Minimal Vertex Unique Labelled Subgraph Mining

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

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

Abstract. This paper introduces the concept of Vertex Unique Labelled Subgraph Mining (VULSM), a specialised form of subgraph mining. A VULS is a subgraph defined by a set of edge labels that has a unique vertex labelling associated with it. A minimal VULS is then a VULS which is not a supergraph of any other VULS. The application considered in this paper, for evaluation purposes, is error prediction with respect to sheet metal forming. The minimum BFS Right-most Extension Unique Subgraph Mining (Min-BFS-REUSM) algorithm is introduced for identifying minimal VULS using a Breadth First Search(BFS) strategy.

Keywords: Data mining, Graph mining

1 Introduction

This paper introduces the concept of Vertex Unique Labelled Subgraph Mining (VULSM), a form of graph mining. Given a subgraph g in some input graph G, if we consider only the structure and edge labelling there may be a number of different compatible vertex labelings with respect to G. A Vertex Unique Labelled Subgraph (VULS) is then a subgraph with a specific structure and edge labelling that has a unique vertex labelling associated with it. A minimal VULS is a VULS that does not contain any subgraphs that are also VULSs. This paper is directed at finding all minimal VULS in a single input graph. To this end the Minimal Breadth First Search Right-most Extension Unique Subgraph Mining (Min-BFS-REUSM) Algorithm is proposed. The distinction between VULSM and more traditional forms of subgraph mining [4, 10, 5, 12] is that we are not interested in frequently occurring subgraphs but VULS. Broadly the proposed algorithm operates using a level-by-level approach. On each iteration a set of k-edge subgraphs that exist in G are identified (where k is also the iteration number). Any VULS identified in this set of subgraphs are stored and "removed" from G (thus G gets smaller and smaller). The process continues until G is empty or some user defined maximum size of VULS has been reached. subgraph generation is conducted using Right Most Extension [1] as popularised in the context of the gSpan transaction graph mining algorithm [11]. VULSM may be applied to various types of graph; in this paper we focus on undirected graphs.

The application domain used to illustrate the work is error prediction in sheet metal forming. More specifically error prediction in Asymmetric Incremental Sheet Forming (AISF) [2, 3, 6–8]. 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 idea is that the geometry of the piece to be manufactured can be represented as a grid, each grid centre point being defined using a X, Y, Z coordinate scheme. The entire grid can then be conceptualised as a graph such that each vertex represents a grid point and each vertex (except at the edges and corners) is connected to its four neighbours by a sequence of edges, which in turn can be labelled with "slope" values. Given an appropriate training set, each vertex can then be labelled with an error (distortion) value.

An example grid and corresponding graph are given in Figure 1. The grid comprises six grid squares. Each grid centre is defined by a X, Y, Z coordinate tuple (the number in each grid is Z value). Each grid centre point is associated with a vertex within the graph. The difference in Z value between each two neighbours in the grid is the "slope". The edges are labelled with these "slope" values. Each vertex will be labelled with an error values (e_1 to e_3 in the figure) describing the distortion experienced 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.

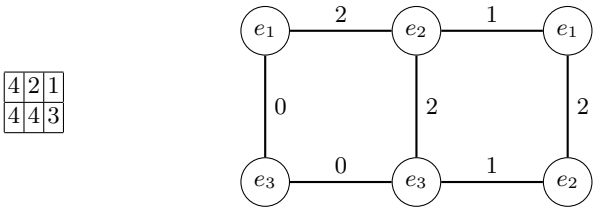

Fig. 1: Grid representation (left) and corresponding graph/lattice (right)

2 Formalism

This section presents a formal definition of the concept of a minimal VULS. A labelled graph G comprises 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 $L_E = \{l_{e_1}, l_{e_2}, \ldots, l_{e_q}\}$.

Alternatively we can think of a graph G as consisting of a set of k one-edge subgraphs: $G = \{P_1, P_2, \ldots, P_k\}$, where P_i is 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 subgraphs, we refer to 1-edge, 2-edge and k-edge subgraphs. For undirected graphs, the edge $\langle v_a, v_b \rangle$ is equivalent to $\langle v_b, v_a \rangle$. 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. We indicate the sets of labels which might be associated with $P_i.v_a$ and $P_i.v_b$ using the notation $P_i.v_a.label$ and $P_i.v_b.label$ ($P_i.v_a.label$, $P_i.v_b.label \in L_V$). We indicate the edge label associated with P_i using the notaion $P_i.label$ ($P_i.label \in L_E$). We also assume that G is connected and labelled.

We use the same notation with respect to any subgraph G_{sub} of G ($G_{sub} \subseteq G$). 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 this subgraph. 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 $S = [[L_{v_{a1}}, L_{v_{b1}}], [L_{v_{a2}}, L_{v_{b2}}], \ldots, [L_{v_{ak}}, L_{v_{bk}}]]$ (where L_{v_i} is the set of labels associate with vertex v_i and $L_{v_i} \subseteq L_v$). Note that each element in S comprises two sub-sets of labels associated respectively with the start and end vertex for each edge in $G_{subedgelab}$, and that 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).

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 $G_{subedgelab}$, $S = [[L_{v_{a1}}, L_{v_{b1}}], [L_{v_{a2}}, L_{v_{b2}}], \ldots, [L_{v_{ak}}, L_{v_{bk}}]]$. 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. A VULS ϕ_i is minimal if there is no subgraph of ϕ_i that is also a VULS.

3 The Min-BFS-REUSM Algorithm

Min-BFS-REUSM algorithum is presented in this section. Recall that REUSM stands for Right-most Extension Unique Subgraph Mining. Right-most extension is the adopted iterative subgraph generation strategy. We use the prefix "Min" to indicate that this variation is for identifying all minimal VULS and "BFS" to indicate that it features a Breadth-First Search strategy.

The pseudo code for the Min-BFS-REUSM algorithum is presented in Algorithms 1 and 2. Algorithm 1 presents the high level control structure while Algorithm 2 the detail of 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 very 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 used in gSpan [11]). The global variable G (line 7 in Algorithm 1) is the part of G_{input} not covered by any of the

Algorithm 1 Min-BFS-REUSM

Input:
G_{input} = Input graph
max = Max subgraph size
Output:
R = Set of minimal VULS
Global variables:
$G = G_{input}$ (Part of 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

identified minimal VULS so far, meanwhile, 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) (with respect to the sheet steel forming example application used as a focus for the work described in this paper, see Section 4, high coverage is desirable). The global variable T_k (line 9) is the set of k-edge non-VULS which will be extended further during the procedure to form $(k+1)$-edge candidate VULS.

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

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 subgraphs ($k = 2$), 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).

The $genMinVULS$ procedure takes as input the current graph size k (where k is the number of edges) and the set of k-edge subgraphs contained in the set G as pruned so far. The procedure returns the set of k-edge VULS. On each call the procedure $genMinVULS$ loops through the input set of k-edge subgraphs and (line 23) for each subgraph 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, as pruned so far, will 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 subgraphs which we will eventually be extended to form G_{k+1}, the set of $(k+1)$-edge subgraphs, 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 subgraphs 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 subgraphs 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 presented here because it is similar to that found in traditional subgraph mining algorithms, for example gSpan [11]. 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 subgraph g is a VULS or not with respect to the current set of k-edge subgraphs 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 poten-

tial 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 $true$.

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 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
```

Thus, as the process proceeds, the input graph G will be continuously pruned with respect to identified VULS. As a result G can become disconnected, any disconnected sub graph of size less than the current value of k cannot therefore contain any k-edge VULS. Although not shown in Algorithm 1 any disconnected subgraphs of size less than k can be discounted therefore speeding up the overall process.

4 Experiments and Performance Study

This section describes the evaluation of the proposed Min-BFS-REUSM algorithm. For experimental purposes the algorithm was implemented using the JAVA programming language; experiments were using a 2.7 GHz Intel Core i5 with 4 GB 1333 MHz DDR3 memory, running OS X 10.8.1 (12B19). The reported experiments were all conducted using real data taken from an AISF sheet metal forming application described in the introduction to this paper, more specifically the fabrication of flat topped pyramid shapes made 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 [9].

4.1 Experimental performance measurement

Three performance measures were used to analyse the effectiveness of the proposed Min-BFS-REUSM algorithm : (i) run time (seconds), (ii) total number of VULS identified, (iii) coverage (%). With respect to the sheet steel forming example application high coverage values were desirable.

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 and the second obtained using optical measuring techniques after application of an AISF process.As noted in the introduction, each vertex was labelled with an error value while the edges were labelled according to the absolute difference in z of the two end vertices (the "slope"). 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). In total ten data sets (graphs) were generated, numbered AISF1 to AISF10, using three different grid sizes (6×6, 10×10 and 21×21 which correspond to 36, 100, 441 as number of vertices translated to resulting graph respectively), and different numbers of edge and vertex labels (from 2 to 4 and 2 to 3 respectively). Some statistics concerning the data sets are presented in Table 1. Thus the graph edge labels describe the geometry of the shape to be manufactured while each vertex label describes the error occurring at that location between the desired shape and the actual shape produced. Any discovered VULS will then describe a particular geometry with a particular error pattern associated with it.

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	AISF10	441	4	2

Table 1: Summary of AISF graph sets

4.3 Run time analysis

The results obtained for the run-time experiments are presented in Table 2 with respect to a range of *max* values. From the table the following can be noted. Firstly, as might be expected, the run time increases as the value of *max* increases, although the run time does not increase dramatically. Similarly, again as might be expected, it takes longer to process the larger graph sets than the smaller graph sets. As can be confirmed by inspection of Table 4, with respect to AISF1, AISF2, AISF3 and AISF5, Min-BFS-REUSM stops at $k = 4$, 6, 5 and 6 respectively. This is because at these values of k the coverage reaches 100% and the process is complete. Thus, in these cases the timings are almost the same for

values of *max* greater than the k value when maximum coverage was reached. The reason for OME (Out of Memory Error) in the case of AISF8 and AISF9 is discussed in sub-section 4.5 below.

graph	*max* Value						graph	*max* Value					
set	3	4	5	6	7	8	set	3	4	5	6	7	8
AISF1	0.07	0.12	0.13	0.13	0.13	0.13	AISF6	0.17	0.29	0.51	0.68	0.77	0.97
AISF2	0.08	0.10	0.16	0.17	0.16	0.21	AISF7	0.33	0.42	0.61	0.65	0.73	0.87
AISF3	0.09	0.26	0.34	0.37	0.39	0.41	AISF8	0.31	0.41	0.70	0.81	1.69	OME
AISF4	0.16	0.19	0.27	0.29	0.32	0.47	AISF9	0.33	0.56	0.92	1.34	2.16	OME
AISF5	0.14	0.23	0.23	0.28	0.35	0.35	AISF10	0.32	0.45	0.60	0.74	0.88	1.43

Table 2: Run time (seconds) comparison for a range of *max* values

4.4 Total number of discovered minimal VULS

Table 3 presents the total number of discovered minimal VULS using Min-BFS-RESUM. From Table 3 it can be observed, as might be anticipated, that as the *max* value increases the total number of discovered minimal VULS increases. As already noted above, for AISF1, AISF2, AISF3 and AISF5, the coverage reaches 100% when $k = 4$, 6, 5 and 6 respectively at which point the algorithm stops. This is why in these cases the total number of identified VULS remains static at 24, 37, 69 and 79 for *max* values in excess of k.

graph	*max* Value						graph	*max* Value					
set	3	4	5	6	7	8	set	3	4	5	6	7	8
AISF1	9	24	24	24	24	24	AISF6	6	22	94	334	356	395
AISF2	7	13	29	37	37	37	AISF7	11	27	100	226	318	369
AISF3	1	11	69	69	69	69	AISF8	4	22	103	267	717	OME
AISF4	5	16	48	75	93	104	AISF9	4	26	125	272	401	OME
AISF5	13	32	51	79	79	79	AISF10	26	70	96	136	193	279

Table 3: Total number of minimal VULS discovered for a range of *max* values

4.5 Coverage

A comparison of coverage is presented in Table 4. As was to be expected, as *max* increases the coverage rate increases. It should be noted that in some cases the coverage will not reach 100% because no more minimal VULS can be discovered. It is interesting to note, with respect to Table 4 that for AISF1 to AISF7 100% coverage is reached (more or less). It is anticipated that if some efficiency gains can be realised 100% coverage would also be achieved for AISF8 to AISF10. This is an excellent result with respect to AISF sheet metal forming application used as a focus with respect to the evaluation presented in this section.

In the context of AISF8 and AISF9 the OME occurs because: (i) these graphs are much larger than the graphs for AISF1 to AISF6, and (ii) fewer VULS were discovered during early stages of the Min-BFS-REUSM process. AISF7, AISF8, AISF9 and AISF10 were generated from the same 21×21 grid data set

graph set	*max* Value						graph set	*max* Value					
	3	4	5	6	7	8		3	4	5	6	7	8
AISF1	52.8	100.0	100.0	100.0	100.0	100.0	AISF6	19.0	40.0	67.0	85.0	99.0	99.0
AISF2	86.1	91.7	97.2	100.0	100.0	100.0	AISF7	64.2	92.1	93.0	99.6	99.8	99.8
AISF3	13.9	47.2	100.0	100.0	100.0	100.0	AISF8	24.9	70.5	71.7	71.7	71.7	OME
AISF4	59.0	87.0	98.0	99.0	99.0	99.0	AISF9	7.0	27.7	58.7	69.8	76.0	OME
AISF5	80.0	81.0	81.0	100.0	100.0	100.0	AISF10	76.4	86.9	87.3	87.3	87.3	88.2

Table 4: Coverage (%) for a range of *max* values

as described in subsection 4.2 above, however discretised in different manners using different numbers of edge and vertex labels (as shown in table 1). As k increases the rate at which the copy of the input graph G is pruned is slower for AISF8 and AISF9 than for AISF7 and AISF10 (as can be observed from Table 4). For example for $max = 3$ many fewer $k = 1$, 2 and 3-edge minimal VULS are discovered with respect to AISF8 and AISF9 than for AISF7 and AISF10. Thus for these two data sets the size of G is not significantly reduced during the early stages of Min-BFS-REUSM and hence the memory error occurs. This observation can be validated with reference to Table 5 which shows the number of vertices and edges contained in the AISF7, AISF8, AISF9 and AISF10 data sets before, during and after the application of the process of Min-BFS-REUSM when $max = 8$. From the table it can clearly be seen that many more edges and vertices are removed from G with respect to AISF7 and AISF10 than with respect to AISF8 and AISF9.

graph set	Prior to start		On completion		Removed		graph set	Prior to start		On completion		Removed	
	# verts.	# edges	# verts.	# edges	# verts.	# edges		# verts.	# edges	# verts.	# edges	# verts.	# edges
AISF7	441	840	291	214	150	626	AISF9	441	840	355	392	86	448
AISF8	441	840	372	460	69	380	AISF10	441	840	335	341	106	499

Table 5: Summary of AISF7 to AISF10 graph sets before, during and after application of the Min-BFS-REUSM process when $max = 8$

5 Conclusions and further Study

In this paper we have introduced the concept of VULSM which, as illustrated, has application with respect to error prediction in sheet metal forming. We have also introduced the Min-BFS-RESUM algorithm an algorithm for identifying all minimal VULS in a given input graph. The significance of finding minimal VULS is that they can be used to build larger VULS, it is thus computationally efficient to find only minimal VULS than finding all VULS. The reported experimental results also indicate that our algorithm can successfully identify all minimal VULS in reasonable time and with (in some cases) excellent coverage (an important requirement in the context of the AISF sheet metal forming application used as a focus for the work). Having established a "proof of concept"

there are many interesting research problems related to VULSM that can now be pursued. For instance Min-BFS-REUSM can be modified to identify VULS in directed graphs or trees. The concept of Frequent VULSM (FVULSM) may be realised by combining the techniques of FSM (Frequent Subgraph Mining) and VULSM.

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. 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 (2002)
2. Cafuta, G., Mole, N., tok, B.: An enhanced displacement adjustment method: Springback and thinning compensation. Materials and Design 40, 476–487 (2012)
3. 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)
4. 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)
5. 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)
6. 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)
7. 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)
8. 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)
9. 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)
10. 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)
11. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: Proceedings of the 2002 International Conference on Data Mining. p. 721 (2002)
12. 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)