Mining Disjunctive Sequential Patterns From News Stream

Kazuhiro SHIMIZU,[1] Isamu SHIOYA,[2] Takao MIURA[1]

[1] Dept.of Elect.& Elect. Engr., HOSEI University
3-7-2 KajinoCho, Koganei, Tokyo, 184–8584 Japan
[2] Dept.of Management and Informatics, SANNO University
1573 Kamikasuya, Isehara, Kanagawa 259–1197 Japan

Abstract. Frequent disjunctive pattern is known to be a sophisticated method of text mining in a single document that satisfies *anti-monotonicity*, by which we can discuss efficient algorithm based on APRIORI. In this work, we propose a new *online and single-pass algorithm* by which we can extract current frequent *disjunctive patterns* by a weighting method for past events from a *news stream*. And we discuss some experimental results.

1 Motivation

Recently much attention has been focused on quantitative analysis of several features of text thus a new approach, called *text mining*, has been proposed to extend traditional methodologies[13]. Text mining approach comes from *frequency* and *co-occurrence* among text[1], the former means important words arise many times while the latter says related words occur at the same time[10].

On the other hand, recently much attention has been focused on data mining on *data stream* which is huge amount dynamic data on network[6]. Generally we see that data stream has some characteristics;(1)huge amount, (2)high speed, (3)changing data distribution, (4)continuous. Therefor we like algorithms which should be able to output approximate solutions for any time and any stream during the whole process and rapid analysis with limited calculation resources. That's why we cannot apply traditional methods for this problem[6].

Especially data stream such as news articles and broadcast transcription is called *news stream*. Generally we know that issue interval of news articles are irregular. So we assume that time stamp of news articles correspond to content-time of news articles. Under the news stream environment, we see that news articles always occur, and collections of news articles grow incrementally with new ones. We usually have interests in recent ones, i.e., new ones are more important than old.

There have been important research results obtained based on *anti-monotonicity* such as APRIORI[2, 4]. Generally we can avoid vast range of search but not enough to sequence data, since in text mining the property doesn't hold any more and we can't apply APRIORI technique any more. An interesting approach of *disjunctive patterns* has been proposed with a new counting method for frequency to get efficient pattern mining algorithm on single sequence data. The counting method satisfies anti-monotonicity thus we can obtain efficient mining algorithm based on APRIORI[11].

[1] Text means *sequence data*, i.e., an ordered list of information.

Generally end users got used to try the algorithms many times. This is because they want to get and compare several results with various parameter values through trial and error. Therefor they see that the algorithms take very long time for final results with many exhaustive scan. Moreover they need to re-calculate to get another results for independent processes if they change several parameters. To overcome this issue, *Online Analysis* has been proposed[1]. Particularly under data stream environment, online analysis approach is very important for all but impossible re-calculation. In the previous work, we proposed a new method of extracting disjunctive patterns under static data environment by using online analysis approach[12].

Let us see some related works. As data mining algorithm to data stream, we know *Lossy Counting*[9]. This is an online and single pass algorithm which generates guaranteed frequent itemsets quickly for transaction data stream. Moreover this has many improvement[8] used for large sensor networks. However it doesn't consider adaptability to time progress. There have been several weighting methods proposed so far under data stream environment, such as *decay function*[5, 14] and *Tilted-Time Window*[3]. However, all these don't assume data mining of news stream.

In this investigation, we define a new online and single-pass algorithm by which we can extract current frequent disjunctive patterns by a weighting method for past events from a news stream. We introduce disjunctive patterns and a counting measure in section 2. In section 3, we define a problem of mining disjunctive patterns from news stream. Section 4 contains how to construct condensed information, called *Disjunctive Pattern Lattice* (DPL) and how to extract frequent disjunctive patterns from DPL. Section 5 contains some experimental results. We conclude our work in section 6.

2 Disjunctive Patterns

In this work, we consider a word as a unit, called an *item*. Any element in an itemset $I = \{i_1, .., i_L\}, L > 0$ is called *alphabet*, and a *sequence data* (or, article) $S = s_1...s_m$ $(\infty > m > 0)$ is an ordered list of items, m is called a *length* of S, and S is called m-sequence. Note an item may appear many times in S.

A *disjunctive pattern* (or just a pattern) p is a form of $t_1t_2...t_n$ where each t_i is an alphabet a or a disjunction $[a_1a_2...a_m], m > 0$, each a_j is a distinct alphabet. Given two patterns $p = t_1t_2...t_n$ and $q = v_1v_2...v_m, m \leq n$, we say q is a *sub-pattern* of p, denoted by $q \sqsubseteq p$, if there exist $1 \leq j_1 < ... < j_m \leq n$ such that each v_k corresponds to t_{j_k} (denoted by $v_k \sqsubseteq t_{j_k}$ if no confusion arises) satisfying:

> If v_k is an alphabet a, $t_{j_k} = a$ or t_{j_k} is a disjunction containing a
> If v_k is a disjunction $[a_1a_2...a_m]$, we have both $t_{j_k} = [b_1b_2...b_l]$ and $\{a_1, .., a_m\} \subseteq \{b_1, .., b_l\}$

EXAMPLE 1 *ac* is a sub-pattern of *abcd*. Similarly [*ac*] is a sub-pattern of [*abcd*], *bd* is a sub-pattern of [*ab*]*b*[*cd*]*de*, *b* is a sub-pattern of [*ab*], and "ac" is a sub-pattern of [*ab*][*cd*]. However, *ab* is not a sub-pattern of [*ab*], nor [*ab*] is a sub-pattern of *ab*.

We say a pattern $p = t_1t_2...t_n$ *matches* a sequence $S = c_1c_2...c_m$ if t_1 is an alphabet a_1, there exist $t_1 = a_1 = c_{i_1}, 1 \leq i_1 \leq m$ and the sub-pattern $t_2...t_n$ matches $c_{i_1+1}...c_m$, and if t_1 is a disjunction $[a_1a_2...a_m]$, there exists a permutation $a_{j_1}...a_{j_m}$ of $a_1, ..., a_m$ that matches $c_1...c_{i_1}$, and the subpattern $t_2...t_n$ matches $c_{i_1+1}...c_m$.

EXAMPLE 2 Assume S is $\mathtt{a_1a_2b_3b_4b_5a_6}$ where each suffix shows its position in the occurrence. A pattern a matches S 3 times($\mathtt{a_1}, \mathtt{a_2}, \mathtt{a_6}$), ab 6 times ($\mathtt{a_1b_3}, \mathtt{a_1b_4}$, $\mathtt{a_1b_5}, \mathtt{a_2b_3}, \mathtt{a_2b_4}, \mathtt{a_2b_5}$) and $[ab]$ 9 times. Note we can see more frequency by $[ab]$.

Given a sequence data S, a function $\mathcal{M}_S$ from patterns to non-negative integers satisfies *Anti Monotonicity* if for any patterns p, q such that $q \sqsubseteq p$, we have $\mathcal{M}_S(q) \geq \mathcal{M}_S(p)$. In the following, we assume some S and we say $\mathcal{M}$ for $\mathcal{M}_S$. Given $\mathcal{M}$ and an integer $\sigma > 0$ (called *minimum support*), A pattern p is called *frequent* if $\mathcal{M}(p) \geq \sigma$. If $\mathcal{M}$ satisfies anti-monotonicity, for any q such that $\mathcal{M}(q) < \sigma$, there is no frequent p such that $q \sqsubseteq p$. By using this property, we can reduce search space to extract frequent patterns. However, it is not easy to obtain $\mathcal{M}$ satisfying anti-monotonicity. For example, "the number of matching" is not suitable as $\mathcal{M}$ as shown in EXAMPLE 2. There have already been proposed a counting method that satisfies anti-monotonicity[11].

The first idea is called *head frequency*. Given a sequence $S = s_1s_2...s_r$ and a pattern p of $t_1t_2...t_n$, we define a *head frequency* $H(S,p)$ as $H(S,p) = \sum_{i=1}^{r} Val(S,i,p)$ where $Val(S,i,p)$ is 1 if the following holds, and 0 otherwise:

> Let $S(i)$ be a suffix of S from i-th position, i.e., $S(i) = s_i...s_r$. If t_1 is an alphabet a, we have $s_i = a$ and $t_2t_3...t_n$ matches $S(i+1)$. And if t_1 is a disjunction $[a_1a_2...a_m]$, there exists j such that $s_i = a_j$ (for instance, $j = 1$), and $[a_2a_3...a_m]t_2...t_n$ matches $S(i+1)$.

Intuitively $H(S,p)$ describes the number of matching of p from the heading of S or its suffix. However, it is known that the head frequency $H(S,p)$ doesn't satisfy anti-monotonicity. Note that this counting ignores matching appeared in the subsequent sequence. That's why we introduce a new counting $D(S,p)$, called *total frequency*, which means the minimum $H(S,q)$ for any $q \sqsubseteq p$: $D(S,p) = \mathrm{MIN}\{H(S,q) | q \sqsubseteq p\}$

In fact, we have $D(S,p) = \mathrm{MIN}\{H(S,p), D(S,p(2))\}$ where $p(2)$ means a sub-pattern of p that has the length pf p minus 1. And also it is enough to calculate the ones for all the suffixes of p in ascending order of length. Then we can show that $D(S,p)$ satisfies anti-monotonicity. Note there are only n suffixes of p thus we can reduces search space dramatically[11].

EXAMPLE 3 Let S be $\mathtt{caabbbc}$. If $p = ab$, we have $H(S,p) = 2$ and $D(S,p) = 2$. If $p = ac$, we have $H(S,p) = 2$ and $D(S,p) = 2$. And if $p = [ac]$, we see $H(S,p) = 3$ and $D(S,p) = 2$ while p matches S 4 times. In these cases, sub-patterns (i.e., a, c) of ac and $[ac]$ appear more interspersed in S and this is why *total frequency* is different from *head frequency* and the number of matching.

In this work we assume *single* document and the frequency of words corresponds to the importance directly. However we examine a test collection containing multiple articles in this experiment which means that the longer articles become the more words appear, and the frequency doesn't always mean the importance. For this reason we give some *weights* to sequences to avoid the effect of the length. In this work, we introduce *Term Frequency* (TF) for frequent words[7]. For a pattern p and a pattern length of p n, denoted by $H_w(S,p)$, is defined as $H_w(S,p) = \frac{H(S,p)}{|S|-(n-1)}$ where $|S|$ means the number of words in S. Then we give *weightened total frequency*, $D_w(S,p)$, defined as $D_w(S,p) = \mathrm{MIN}\{H_w(S,p), D_w(S,p(2))\}$ In our new method which is proposed at following sections, we utilize $H_w(S,p)$ and $D_w(S,p)$ as both head frequency and total frequency.

3 Problem Definition

Let us define what mining disjunctive patterns from news stream means. Given *3 inputs* of news stream NS, minimum support σ and error bound ϵ, we want to find all frequent patterns which have frequency $(\sigma - \epsilon)$ or above when it is current time $time_{now}$.

Putting it into more specific description, we define an infinite ordered list of sequence data $S_1...S_i...S_j...$, called news stream NS, and S_i means i-th sequence data in NS. Each S_i in NS has time stamp $time_i$, and if $i \leq j$ then $time_i \leq time_j$. In this work, we consider sequence data $S_{i_1}...S_{i_{wsize}}$ with a common $time_i$ as a single sequence data S_i. Window size $wsize$ means the number of sequence data which can be calculated at once. Current time $time_{now}$ is a query time which has different interpretations at each point. In this section, a text pattern is called *frequent pattern* if the frequency is more than a threshold σ.

Most of algorithms which take data stream have to consider huge amount data and long term analysis. Generally, they output approximate frequent itemsets to avoid a lot of computation cost. In this investigation, we propose a new algorithm which is based on *Lossy Counting*[9] for mining disjunctive patterns from news stream. Lossy Counting constructs a data structure by using each item's estimate frequencies and maximum errors. The constructed data structure satisfies two important properties: *Property1* : If an item is in a data structure, then its exact frequency is more than its estimate frequency and less than the sum total of its estimate frequency and maximum error. *Property2* : If an item isn't in a data structure, then its exact frequency is less than an error bound.

The solutions from the constructed data structure at chosen random time are guaranteed frequent itemsets by an error bound ϵ, in other words, they have frequencies which are more than $\sigma - \epsilon$.

Here is a new online and single-pass algorithm which consists of 3 steps.

(1) Generating a collection of candidate patterns from sequence data.
(2) Constructing a disjunctive pattern lattice from a generated collection of candidate patterns.
(3) Extracting a collection of frequent patterns from a constructed disjunctive pattern lattice.

The algorithm goes through step (1) and (2) in a repeated manner whenever new article comes in from news stream. Or the algorithm generates a collection of frequent patterns by step (3) if users want. A collection of candidate patterns are generated efficiently using *list of positions* at step (1), and a particular data structure called *disjunctive pattern lattice* are also constructed at step (2). Then we extract frequent patterns quickly from a constructed disjunctive pattern lattice at step (3).

Here we assume that frequencies on the latest sequence are the most important and we give a *weight* based on timestamp of sequence data, denoted by ω. In the algorithm, we give a *decay function* ω as a weight when we access each node in a disjunctive pattern lattice at step (2) and (3), and consequently, frequencies on sequence data which have old timestamps are decreased exponentially: $\omega = u\lambda^{(time_{now} - time_{last})}$ where $time_{last}$ means last update time of each pattern in a disjunctive pattern lattice and λ means the decay constant $0 \leq \lambda \leq 1$. If λ becomes smaller, then ω becomes smaller too and frequencies are decreased more. Also u describes the unit step function such that $u = 1$ if $(time_{now} - time_{last} \leq \tau)$, and $u = 0$ otherwise, where τ means valid time.

4 Algorithm

Let us discuss how to generate a collection of *candidate patterns* from sequence data S in NS. The main idea is that we keep a list of pairs of *head* and *tail* positions to each frequent pattern p where "head" means starting position of p in S and "tail" means ending position. Let $P_p(head, tail)$ be the two positions of p in S. Then we define a *list of positions of* p (in S), $list_p$, as $\{P_{p_i}(head, tail), ..., P_{p_{max}}(head, tail)\}$ where max means the head frequency and all the head values are distinct by definition. Here is the algorithm to generate a collection of n -candidate patterns C_n from S.

Input: a sequence data S
Output: a collection of n -candidate patterns C_n
Algorithm:
if $n = 1$**:** After scanning S, obtain all 1-frequent patterns and generate the corresponding nodes and the arcs to C_n. Let n be 2.
if $n > 1$**:** By examining the lists, obtain all n-frequent patterns and add the nodes and the arcs to C_n. Let n be $n + 1$.

In this algorithm, we obtain $list_{pq}$ of an n-frequent pattern pq by examining $list_p$ and $list_q$ of $(n-1)$-frequent patterns p, q, described below:

Input: $list_p$, $list_q$
Output: $list_{pq}$
Algorithm:
1. By examining $list_p$ and $list_q$, we find all the pairs of $P_{p_i}(head)$ and $P_{q_j}(tail)$ such that $P_{p_i}(tail) < P_{q_j}(head)$, and add the pair to $list_{pq}$ as $(head, tail)$ of k-th element P_{pq_k}.
2. After completing the match, output $list_{pq}$.

Eventually we have $list_p, list_q$ and $list_{pq}$:

$$list_p = \{P_{p_i}(head, tail), ..., P_{p_{imax}}(head, tail)\}$$
$$list_q = \{P_{q_j}(head, tail), ..., P_{q_{jmax}}(head, tail)\}$$
$$list_{pq} = \{P_{pq_k}(P_{p_i}(head), P_{q_j}(tail))|P_{p_i}(tail) < P_{q_j}(head)\}$$

EXAMPLE 4 Let S be `caabbbc`, and a, b, c be 3 patterns (of 1 item). We get 3 lists $list_a = \{P_{a_1}(2,2), P_{a_2}(3,3)\}$ $list_b = \{P_{b_1}(4,4), P_{b_2}(5,5), P_{b_3}(6,6)\}$, and $list_c = \{P_{c_1}(1,1), P_{c_2}(7,7)\}$. Also we get 2 lists $list_{ab} = \{P_{ab_1}(2,4), P_{ab_2}(3,4)\}$ and $list_{[ac]} = \{P_{[ac]_1}(1,2), P_{[ac]_2}(2,7), P_{[ac]_3}(3,7)\}$.

Given a set I of items and a single sequence S, to examine online analysis for disjunctive patterns, we make up a lattice over a power set 2^I, called *Disjunctive Pattern Lattice* (DPL), as shown in a figure 1. Formally a DPL is a rooted, acyclic directed graph (V, E) where V means a finite set of nodes with labels and E means a set of arcs such that $E \subseteq V \times V$. In a DPL, there exists only one node, called a root, without any label. A node $v \in V$ with the distance $n \geq 0$ from the root corresponds to a candidate pattern of n items (called an n-candidate pattern) with the *label* $(f, \Delta, time_{last})$ where f means its estimated pattern frequency which has been calculated until $time_{last}$, Δ means its pattern's maximum error and $time_{last}$ means last update time of its node. There is an *arc* $e\langle v_1, v_2\rangle \in E$ if and only if n-candidate pattern v_1 has v_2 as an $(n+1)$-candidate sub-patterns for some n.

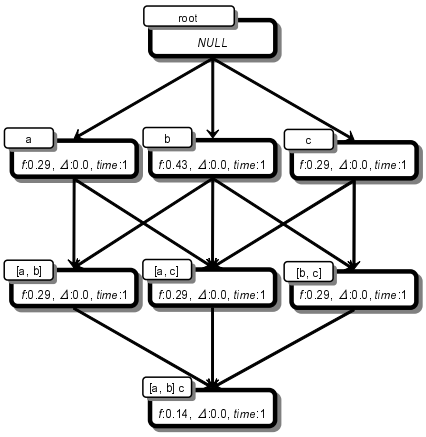

Fig. 1. DPL of `cabcbba` with $\epsilon = 0.1$

EXAMPLE 5 Let $S \in NS$ be a sequence `cabcbba`, $time = 1$ and $\epsilon = 0.1$. Then we have the DPL as in a figure 1.

Here is the algorithm to append $p \in C_n$ to DPL $\mathcal{D}$ and update/delete a node of p in $\mathcal{D}$ from sequence data S_i.

Input: a collection of candidate patterns C, error bound ϵ
Output: DPL $\mathcal{D}$
Operations:
***Insert*:** Insert a node v_p with a label $(\sum D_w(S_i, p), \epsilon N_{last}, time_i)$ to $\mathcal{D}$ if $\exists v_p \notin \mathcal{D}$ and $\sum D_w(S_i, p) \geq \epsilon(N_{all} - N_{last})$.
***Update*:** Update a label of a node v_p in $\mathcal{D}$ to $(\sum D_w(S_i, p) + \omega f, \Delta, time_i)$ if $\exists v_p \in \mathcal{D}$ and $(\sum D_w(S_i, p) + \omega f + \Delta \geq \epsilon N_{all})$.
***Delete*:** Delete v_p and all of v_p's super nodes in $\mathcal{D}$ if $\exists v_p \in \mathcal{D}$ and $(\sum D_w(S_i, p) + \omega f + \Delta < \epsilon N_{all})$.

Note N_{all} means all of processed sequence data which include S_i and N_{last} means processed sequence data $N_{all} - wsize$ without S_i. Also let us note $\sum D_w(S_i, p)$ describes the summation of total frequencies $D_w(S, p)$ from $N_{last} + 1$ to N_{all}, that is, $\sum D_w(S_i, p) = \sum_{l=N_{last}+1}^{N_{all}} D_w(S_{i_l}, p)$ Also $time_{now}$ is used to obtain ω in this step which corresponds to a timestamp $time_i$ of S_i.

EXAMPLE 6 Now we update a node v_a in the DPL of a figure 1. We are given $\epsilon = 0.1$, $N_{all} = 3$, $time_{now} = 3$, $wsize = 1$, $\lambda = 0.98$, $\tau = 3$ and $\sum D_w(S, a) = 0.2$. Then $\omega = 0.96$ and a weighted frequency of p is $\sum D_w(S, p) + \omega f + \Delta = 0.48$. Also $\epsilon N_{all} = 0.3$ and we see that v_a's label is updated $(f : 0.48, \Delta : 0, time_{last} : 3)$. On the other hand, we assume that v_c is deleted from the DPL. In this case, we can see that it's supper patterns $v_{[a,c]}, v_{[b,c]}, v_{[a,b]c}$ are deleted as well. After operating nodes, we get the DPL as in a figure 2(left).

Now let us describe how to extract frequent patterns from DPL. Basically we apply *depth first search* algorithm to DPL. Note $time_{now}$ is used to obtain ω in this step.

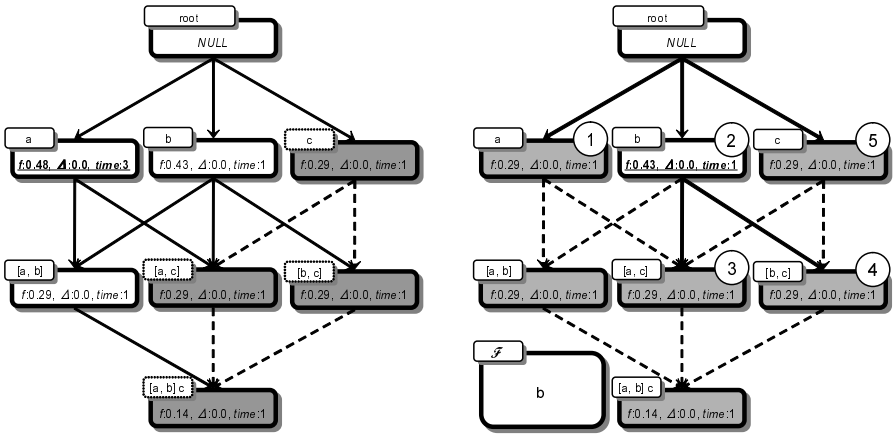

Fig. 2. Updating/Deleting, and Extracting Frequent Patterns from a DPL ($\sigma = 0.5$)

Input: DPL $\mathcal{D}$, minimum support σ, error bound ϵ
Output: a collection of frequent patterns $\mathcal{F}$
Algorithm:

1. Start with the root in DPL, find 1-frequent patterns which satisfies $(\sigma - \epsilon)N_{all} \leq \omega f$, push them to a stack K and add them to $\mathcal{F}$.
2. For each v popped from K, go to v' through an arc $e\langle v, v' \rangle$ in DPL. Push v' to K if we have never visited v' and add v' to $\mathcal{F}$ if v' satisfies $(\sigma - \epsilon)N_{all} \leq \omega f$. Goto (2).
3. Discard v' if we have already visited v', and goto (2)
4. After examining all the K elements, we output $\mathcal{F}$.

EXAMPLE 7 We extract all the frequent patterns which have frequencies $(\sigma - \epsilon)N_{all} = 0.4$ from a DPL in a figure 1. Now, we are given $\sigma = 0.5$, $\epsilon = 0.1$, $N_{all} = 1$, $time_{now} = 2$, $\lambda = 0.98$ and $\tau = 3$. According to the traversal as in a figure 2 (right), we obtain $\mathcal{F}$.

5 Experimental Results

5.1 Preliminaries

In this section we examine 4 kinds of experiments containing scalability, efficiency of construction of DPL and extracting frequent patterns from DPL, and space utilization for candidate patterns in DPL and the flexibility of time progress. We also analyze about the validity of our experimental results and deliberate about the weighting method in our method. In this experiment, we discuss only 1 level of disjunctive patterns where length are 5 or less such as `a, [ab], [ab]c, [abc]d, [abcd]e` . Also we assume disjunctive sub-pattern appears at most one time in any patterns.

Here we examine *Reuters-21578 text categorization test corpus* as a test collection. The collection consists of 21,578 news articles in 1987 provided by Reuters kept in time order. We have selected 2200 articles as 22 datasets which include 100 articles which have a common time stamp per dataset in time order and cleaned them by stemming and removal of stop-words[7]. We take "day" which

is relative time for the first dataset as a time unit of time stamps through our experiments. Here are some examples of the results. `shower continu week bahia cocoa zone allevi drought earli januari improv prospect temporao humid level restor...` Details of our experimental data are found in a table 1.

In our experiments, we take 4 kinds of the decay constants λ which are 1.00(without weighting), 1.00_7(valid for 1 week), 0.98(approx. $\omega = 0.5$ of 1 month past) and 0.91(approx. $\omega = 0.5$ of 1 week past).

dataset#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
time[day]	0	5	7	11	14	15	19	21	25	26	28	32	34	36	40	41	46	95	96	113	235	236
diff.[day]	-	5	2	4	3	1	4	2	4	1	2	4	2	2	4	1	5	49	1	17	122	1

Table 1. Details of experimental data

5.2 Results

In the first experiment, we give 0.005, 0.008 and 0.010 as error bound ϵ, 0.98 as a decay constant λ, and examine the construction time (in seconds) of a DPL every 50 articles in the experimental data. We show the scalability of our method in a figure 3(left).

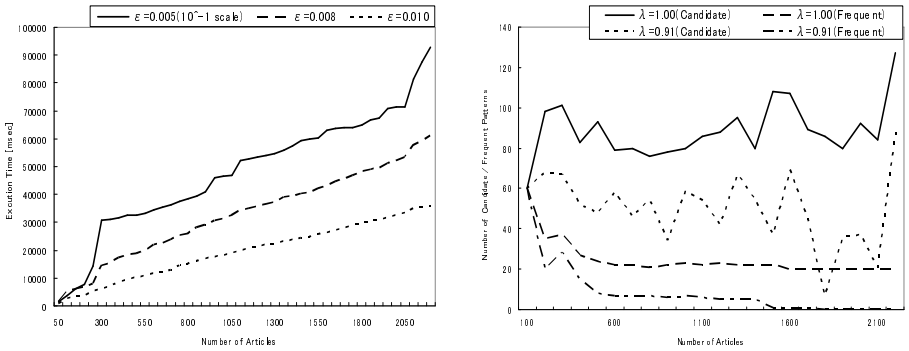

Fig. 3. scalability; the number of nodes in DPL and frequent paterns

In the experiment 2, we examine the frequency of the operations which are "Insert", "Update" and "Delete" and the execution time for the construction of DPL for each threshold, (1)error bound ϵ, (2)the number of simultaneous processed articles and (3)decay constant. In the case of (1), we give 0.005, 0.008 and 0.010 as ϵ with the number of simultaneous processed articles. In the case of (2), we give 25, 50 and 100 as the number of simultaneous processing articles with 0.008 as ϵ. In the case of (3), we give 0.008 as ϵ with 50 as the number of simultaneous processed articles. Both tables in a table 2 and a table 3(upper) illustrate the experimental results on each case.

A table 3 (bottom) contains the results of the experiment 3. We give $\epsilon = 0.005$, $\lambda = 0.98$, 100 as the number of simultaneous processed articles and give 0.005, 0.006, ...,0.010, 0.015, 0.020, 0.030 as minimum support σ and examine the execution time (in seconds) to extract all the frequent patterns from 300 articles in the experimental data.

In the experiment 4, we give $\epsilon = 0.005$, $\sigma = 0.010$ and 100 as the number of simultaneous processed articles and we examine the comparison of the number of frequent patterns and candidate patterns which mean nodes in a DPL with

λ	1.00			1.00_7			0.98			0.91		
ϵ	.005	.008	.01	.005	.008	.01	.005	.008	.01	.005	.008	.01
Insert	1301	461	258	1360	483	275	1388	487	281	1453	521	296
Update	1579	614	364	1368	530	316	1230	486	289	903	359	223
Delete	1125	400	223	1206	425	241	1233	429	247	1293	462	261
Time[sec]	1041.1	67.8	36.2	1054.3	66.8	36.9	929.1	61.2	36.1	738.6	56.5	35.7

λ	1.00	1.00_7	0.98	0.91	λ	1.00	1.00_7	0.98	0.91
Insert	461	483	487	521	Delete	400	425	429	462
Update	614	530	486	359	Time[sec]	67.8	66.8	61.2	56.5

Table 2. Constructing a DPL for ϵ (upper) and λ (bottom)

λ	1.00			1.00_7			0.98			0.91		
# of articles	25	50	100	25	50	100	25	50	100	25	50	100
Insert	2597	461	172	2618	483	187	2626	487	188	2643	521	203
Update	1450	614	266	1328	530	221	1210	486	196	1024	359	124
Delete	2494	400	134	2524	425	156	2532	429	158	2549	462	172
Time[sec]	224.7	67.8	48.3	239.1	66.8	50.7	208.9	61.2	44.3	196.4	56.5	39.4

σ	.005	.006	.007	.008	.009	.01	.015	.02	.03
Time[msec]	47	46	32	31	31	31	31	31	16
# of patterns	101	101	72	51	39	37	7	5	2

Table 3. Constructing a DPL on number of articles and Execution Time (sec.)

the cases of $\lambda = 1.00$ and $\lambda = 0.91$. Also we examine the number of frequent patterns which are extracted from constructed DPL by 300 articles every 7 days under same conditions. A figure 3(right) and a table 4 illustrate the results.

5.3 Discussion

Let us discuss what the results mean. In the experiment 1, the reader sees the scalability of our method. In our method, 2 times bigger ϵ causes 25.7 times longer execution time. The result as a whole becomes approximately linear increase for any ϵ. However, in a first part(0 to 300 articles) of the result, the efficiency is much worse by perpetual inserting nodes to a DPL. Also in a final part(2050 to 2200 articles) of the result, the efficiency is much worse as well. This is because our method gets the effect of time progress which becomes bigger with smaller ϵ.

Through experiment 2, the reader can see the efficiency of construction of DPL. In the case of (1), we see that 2 times bigger ϵ causes 28.8 times longer execution time and 4.74 times larger the number of all operations which are "Insert", "Update" and "Delete" when we give 1.00 as λ. Similarly we need 20.7 times longer execution time and 4.68 times bigger costs when we give 0.91 as λ. This is because our method generates more candidate patterns with smaller ϵ. In the case of (2), we see that 4 times bigger the number of simultaneous processed

Past days	0	7	14	21	28	35	42	49	56	63	70	Past days	0	7	14	21	28	35	42	49	56	63	70
$\lambda = 1.00$	37	37	37	37	37	37	37	37	37	37	37	$\lambda = 0.98$	37	31	20	18	8	7	7	6	5	4	2
$\lambda = 1.00_7$	35	0	0	0	0	0	0	0	0	0	0	$\lambda = 0.91$	28	7	2	0	0	0	0	0	0	0	0

Table 4. The number of frequent patterns for past days

articles causes 4.65 times shorter execution time and 11.3 times smaller the number of all operations when we give 1.00 as λ. Similarly we need 4.99 times shorter execution time and 12.5 times smaller costs when we give 0.91 as λ. Because our method can execute the operations which is like a batch processing for DPL by simultaneous processing of articles. In the case of (3), we see that smaller λ reduces the costs of construction, more specifically, the number of "Insert" and "Delete" is increased and "Update" is decreased. This is because our method updates the candidate patterns in DPL frequently for smaller λ.

Experiment 3 shows how efficiently our method works. Our method takes more bigger execution time when we give smaller minimum support σ. However we think that it isn't much worse. For example, 6 times bigger σ causes 2.9 times longer execution time. This is because our method scans the collection only once to extract frequent patterns.

In the experiment 4, we examine space utilization for candidate patterns in DPL and the flexibility of time progress.

As for space utilization for candidate patterns, our method gets worse efficiency for remaining candidate nodes in DPL with smaller λ, an average of 28% with 1.00 and 17% with 0.91. This is because, in our method, candidate patterns in DPL become infrequent quickly for bigger weight with smaller λ.

As for the flexibility of time progress, after the 1600 articles, we see that the number of candidate patterns and frequent patterns in DPL are reduced markedly when we give 0.91 as λ. Moreover, in a table 4, we can see that our method takes account of newer patterns in DPL with smaller λ, the number of extracted frequent patterns from DPL are 84% after 1 weak and 54% after 3 weeks with 0.98, 25% and 0% with 0.91 and both 0% with 1.00_7. From this results, we think that our method deletes old candidate patterns from DPL with given weight.

Finally, we analyze about the validity of our experimental results and deliberate about the weighting method in our method. In our experiments, we can see that smaller λ improves the execution efficiency of our method. In other words, we think that our method pays a consideration of that. Therefor we construct a DPL with 0.005 as ϵ and 100 as the number of simultaneous processed articles. After that we extract frequent patterns from a constructed DPL with 0.01 as σ, select dominant patterns which have frequency of 81% for all of extracted frequent patterns' frequency, and compare the coverage of dominant patterns for any λ. We give the coverage (percentage) of dominant patterns $Cover$ as $\frac{\#\ of\ extraction\ of\ patterns\ for\ any\ \lambda}{\#\ of\ extraction\ of\ patterns\ for\ \lambda = 1.00} \times 100$

A table 5(left) shows the coverage of dominant patterns for whole experimental data. As shown, smaller λ causes reduction of the coverage, 33.3% with 0.91. That is, we obtain efficiency of our method by reduction of the coverage. Therefore, we limit local appearance of frequent patterns by using the weighting method. That's reason why we can improve the execution efficiency of our method.

From a table 1, we can see that there are a lot of time interval from 1800 to 2200 of the experimental data, that is to say, the update interval of the patterns in DPL becomes longer. A table 5 (right) shows the coverage of dominant patterns for 1700 articles of the experimental data. By this table and a table 5 (left), we get 21.4% smaller coverage with 1.00_7 as λ. That is, we can extract frequent patterns of the whole data in relevant execution time by using the weighting method.

Cover[%]	said	mln	dlr	pct	year	billion	campani
$\lambda : 1.00$	100/100	100/100	100/100	100/100	100/100	100/100	100/100
$\lambda : 1.00_7$	77.3/100	77.3/100	77.3/100	77.3/100	77.3/100	100/100	77.3/100
$\lambda : 0.98$	90.9/100	86.4/100	77.3/100	77.3/100	72.7/94.1	40.0/40.0	77.3/100
$\lambda : 0.91$	77.3/100	63.6/82.4	63.6/82.4	45.5/58.8	18.2/23.5	6.7/6.7	18.2/23.5
Cover[%]	bank	share	ct	net	loss	sale	*Total*
$\lambda : 1.00$	100/100	100/100	100/100	100/100	100/100	100/100	*100/100*
$\lambda : 1.00_7$	61.5/100	77.3/100	77.3/100	77.3/100	100/100	100/100	*78.6/100*
$\lambda : 0.98$	0/0	72.7/94.1	77.3/100	50.0/64.7	0/0	33.3/33.3	*67.1/82.6*
$\lambda : 0.91$	0/0	18.2/23.5	45.5/58.8	13.6/17.6	0/0	33.3/33.3	*33.3/44.6*

Table 5. The coverage of dominant patterns with 2200/1700 articles

6 Conclusion

In this investigation we have proposed an online and single-pass algorithm by which we can extract current frequent disjunctive patterns by a weighting method for past events from a news stream. We have introduced a sophisticated structure, DPL, discussed how to construct DPL and given a decay function as a weighting method for time progress. Also, by experimental results, we have shown that our method is effective for extraction of disjunctive patterns and construction in real data.

References

1. Aggarwal, C.C.. and Yu, P.S.: Online Generation of Association Rules, ICDE, 1998, pp.402-411
2. Agrawal, R. and Srikant, R.: Fast Algorithm for Mining Association Rules, proc. VLDB, 1994, pp.487-499
3. Chen, Y., Dong, G., Han, J., Wah, B. W. and Wang, J.: Multi-dimensional regression analysis of time-series data streams, International Conference on Very Large Databases, 2002, pp.323-334
4. Goethals, B.: A Survey on Frequent Pattern Mining, Univ.of Helsinki, 2003
5. Ishikawa, Y. and Kitagawa, H.: An Improved Approach to the Clustering Method Based on Forgetting Factors, DBSJ Letters Vol.2 No.3, 2003, pp.53-56(in Japanese)
6. Jian, N. and Gruenwald, L.: Research Issues in Data Stream Association Rule Mining, SIGMOD Record 35-1, 2006, pp.14-19
7. Grossman,D. and Frieder,O.: Information Retrieval – Algorithms and Heuristics, Kluwer Academic Press, 1998
8. Loo, K.K., Tong, I. and Kao, B.: Online Algorithms for Mining Inter-stream Associations from Large Sensor Networks, Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2005, pp.143-149
9. Manku G. S. and Motwani R.: Approximate frequency counts over data streams, the 28th International Conference on Very Large Data Bases, 2002, pp.346-357
10. Nagao, M.: Natural Language Processing, Iwanami, 1996 (in Japanese)
11. Shimizu, K. and Miura, T.: Disjunctive Sequential Patterns on Single Data Sequence and its Anti-Monotonicity, International Conference on Machine Learning and Data Mining (MLDM), 2005, pp.376-383
12. Shimizu, K. and Miura, T.: Online Analysis for Disjunctive Sequential Patterns, ADBIS Workshop on Data Mining and Knowledge Discovery (ADMKD), 2006, pp.61-72
13. Takano,Y., Iwanuma, K. and Nabeshima, H.: A Frequency Measure of Sequential Patterns on a Single Very-Large Data Sequence and its Anti-Monotonicity, in Japanese, proc.*FIT*, 2004, pp.115-118 (in Japanese)
14. Uejima, H., Miura, T. and Shioya, I.: Estimating Timestamp From Incomplete News Corpus, proc.DEWS, 2005, 3C-o4(in Japanese)