Semi-structure Mining Method for Text Mining with a Chunk-based Dependency Structure

Issei Sato¹ and Hiroshi Nakagawa²

¹ Graduate School of Information Science and Technology, The University of Tokyo
² Interfaculty Initiative in Information Studies

sato@r.dl.itc.u-tokyo.ac.jp¹, nakagawa@dl.itc.u-tokyo.ac.jp²

Abstract. In text mining, word frequency is an important element. However, when we need more precise information than word frequencies such as the relationships among words, it is necessary to extract frequent patterns of words with a dependency structure in a sentence. This paper proposes a semi-structure mining method for extracting frequent patterns of words with a dependency structure from a text corpus. First, it describes the data structure representing the dependency structure. This is a tree structure in which each node has multiple items. Then, a mining algorithm for this data structure is described. Our method can extract frequent patterns that cannot be extracted by conventional methods.

1 Introduction

A huge amount of text data has been accumulated and is available in companies, on the worldwide web, and in other places. This makes text mining techniques critical. In text mining, word frequency is certainly an important element. However, when we need more precise information than word frequencies such as the relationships among words, it is necessary to extract frequent patterns of words with a dependency structure within a sentence. In Japanese, the dependency structure is a chunk-based dependency structure, which is a dependency structure based on a chunk of words as a unit. Some examples of word chunks are verb phrases in Japanese and noun phrases and prepositional phrases in English. For example, Figure 1(a) shows the chunk-based dependency structure of "安倍総理が中国に訪れた (Japan's premier Abe went to China)". Generally speaking, a chunk-based dependency structure is represented as a tree structure in which a chunk of words is regarded as one label. This enables conventional semi-structure mining algorithms such as FREQT[5,6] to be used. However, the representation that regards a chunk of words as one label causes the following problem. If we want to extract frequent substructures from two trees (a) and (b) in figure 1 based on this representation schema, substructure (c) is extracted. It is a serious problem that substructure (c) has so few nodes that there is insufficient information about the relationship among words. Consequently, patterns that have too few nodes are extracted. In this case, relationships such as "安倍 (Abe)" , "訪れた (went)" and "中国 (China)" are not extracted.
Fig. 1. The Example of Extracting Sub-structure by Conventional Methods. Fig. 2. Example of Extracting Substructure by Our Method.

To solve this problem, we propose a new data structure that represents a chunk-based dependency structure. It is a tree structure in which each node has multiple items rather than one label. The multiple items correspond to the words in a chunk. In addition, we propose a mining algorithm for this data structure. Our mining algorithm is an improved version of the sequential pattern mining algorithm PrefixSpan[1,2]. An example of the data structure is shown in Figure 2. Each node has multiple items and each item is a word. In Figure 2, substructure (c) is extracted from two trees (a) and (b). The key point is that in Figure 2, it is possible to extract nodes with more items than those in Figure 1. Substructure (c) in Figure 2 has three nodes, while substructure (c) in Figure 1 has one node. Therefore, our method can extract patterns that have much more information, such as the extracted patterns shown in Figure 3.

In the work reported in this paper, our aim was to extract frequent patterns with more nodes from text data with a chunk-based dependency structure.

The remainder of this paper is organized as follows. Sections 2 and 3 introduce, as related work, sequential pattern mining and its algorithm PrefixSpan and labeled ordered tree mining and its algorithm FREQT, respectively. Section 4 explains our method and Section 5 evaluates it. Section 6 summarizes our work.

2 Sequential Pattern Mining[1,2]

2.1 Term Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. This set of items is also called an element. An element is denoted by $(i_1, i_2, \ldots, i_m)$. An example of an element is 

(a, b)" in Figure 3. An inclusive relation between element $e_1$ and element $e_2$ is denoted as $e_1 \subseteq e_2$ if all items of $e_1$ are included in $e_2$. Since an element
is a set of items, it is necessary to sort items in an element in lexical order in advance: it is possible to consider the original order of items if we do not sort them in the element. A sequence is an ordered list of elements. For example, the sequence \(<(a, b), (a, d)>\) is not equivalent to \(<(a, d), (a, b)>\).

A sequence \(s\) is denoted by \(s = <e_1, e_2, \ldots, e_l>\) where \(e_k\) is an element. The number of items in a sequence is called the length of the sequence. A sequence with length \(L\) is called an \(L\)-sequence. A sequence \(\alpha =< a_1, a_2, \ldots, a_n>\) is called a subsequence of another sequence \(\beta =< b_1, b_2, \ldots, b_m>\) if there exist integers \(1 \leq j_1 \leq j_2 \leq \cdots \leq j_n \leq m\) such that \(a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \cdots, a_n \subseteq b_{j_n}\) This relationship between \(\alpha\) and \(\beta\) is denoted as \(\alpha \subseteq \beta\).

A sequence database \(S\) is a set of tuples \((\text{sid}, s)\), where \(\text{sid}\) is a sequence id and \(s\) is a sequence defined as follows:

\[
S = \{(\text{sid}_1, s_1), (\text{sid}_2, s_2), \ldots, (\text{sid}_n, s_n)\}
\]

The support of a sequence \(\alpha\) in a sequence database \(S\) is the number of tuples containing \(\alpha\) in the database defined as follow:

\[
support_\alpha(s) = |\{(\text{sid}, s) | (\text{sid}, s) \in S \land \alpha \subseteq s\}|
\]

A frequent sequence \(\alpha\) is defined as a sequence whose support is greater than the minimum support \(\zeta\), which is a threshold, i.e., \(support_\alpha(s) \geq \zeta\).

Sequential Pattern Mining is defined as the problem of extracting all frequent sequences from a sequence database.

### 2.2 PrefixSpan [2]

Pei et al. proposed PrefixSpan in 2000 as a fast sequential pattern mining algorithm[2]. PrefixSpan extracts frequent sequences with a depth-first search by recursively executing projection operations, which is called Prefix projection.

Term Definition Given a sequence \(s = <e_1, e_2, \ldots, e_n>\) and item \(a\), it is assumed that there exists a positive integer \(m (\leq n)\) such as \(e_1 \not\subseteq a, e_2 \not\subseteq a, \ldots, e_{m-1} \not\subseteq a, e_m \subseteq a\). Moreover, suppose that \(e_m = (a_1, a_2, \ldots, a_l), e_{m}^+ = (a_1, a_2, \ldots, a_k) \not\subseteq a(k < j), a_j = a\), and \(e_{m}^- = (a_{j+1}, \ldots, a_l)\). A sequence \(<e_1, e_2, \ldots, e_m^->\) is defined as the prefix of \(s\) based on item \(a\). A sequence \(<e_m^+, e_{m+1}, \ldots, e_n>\) is defined as the postfix of \(s\) based on item \(a\). If there is no \(m\), prefix and postfix are not defined.

Prefix projection The Prefix projection of a sequence database \(S\) with item \(a\) is defined as the operation that constructs a projected database from the postfixes of sequences based on item \(a\). It adds prefix \(\sim\) to items that are in the same element as a projecting item.

An \(<a>\)-projected database \(S\) is defined as a database that is projected with item \(a\) and is denoted by \(S_{<a>}\). Figure 3 shows examples of projection and of a projected database.

In Figure 3, suppose that sequence database \(S\) is projected with item \(a\). The postfix of each \(\text{sid}\) is \(\text{sid} = 10 :< (b)(c) >, \text{sid}' = 20 :< (b)(c) >,\)
\[ \text{sid} = 30 : \langle \text{j} \rangle (b, c) \rangle. \text{Therefore a set of these tuples is an} \langle a \rangle \text{-projected database} S_{<a>}. \]

**PrefixSpan Algorithm** The PrefixSpan algorithm is as follows.

1. **Find length 1-frequent sequences**
   Scan a sequence database \( S \) to find length 1-frequent sequences (= frequent items) whose supports are greater than the minimum support. Then, all frequent sequences are partitioned into subsets that have the length 1-frequent sequences as each prefix.

2. **Find subsets of length \( k \)-frequent sequence**
   For \( k(\geq 2) \), do the following procedure by incrementing \( k \) until frequent items can not be extracted. Each subset of the \( k \)-frequent sequence is mined recursively by constructing a projected database corresponding to each \( (k-1) \)-frequent sequence.

Figure 3 shows the running example of PrefixSpan. Given that the minimum support is two, the extracted length 1-frequent sequences (= frequent items) are \( a,b \) and \( c \). All frequent sequences are partitioned into subsets whose prefixes are \( a,b \) and \( c \), respectively.

The frequent sequences whose prefix is \( a \) are extracted from the \( <a> \)-projected database \( S_{<a>} \). The length 2-frequent sequences whose prefix is \( a \) are \( <(a,b)>, <(a)/(b)>, and <(a)/(c)> \). In this case, length 1-frequent sequences (= frequent items) in \( <a> \)-projected database \( S_{<a>} \) are \( b \) and \( c \). In the same way, the frequent sequences whose prefix is \( <(a,b) > \) are extracted from \( <(a,b)> \)-projected database \( S_{<(a,b)>} \), which is constructed by projecting \( <a> \)-projected database \( S_{<a>} \) with \( \text{b} \). When all frequent sequences whose prefix is \( a \) have been extracted, the extraction of frequent sequences whose prefix is \( b \) starts. As described above, PrefixSpan recursively extracts frequent sequences with a depth-first search.

### 2.3 Constraint-based Sequential Pattern Mining

Pei et al. indicate that other constraints as well as minimum support should be used [3]. For example, Hirate et al. apply time constraint to sequential pattern mining of time series data [4]. Our proposed mining algorithm is also categorized into this type of constraint based sequential pattern mining.

### 3 Labeled Ordered Tree Mining

#### 3.1 Term Definition

Let \( L = \{l_1, l_2, \ldots, l_n\} \) be a finite set of labels. A **labeled ordered tree** on \( L \) is defined as \( \{V,E,B,label, u_0\} \) satisfying the following properties. \( V \) is a set of nodes. \( E \subseteq V^2 \) represents the binary relationship between a parent node and a child node. \( E \) is also called a branch. \( B \) keeps information about the order in a
sibling relationship. \( \text{label} : V \rightarrow L \) is a projection that assigns a label \( l \in L \) to each node \( v \in V \). \( v_0 \) is the root node which does not have a parent node. A node that does not have a child node is called a leaf node. Given a node sequence \( P_v = \langle v_1, \ldots, v_n \rangle \) (\( v_j \in V \)), \( P_v \) is defined as the path from \( v_1 \) to \( v_n \) if and only if \( \langle v_i, v_{i+1} \rangle \in E \) for all \( i(1 \leq i < n) \). The depth of \( v \) is defined as the number of nodes in the path from \( v_0 \) to \( v \) and is denoted by \( \text{depth}(v) \). Hereinafter, the labeled ordered tree is simply referred to as ordered tree.

If an ordered tree \( \alpha \) is a subset of another ordered tree \( \beta \), it is defined that \( \beta \) 'includes' \( \alpha \), denoting \( \alpha \subseteq \beta \). Specifically, it is necessary that there exists such a projection \( \varphi : \alpha \rightarrow \beta \) which has the following three properties:

1. \( \varphi \) is an injection.
2. \( \varphi \) keeps the parent-child relationship.
3. \( \varphi \) keeps the sibling relationship; however, note that nodes neighboring in \( \alpha \) do not need to be neighboring in \( \beta \).

An ordered tree database is a set of tuples \((\text{tid}, t)\), where \( \text{tid} \) is an ordered tree id and \( t \) is an ordered tree defined as follows.

\( T = \{(\text{tid}_1, t_1), (\text{tid}_2, t_2), \ldots, (\text{tid}_n, t_n)\} \)

The support of an ordered tree \( \alpha \) in an ordered database \( T \) is the number of tuples in the database containing \( \alpha \) defined as follows:

\( \text{support}_T(\alpha) = || \{ (\text{tid}, t) \mid (\text{tid}, t) \in T \land \alpha \subseteq t \} || \)

A frequent ordered tree \( \alpha \) is defined as an ordered tree whose support is greater than minimum support \( \zeta \), which is the threshold, i.e., \( \text{support}_T(\alpha) \geq \zeta \).

Labeled Ordered Tree Mining is defined as the problem of extracting all frequent ordered trees from an ordered tree database corresponding to the minimum support.
The number of nodes of an ordered tree $t$ is called the size of an ordered tree $t$ and is denoted as $|t|$. A frequent ordered tree whose size is $l$ is called a $l$-frequent ordered tree.

3.2 FREQT[5, 6]

Abe, Asai, and Arimura et al. proposed FREQT in 2002 as a fast labeled ordered tree mining algorithm [5, 6]. FREQT extracts frequent ordered trees by the technique of growing a tree by attaching new nodes only on the rightmost branch of the tree, which is called rightmost expansion. Rightmost expansion was also proposed by Zaki et al.[7]. Rightmost expansion is the extension of Item Enumeration Tree by Bayardo[8] to an ordered tree.

FREQT Algorithm The FREQT algorithm is as follows.

1. Find size 1-frequent ordered trees. $k$ is set to 2.
2. Generate candidates trees of a size $k$-frequent ordered tree by attaching new nodes on size $(k - 1)$-frequent ordered tree.
3. Extract from the candidates the size $k$-frequent ordered trees whose support is greater than the minimum support.

Procedures (2) and (3) are applied recursively. In this regard, however, since the arbitrary attachment of new node on size $(k - 1)$-trees generates a redundant size $k$-ordered trees, it is necessary to constrain the attachment procedure. This constraint is called the rightmost expansion.

In rightmost expansion, the rightmost leaf node is defined as the node that is reached after traversing an ordered tree from a root node with a depth-first search in the anticlockwise direction. The rightmost branch is defined as the path from the root node to the rightmost leaf node. Rightmost expansion is the procedure to attach a new node on a size $(k - 1)$-ordered tree in such a way that the new node becomes the rightmost leaf node of the size $k$-ordered tree. Figure 4 shows an example of the rightmost expansion that attaches a node $e$ to an ordered tree.

4 Proposed Method

A chunk-based dependency structure was previously represented as a tree structure and a chunk of words is assigned as one label to each of its nodes. But the representation causes the extraction of substructures having few nodes. To solve this problem, we propose a new data structure to represent the chunk-based dependency structure and a mining algorithm for the new data structure. This is a tree structure and each node has an element, that is, a set of items. For example, a chunk such as "with a depth-first search" is represented as a node with an element including four items, i.e., ( 'with', 'a', 'depth-first', 'search' ). In this regard, however, items in the element keep their order. This section explains the algorithm for extracting frequent patterns from this data structure.
4.1 Overview of Proposed Method

Since our data structure is not a labeled ordered tree, an existing mining algorithm such as FREQT cannot be applied directly. But, if we transform the data in the following way, we can make the sequential pattern mining algorithm applicable to our data structure.

1. Enumerate elements by traversing the data structure from the root node with a depth-first search in the anticlockwise direction.
2. Assign an index to each node in the enumerated order. As the result, the data structure becomes one that can be dealt with sequential pattern mining. In this regard, however, items in an element keep their order.
3. Each element has index information about a structure, i.e., (parent, first child, next sibling). If there is none, the index is set to -1. The first child of a node is the leftmost node among its child nodes. The next sibling of a node is a neighboring node among its sibling nodes, that is, the leftmost node among its sibling nodes.

The transformed data structure is called a semi-structured sequence.

An example of the above transformation is shown in Figure 5. The data structure surrounded by the quadrangle in Figure 5 is the data structure of sequential pattern mining. Therefore, the sequential pattern mining algorithm PrefixSpan can be applied to our data structure. However, arbitrary use of PrefixSpan leads to the extraction of disconnected patterns such as \(<(a, b, c)(c, b)\) in Figure 5. Therefore, it is necessary to put a certain constraint on PrefixSpan. Our mining algorithm is an expanded PrefixSpan algorithm with a projection that is constrained in order to extract only connected patterns from the data structure transformed by the method described above. This constrained projection is called a Tree Projection.
4.2 Term Definition

A **projecting stack** is a stack into which projecting items are pushed. After projection with item $i$, the item $i$ is pushed into the projecting stack. When the projection with item $i$ has finished, item $i$ is deleted from the projecting stack. The patterns of items in the projecting stack are frequent patterns. The projecting stack is denoted as Proj-Items. Tree projection is constrained to projection with items which is connected with items in Proj-Items.

Suppose that there exists a path between an element $I$ and element $A$, and that the difference from depth($A$) to depth($I$) is $k$-1. The element $A$ is called the **$k$-th ancestor** of the element $I$. The parent element of the element $I$ is 1st ancestor of the element $I$. The 0th ancestor of the element $I$ is $I$ itself. For example, in Figure 5, the 1st ancestor of element $(c, b)$ is element $(d, a)$ and the 2nd ancestor is element $(a, b, c)$.

4.3 Tree projection

The tree projection of a semi-structured sequence $S$ with item $i$ is a projection constrained as follow.

**Constraint:** The only items included in projected database $S_{<i>}$ are ones that have a path with the items in Proj-Items. Figure 6 shows the running example of tree projection with item $a$ (index=0).

Tree projection is composed of tree projections as follows:

- element-projection
- child-projection
- Level $k$ sibling-projection

After the extraction of frequent items on each projection has finished, the next projecting item is selected and tree projection is called recursively in each projection. We explain each projection as follows.

**element-projection** The element-projection with item $i$ is a projection selecting items whose element is equal to the element of the projecting item $i$. In PrefixSpan, these items are denoted by adding $'\_\_\_'$ as a prefix.

**child-projection** The child-projection with item $i$ is a projection selecting items whose element is the child element of projecting item $i$. Specifically, it operates as follows:

1. Find the index of the first child element of the projecting item $i$
2. Find the index of the next sibling element of the first child element.
3. Find the index of the next sibling element iteratively

As described above, child-projection can extract items that have the parent-child relationship of item $i$. These items are stored into a projected database. Figure 7 shows a running example of child projection. Support that we project
a database with item a (index=0). Since a postfix based on item a(index=0)
is < (b)(b,c)(d,e,f,g)(h) >, Prefix projection puts all these items into
the projected database. But child-projection includes only (b,c)(h) which are child
elements of the projecting item a. These indices are gained in the following ways:

1. Find index(=1) of the first child element (b,c) of the projecting item a(index=0).
   The index(=1) is kept by the projecting item a.
2. Find index(=4) of the next sibling element (h) of the element (b,c).
   The index(=4) is kept by the element (b,c).
3. Find index(=1) of the next sibling element of the element (h).
   The index(=1) is kept by the element (h).
4. Since the found index is -1, finish child-projection.

**Level k sibling-projection** Level k sibling projection with item i is a projec-
tion selecting items whose element is a sibling element of the k-th ancestor of
the projecting item i. Specifically, it operates as follows:

1. Find the index of the k-th ancestor of the projecting item i.
2. Find the index of the next sibling element of the k-th ancestor.
3. Find the index of the next sibling element iteratively.

Suppose that d is the depth of the projecting item i from the projection start
item. Iterate Level k sibling-projection from k=0 to k=d. As described above,
Level k sibling-projection can extract the items that are the sibling elements of
the k-th ancestor of item i. Put these items into a projected database. Figure 8
shows a running example of sibling-projection.

Suppose that Proj-Items is \{a,c(index=0),b,c(index=0)\} and that we project
a database with item d(index=2). Item d(index=2) is pushed into Proj-Items.
Since a postfix based on item d(index=2) is < (c)(e,f,g)(h) >, Prefix projection
includes all these postfixes in projected database. But Level k sibling projection
with item d is a projection selecting items whose element are the sibling elements
of the k-th ancestor of the projecting item $d$. Therefore, only $< (e, f, g)(h) >$ is included in the projected database. These indices are gained in the following ways:

1. $k=0$(Level 0 sibling-projection)
   (a) Find index($=3$) of the next sibling element $(e, f, g)$ of the projecting item $d$ (index=$2$).
      The index($=3$) is kept by the projecting item $d$.
   (b) Find index($=-1$) of the next sibling element of the element $(e, f, g)$.
      The index($=-1$) is kept by the element $(e, f, g)$.
   (c) Since the found index is $-1$, finish Level 0 sibling-projection.

2. $k=1$(Level 1 sibling-projection)
   (a) Find index($=1$) of the 1st ancestor $(b, c)$ of the projecting item $d$ (index=$2$).
      The index($=1$) is kept by the projecting item $d$.
   (b) Find index($=4$) of the next sibling element $(h)$ of the element $(b, c)$.
      The index($=4$) is kept by the element $(b, c)$.
   (c) Find index($=-1$) of the next sibling element of the element $(h)$.
      The index($=-1$) is kept by the element $(h)$.
   (d) Since the found index is $-1$, finish Level 1 sibling-projection.

5 Evaluation of Proposed Method

In this section, we present two evaluations: one evaluating the execution time and the other evaluating the statistics of the number of extracted nodes. In particular, we show that our mining method with our new data structure extracts frequent patterns with more nodes than conventional methods.

5.1 Dataset and Evaluation Experiment

The dataset was Aviation Safety Report 3 gathered by Japan Airlines International Co., Ltd. Dependency parsing by Cabocha 4 is run to one sentence as a unit. In a labeled ordered tree, a segment in Japanese is regarded as a label 5. In our data structure, a segment is an element and a morpheme in a segment is an item 6.

All the experiments were performed on a 1.1 GHz Pentium M PC with 1024 kB of L2 cache and 7.56 MB of main memory. All the programs were written in C++ and compiled with gcc 3.3.3.

No conventional mining method can handle our data structure, so we could not evaluate the execution time of our method in comparison with a conventional method. However, our data structure includes a labeled ordered tree as a subset,

---

3 It cannot identify individuals because individual information was eliminated
4 http://chasen.org/~taku/software/cabocha/
5 refer to Figure 1
6 refer to Figure 2
that is, a labeled ordered tree is our structure where an element has only one item. In this case, tree projection corresponds to the rightmost expansion of FREQT. Therefore, we evaluated the execution time of our method based on a labeled ordered tree. Our method is called pFREQT (projection-based FREQT).

Figure 9 compares the execution time by pFREQT and by FREQT\textsuperscript{7} based on a labeled ordered tree. The minimum support was two. pFREQT extracted frequent patterns faster than FREQT. It also projected an item partially corresponding to the relationship of the projection nodes. Therefore, pFREQT used less memory than FREQT when searching for patterns. Figure 10 shows the difference in memory usage for FREQT and pFREQT. Apparently, the more sentences there were to handle, the greater the additional memory consumed by FREQT compared with pFREQT. Figure 11 shows the statistics of the number of extracted nodes. We extracted frequent patterns with more than two nodes. We made five datasets, which had a total of two thousand sentences. We evaluated the meanscore, median and maximal values of the number of extracted nodes. The horizontal axis shows each dataset. The vertical axis shows the number of extracted nodes. Figure 11 shows that for each set of statistics, more nodes are extracted by our method than by conventional methods. The more nodes that can be extracted, the more relationships among words with a dependency structure can be extracted. Thus, our method can extract frequent patterns with a larger node size than conventional methods. Therefore, our method can extract the relationships among words with a dependency structure that cannot be extracted by conventional methods. In this regard, however, since the number of extracted patterns is a large amount, we have the problem of selecting patterns that are meaningful as knowledge. This is the next step of knowledge mining, so this paper does not treat this problem.

\textsuperscript{7} http://chasen.org/~taku/software/freqt/
Fig. 11. Statistics of the Number of Extracted Nodes.

6 Conclusion

This paper described a semi-structure mining method for extracting frequent patterns of words with a chunk-based dependency structure. It also described a new data structure representing a chunk-based dependency structure and a mining algorithm for it. Our method can extract frequent patterns that the conventional methods cannot extract.

Acknowledgments We appreciate the cooperation of Takashi Saito and Akira Terada of Japan Airlines International Co., Ltd. This research was funded in part by MEXT Grant-in-Aid for Scientific Research on Priority Areas "i-explosion" in Japan.

References