above, from the raw data, different sized grid representations, and consequently graph representations, may be generated. For experimental purposes three grid formats were used 6×6 , 10×10 and 21×21. We can also assign different numbers of edge labels to the vertices and edges, for the evaluation reported here values of two and three were used in three different combinations. In total nine different graph data sets were generated, numbered AISF1 to AISF9. AISF1 to AISF3 were generated using a 6×6 grid, while AISF4 to AISF6 were generated using a 10×10 grid, and AISF7 to AISF9 were generated using a 21×21 grid. Some statistics concerning these graph sets are presented in Table 2.

Table 2: Summary of AISF graph sets

graph set	# vertices	# edge labels	# vertex labels	graph set	# vertices	# edge labels	# vertex labels
AISF1	36	3	2	AISF6	100	2	3
AISF2	36	2	2	AISF7	441	3	2
AISF3	36	2	3	AISF8	441	2	2
AISF4	100	3	2	AISF9	441	2	3
AISF5	100	2	2				

4.3 Experimental results and analysis

For experimental purposes REVULSM was implemented in 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). The results obtained are presented in Figures 6 to 17. Figures 6 to 8 give the run time comparisons with respect to the nine graph sets. Figures 9 to 11 give the number of discovered VULS in each case. Figures 12 to 14 present a comparison of the recorded discovery rates with respect to the nine graph sets considered. Finally Figures 15 to 17 give a comparison of the coverage rates.

From Figures 6 to 8 it can be seen, as might be expected, that as the value of the Max parameter increases the run time also increases because more subgraphs and hence more VULS are generated. The same observation is true with respect to the size of the graph; the more vertices the greater the required runtime.

From Figures 9 (6×6 grid), 10 (10×10 grid) and 11 (21×21 grid) it can be observed that as Max increases the number of VULS will also increase, again this is as might be expected. Comparing AISF1, 4 and 7 with AISF2, 5 and 8 respectively, it can be seen that as the number of edge labels increases while the number of vertex labels is kept constant the number of VULS also increases (AISF1 and AISF2 have the same number of vertex labels; as do AISF4 and AISF5, and AISF7 and AISF8). This is because the likelihood of VULS existing increases as the input graph becomes more diverse. However, comparing AISF2, 5 and 8 with AISF3, 6 and 9 respectively it can be seen that as the number of vertex labels increases, while the number of edge labels is kept constant, the number of VULS generated will decrease (AISF2 and AISF3 have the same number of edge labels; as do AISF5 and AISF6, and AISF8 and AISF9); because, given a high number of vertex labels, the likelihood of VULS existing decreases.

Figure 12 (6×6 grid), 13 (10×10 grid) and 14 (21×21 grid) show the recorded discovery rate values. Comparing AISF1, 4 and 7 with AISF2, 5 and 8 respectively; when the number of edge labels increases, while the number of vertex labels is kept constant, the discovery rate increases. This is

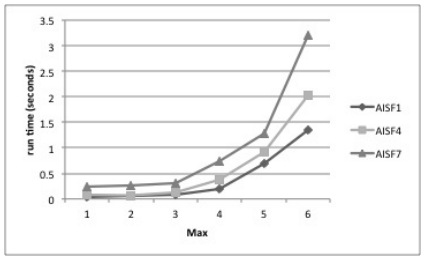

Fig. 6: Run time comparison using 3 edge and 2 vertex labels (AISF1, AISF4 and AISF7)

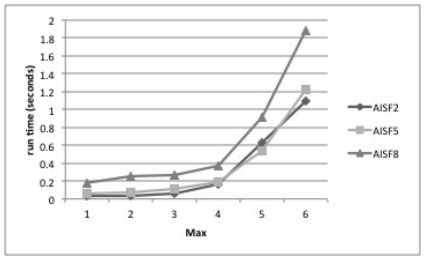

Fig. 7: Run time comparison using 2 edge and 2 vertex labels (AISF2, AISF5 and AISF8)

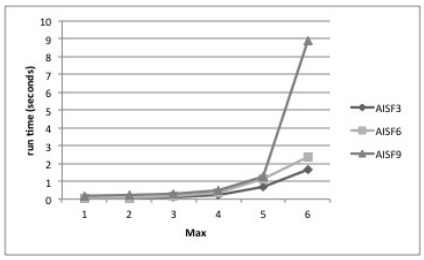

Fig. 8: Run time comparison using 2 edge and 3 vertex labels (AISF3, AISF6 and AISF9)

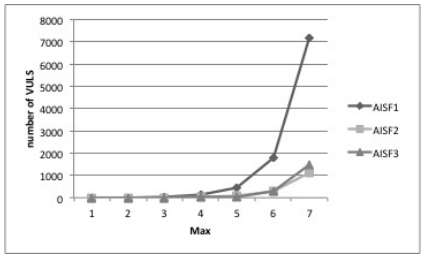

Fig. 9: Comparison of number of VULS generated (AISF1, AISF2 and AISF3)

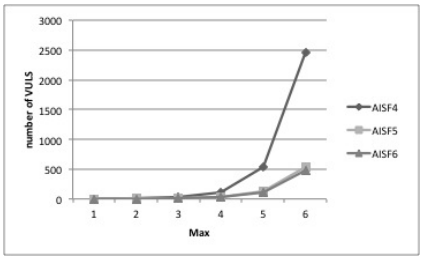

Fig. 10: Comparison of number of VULS generated (AISF4, AISF5 and AISF6)

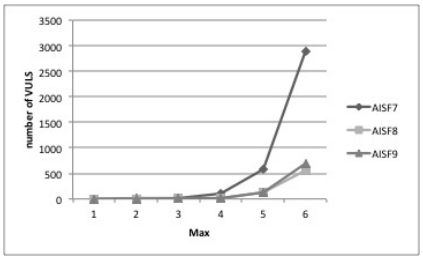

Fig. 11: Comparison of number of VULS generated (AISF7, AISF8 and AISF9)

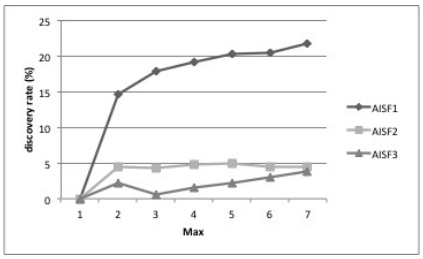

Fig. 12: Comparison of discovery rate (AISF1, AISF2 and AISF3)

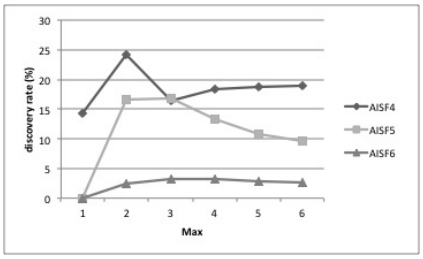

Fig. 13: Comparison of discovery rate (AISF4, AISF5 and AISF6)

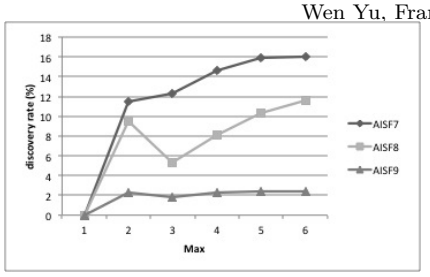

Fig. 14: Comparison of discovery rate (AISF7, AISF8 and AISF9)

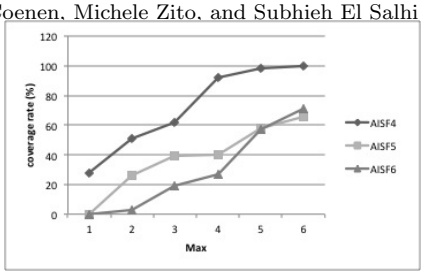

Fig. 16: Comparison of coverage rate (AISF4, AISF5 and AISF6)

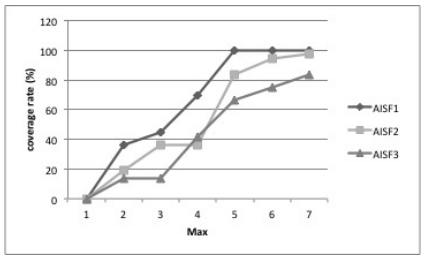

Fig. 15: Comparison of coverage rate (AISF1, AISF2 and AISF3)

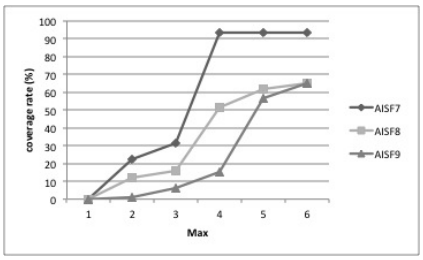

Fig. 17: Comparison of coverage rate (AISF7, AISF8 and AISF9)

because regardless of the number of edge labels a graph has (all other elements being kept constant) the number of subgraphs contained in the graph will not change, while (as indicated by the experiments reported in Figures 9 , 10 and 11) the number of identified VULS increases as the number of edge labels increases. Conversely, comparing AISF2, 5 and 8 with AISF3, 6 and 9 respectively, when the number of vertex labels increases while the number of edge labels is kept constant, the discovery rate will decrease because (as already noted) the number of VULS generated decreases as the number of vertex labels increases. It can also be noted that as the Max value increases, the discovery rate does not always increase, as shown in the case of AISF4, 5 and 6. This is because as the Max value increases, the number of VULS goes up as does the number of subgraphs, but they may not both increase at the same rate.

Figures 15, 16 and 17 show the coverage rate. From the figures it can be observed that as Max increases the coverage rate also increases. This is to be expected, however it is interesting to note that the coverage rate in some cases reaches 100% (when $Max = 6$ with respect to AISF1, ASF2 and AISF4). One hundred percent coverage is desirable in the context of the sheet metal forming application so that unique patterns associated with particular error distributions (vertex labels) can be identified for all geometries. Comparing AISF1, 4 and 7 with AISF2, 5 and 8 respectively, the more edge labels a graph has the more VULS will be generated (see above); as a result more vertices will be covered by VULS and hence the coverage rate will go up. On the other hand, comparing AISF2, 5 and 8 with AISF3, 6 and 9 respectively,

the more vertex labels a graph has the less VULS will be generated (see above); as a result fewer vertices will be covered by VULS vertices and hence the coverage rate will go down.

5 Conclusions and further Study

In this paper we have proposed the mining of VULS and presented the REVULSM algorithm. The reported experimental results demonstrated that the VULS idea is sound and that REVULSM can effectively identify VULS in real data. Having established a "proof on concept" there are many interesting research problems related to VULSM that can now be pursued. For instance, currently, when the Max parameter is high REVULSM will run out of memory, although for the purpose of error prediction in sheet metal forming it can be argued that there is no requirement for larger VULS, it may be of interest to investigate methods whereby the efficiency of REVULSM can be improved. Finally, at present, REVULSM finds all VULS up to a predefined size, it is conjectured that efficiency gains can be made if only minimal VULS are found.

6 Acknowledgements

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. Agrawal, R., Srikant, R.: fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases(VLDB '94), pp. 487–499 (1994)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proceedings of the Eleventh International Conference on Data Engineering(ICDE '95), pp. 3–14 (1995)
3. Asai, T., Abe, K., Kawasoe, S., Sakamoto, H., Arikawa, S.: Efficient substructure discovery from large semi-structured data. In: In Proc.2002 SIAM Int.Conf. Data Mining, pp. 158–174 (2002)
4. Cafuta, G., Mole, N., tok, B.: An enhanced displacement adjustment method: Springback and thinning compensation. Materials and Design **40**, 476–487 (2012)
5. El-Salhi, S., Coenen, F., Dixon, C., Khan, M.S.: Identification of correlations between 3d surfaces using data mining techniques: Predicting springback in sheet metal forming. In: Research and Development in Intelligent Systems XXIX, pp. 391–404 (2012)

6. 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)
7. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: Current status and future directions. Data Mining and Knowledge Discovery **15**(1), 55–86 (2007)
8. Huan, J., Wang, W., Prins, J., Yang, J.: SPIN: mining maximal frequent subgraphs from graph databases. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 581–586 (2004)
9. Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. In: In Principles of Data Mining and Knowledge Discovery, pp. 13–23 (2000)
10. 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)
11. Khan, M.S., Coenen, F., Dixon, C., El-Salhi, S.: Finding correlations between 3-d surfaces: A study in asymmetric incremental sheet forming. Machine Learning and Data Mining in Pattern Recognition Lecture Notes in Computer Science **7376**, 366–379 (2012)
12. 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)
13. 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)
14. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: Proceedings of the 2002 International Conference on Data Mining, pp. 721–724 (2002)
15. Yan, X., Han, J.: Close Graph: mining closed frequent graph patterns. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 286–295 (2003)
16. Zhu, F., Yan, X., Han, J., Yu, P.S.: gPrune: a constraint pushing framework for graph pattern mining. In: Proceedings of 2007 Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'07), pp. 388–400 (2007)