Technical Notes on Extending gSpan to Directed Graphs

Cane Wing-ki Leung
School of Information Systems
Singapore Management University
Singapore 178902
cane.leung@gmail.com

October 6, 2010

1 About this Article

This article describes an extension of gSpan [3, 2] to directed graphs. It also aims to provide supplementary materials for our paper on mining link formation rules [1], in which we propose the LFR-Miner algorithm by extending gSpan, but were unable to detail parts of the extension due to space constraint.

We assume readers of this article to have essential knowledge about gSpan, and our focus is on the revisions required for gSpan's definitions of DFS edge, DFS code's neighborhood restriction and DFS lexicographic order.

2 Preliminary Concepts

2.1 Directed and Labeled Graph

We consider in this article a simple directed and labeled graph[1], defined as a 4-tuple $G = (V, E, L, l)$. V is a set of vertices/nodes representing individuals in the network. E is a set of directed links/edges representing relationships between individuals. An element $(u, v) \in E$, where $u, v \in V$, $u \neq v$, is an edge from u to v. L is a finite set of node and edge labels. $l : V \cup E \rightarrow L$ assigns labels to elements in V and E.

This definition can be generalized to model a partially labeled graph if the label set L includes an empty label. G is essentially an unlabeled graph if L is a singleton set.

2.2 Overview of gSpan

This subsection introduces several elements in gSpan to provide context for the subsequent discussions.

DFS Subscripting

Given a graph pattern p, gSpan applies *DFS subscripting* to all nodes in p to encode the order in which they are traversed in a certain DFS tree. For two "DFS subscripted" nodes v_i and v_j, $i < j$ *iff* v_i is traversed before v_j. We use p_T to denote a DFS subscripted p. The first and the last nodes traversed in a given p_T are called the *root* and the *rightmost vertex* respectively.

[1] Our work in [1] considers a directed, labeled and *temporal (time-stamped)* graph. We omit the temporal aspect of the graph here because it is not concerned with edge directionality. It plays a role only in the enumeration of pattern occurrences in our work.

In building a certain p_T, an edge that is visited is called a *forward edge*, while one that is not visited a *backward edge*. An edge in p_T can be simply represented as (v_i, v_j) or (i, j). It is a forward edge if $i < j$, and a backward edge otherwise. The path from the root to the rightmost vertex, built upon forward edges, is called the *rightmost path*.

DFS Code, DFS Lexicographic Order and Minimum DFS Code

Each p_T can be mapped to a DFS code, which comprises a DFS edge sequence. Each DFS edge corresponds to one edge in p. gSpan defines a linear order among DFS edges, which in turns helps define a linear *DFS lexicographic order* among all DFS codes. A pattern p may be represented by multiple DFS codes. gSpan defines the canonical label of p as its *minimum DFS code*, as determined by the DFS lexicographic order.

3 Adding Edge Directions

A few definitions proposed in gSpan have to be revised to take care of edge directions and *mutual links* that may exist in directed graphs.

3.1 DFS Edge with Direction

Yan and Han mentioned in [4] that gSpan can be extended to handle directed graphs by adding a *direction* value to a DFS edge. In addition, we should define a *lexicographic order among edge directions* to ensure the correctness of DFS lexicographic order, which is crucial for minimum DFS code computation.

DFS Edge Representation

A DFS edge with edge direction can be represented as a 6-tuple: $\langle i, j, l_i, l_{(i,j)}, l_j, d_{(i,j)} \rangle$ as mentioned in [4]. i and j are DFS subscripts, l_i and l_j are the labels of vertices v_i and v_j respectively, $l_{(i,j)}$ is the edge label, and $d_{(i,j)}$ takes two possible values to capture the direction of the edge. We define $d_{(i,j)} = \rightarrow$ if the edge points from v_i to v_j, and $d_{(i,j)} = \leftarrow$ if it points from v_j to v_i.

Fig. 1 depicts a sample pattern p with directed edges. Table 1 lists the two possible DFS codes of p, denoted by p_{T1} and p_{T2}. This example illustrates how DFS edge with directions and mutual links between a node pair can be represented.

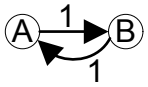

Figure 1: A sample pattern p.

Edge	p_{T1}	p_{T2}
p_0	$\langle 0, 1, A, 1, B, \rightarrow \rangle$	$\langle 0, 1, B, 1, A, \rightarrow \rangle$
p_1	$\langle 1, 0, B, 1, A, \rightarrow \rangle$	$\langle 1, 0, A, 1, B, \rightarrow \rangle$

Table 1: Examples of DFS code with edge directions.

Lexicographic Order Among Edge Directions

We define $\prec_D$ to be the lexicographic order among $d_{(i,j)}$'s (edge directions), and $d_{(i,j)} = \rightarrow$ is lexicographically smaller than $d_{(i,j)} = \leftarrow$. $\prec_D$ is essential in extending gSpan to directed graphs because every edge can now take two possible directions, and mutual links may exist between a node pair. When two DFS edges have the same values of $i, j, l_i, l_{(i,j)}, l_j$ *but different directions*, $\prec_D$ is used to determine their lexicographic order.

3.2 DFS Code's Neighborhood Restriction

DFS code's neighborhood restriction property (Property 1 in [3]) restricts the i and j indices of two neighboring DFS edges in any DFS code. This property has to be revised to take mutual links into account.

Given a DFS code $\mathbf{x} = (x_0, x_1, ..., x_m)$, let $m \geq 2$. Let x_k and x_{k+1}, where $0 \leq k < m$, be two neighboring edges in $\mathbf{x}$. $x_k = \langle i_k, j_k, l_{i_k}, l_{(i_k,j_k)}, l_{j_k}, d_{(i_k,j_k)}\rangle$, and $x_{k+1} = \langle i_{k+1}, j_{k+1}, l_{i_{k+1}}, l_{(i_{k+1},j_{k+1})}, l_{j_{k+1}}, d_{(i_{k+1},j_{k+1})}\rangle$. Also, let $E^f_{\mathbf{x}}$ denotes the forward edge set of $\mathbf{x}$, and $E^b_{\mathbf{x}}$ denotes the backward edge set of it. The revised DFS code's neighborhood restriction property states that x_k and x_{k+1} must agree with the following rules:

Rule 1: if $x_k \in E^b_{\mathbf{x}}$, then either of the following is true:

(i) if $x_{k+1} \in E^f_{\mathbf{x}}$, $i_{k+1} \leq i_k$ and $j_{k+1} = i_k + 1$ as in gSpan.

(ii) if $x_{k+1} \in E^b_{\mathbf{x}}$, $i_{k+1} = i_k$ and $j_k \leq j_{k+1}$.

Rule 2: if $x_k \in E^f_{\mathbf{x}}$, then either of the following is true:

(i) if $x_{k+1} \in E^f_{\mathbf{x}}$, $i_{k+1} \leq j_k$ and $j_{k+1} = j_k + 1$ as in gSpan.

(ii) if $x_{k+1} \in E^b_{\mathbf{x}}$, $i_{k+1} = j_k$ and $j_{k+1} \leq i_k$.

Condition (ii) of both Rule 1 and Rule 2 allow for mutual links to exist. Consider the sample pattern q in Fig. 2, and one of its possible DFS codes q_{T1} in Table 2. The neighboring edges q_2 and q_3 satisfy Condition (ii) of Rule 1. Referring to the sample pattern p in Fig. 1, the neighboring edges p_0 and p_1 in both codes in Table 1 satisfy Condition (ii) of Rule 2.

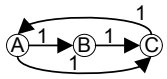

Figure 2: A sample pattern q.

Edge	q_{T1}
q_0	$\langle 0, 1, A, 1, B, \rightarrow\rangle$
q_1	$\langle 1, 2, B, 1, C, \rightarrow\rangle$
q_2	$\langle 2, 0, C, 1, A, \rightarrow\rangle$
q_3	$\langle 2, 0, C, 1, A, \leftarrow\rangle$

Table 2: A DFS code of q built by taking node A as v_0.

3.3 DFS Code Lexicographic Order

Recall that $\prec_D$ defines the lexicographic order among edge directions. Let $\prec_L$ be the lexicographic order among labels in the label set L. We now define the revised DFS code lexicographic order as follows.

Given are two DFS codes $\mathbf{x} = (x_0, x_1, ..., x_m)$ and $\mathbf{y} = (y_0, y_1, ..., y_n)$. Let $E^f_{\mathbf{x}}$ and $E^b_{\mathbf{x}}$ respectively be the forward edge set and the backward edge set of $\mathbf{x}$, and similar for $E^f_{\mathbf{y}}$ and $E^b_{\mathbf{y}}$. Also, let $x_t = \langle i_x, j_x, l_{i_x}, l_{(i_x,j_x)}, l_{j_x}, d_{(i_x,j_x)}\rangle$, and $y_t = \langle i_y, j_y, l_{i_y}, l_{(i_y,j_y)}, l_{j_y}, d_{(i_y,j_y)}\rangle$ be the t^{th} DFS edge in $\mathbf{x}$ and in $\mathbf{y}$ respectively. $\mathbf{x} \leq \mathbf{y}$ *iff* either of the following is true:

(i) $x_k = y_k$ for $0 \leq k \leq m$, and $m \leq n$.

(ii) $\exists t, 0 \leq t \leq \min(m, n), x_k = y_k$ for $k < t$, and $x_t \prec_e y_t$,

where $x_t \prec_e y_t$ is true if either of the following conditions is true:

1. $x_t \in E^b_{\mathbf{x}}$, and $y_t \in E^f_{\mathbf{y}}$.
2. $x_t \in E^b_{\mathbf{x}}$, $y_t \in E^b_{\mathbf{y}}$, and $j_x < j_y$.
3. $x_t \in E^b_{\mathbf{x}}$, $y_t \in E^b_{\mathbf{y}}$, $j_x = j_y$, and $l_{(i_x,j_x)} \prec_L l_{(i_y,j_y)}$.
4. $x_t \in E^b_{\mathbf{x}}$, $y_t \in E^b_{\mathbf{y}}$, $j_x = j_y$, $l_{(i_x,j_x)} = l_{(i_y,j_y)}$, and $d_{(i_x,j_x)} \prec_D d_{(i_y,j_y)}$.
5. $x_t \in E^f_{\mathbf{x}}$, $y_t \in E^f_{\mathbf{y}}$, and $i_y < i_x$.
6. $x_t \in E^f_{\mathbf{x}}$, $y_t \in E^f_{\mathbf{y}}$, $i_x = i_y$, and $l_{i_x} < l_{i_y}$.
7. $x_t \in E^f_{\mathbf{x}}$, $y_t \in E^f_{\mathbf{y}}$, $i_x = i_y$, $l_{i_x} = l_{i_y}$, and $l_{(i_x,j_x)} \prec_L l_{(i_y,j_y)}$.
8. $x_t \in E^f_{\mathbf{x}}$, $y_t \in E^f_{\mathbf{y}}$, $i_x = i_y$, $l_{i_x} = l_{i_y}$, $l_{(i_x,j_x)} = l_{(i_y,j_y)}$, and $l_{j_x} \prec_L l_{j_y}$.
9. $x_t \in E^f_{\mathbf{x}}$, $y_t \in E^f_{\mathbf{y}}$, $i_x = i_y$, $l_{i_x} = l_{i_y}$, $l_{(i_x,j_x)} = l_{(i_y,j_y)}$, $l_{j_x} = l_{j_y}$, and $d_{(i_x,j_x)} \prec_D d_{(i_y,j_y)}$.

Note that Conditions 4 and 9 above consider $\prec_D$ when ordering DFS edges.

3.4 Remarks on DFS Code Children Generation

As described in [3], we need to generate all potential children C of a given DFS code p and count their support. One possible way to do this, among others, is to generate C by adding one edge to p first, and then count their support in the actual graphs. It is worth mentioning that one shall detect and disallow *multiple edges* from one node to another in this children generation process. We explain this with an example. Consider a code p with only one edge $p_0 = \langle 0, 1, A, 1, B, \rightarrow \rangle$. Suppose we grow p to obtain a child c by an edge $p_1 = \langle 1, 0, B, 1, A, \leftarrow \rangle$. Although p_0 and p_1 do not violate the revised neighborhood restriction property and DFS lexicographic order, they are in fact multiple edges pointing from node v_0 to node v_1. One can detect similar cases of multiple edges at a negligible cost and discard them in the first place.

Another approach to generating and counting the support of the children set C of p does not require generating C in advance. Instead, this approach adds a child c to C and counts it whenever it is found given an occurrence of p. Taking this approach does not require handling the problem of multiple edges, because they would never be found given that the input graph G is a simple graph.

4 Contact

Any comment or question regarding this article would be greatly appreciated, and could be forwarded to Cane Leung at `cane.leung@gmail.com`.

References

[1] C. W. K. Leung, E. P. Lim, D. Lo and J. Weng. Mining Interesting Link Formation Rules in Social Networks. In Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM), 2010.

[2] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining, University of Illinois at Urbana-Champaign, Tech. Rep. UIUCDCS-R-2002-2296, 2002.

[3] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining, in Proceedings of ICDM, pages 721–724, 2002.

[4] X. Yan and J. Han. CloseGraph: Mining closed frequent graph patterns. In KDD, pages 286–295, 2003.