INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 10(5):654-666, October, 2015.

Mining Periodic Traces of an Entity on Web

X. Huang, X. Wang, Y. Zhang, J. Zhao

Xinyan Huang
1. Shandong University
2. Shandong University of Finance and Economics
Num 1500, SunHua Road in High Tech Industrial Development Zone
Ji'nan, China
20063462@sdufe.edu.cn

Xinjun Wang*, Yan Zhang, Jinxin Zhao
Shandong University
Num 1500, SunHua Road in High Tech Industrial Development Zone
Ji'nan, China
wxj@sdu.edu.cn, zy@sdu.edu.cn, zjx@sdu.edu.cn
*Corresponding author: wxj@sdu.edu.cn

Abstract: A trace of an entity is a behavior trajectory of the entity. Periodicity is a frequent phenomenon for the traces of an entity. Finding periodic traces for an entity is essential to understanding the entity behaviors. However, mining periodic traces is of complexity procedure, involving the unfixed period of a trace, the existence of multiple periodic traces, the large-scale events of an entity and the complexity of the model to represent all the events. However, the existing methods can't offer the desirable efficiency for periodic traces mining. In this paper, Firstly, a graph model(an event relationship graph) is adopted to represent all the events about an entity, then a novel and efficient algorithm, TracesMining, is proposed to mine all the periodic traces. In our algorithm, firstly, the cluster analysis method is adopted according to the similarity of the activity attribute of an event and each cluster gets a different label, and secondly a novel method is proposed to mine all the Star patterns from the event relationship graph. Finally, an efficient method is proposed to merge all the Stars to get all the periodic traces. High efficiency is achieved by our algorithm through deviating from the existing edge-by-edge pattern-growth framework and reducing the heavy cost of the calculation of the support of a pattern and avoiding the production of lots of redundant patterns. In addition, our algorithm could mine all the large periodic traces and most small periodic traces. Extensive experimental studies on synthetic data sets demonstrate the effectiveness of our method.
Keywords: Event, Periodic Trace, Pattern.

1 Introduction

An event is something that happens at some specific time. Nowadays, due to the popularity of the internet, an increasing number of events about an entity are reported on web every day. However, since these events are usual scattered and redundant, meaningful information can not be extracted by people, such as a behavior trajectory of an entity which is named a trace.

Periodicity is one of the most common phenomena that these traces show, such as the launch of new products, the product promotion and so on, which are named periodic traces by us. In addition, a periodic trace can be loosely defined as the repetitive events series with several kinds of event relationship between them, such as causal, part of and following relationship etc. Periodic traces can provide an insightful and concise explanation over the long development history of one entity, which are very valuable in the prediction of future events' happening of an entity.

Unfortunately, mining periodic traces from an entity's long and noisy history data is a challenge, which includes the following major issues:

Firstly, the period of a trace are usually unfixed. However, in previous work, Li Z [8,10] studied the frequent periodic behaviors for moving objects with the period fixed. Unfortunately, the fixed period is unfit for the periodic trace. For example, the period of the launch of new products is usually varying with the changes of the competitive environment or others factors.

Secondly, the model to represent so large events set is a complex issue. Considering the big scale of the events and the relatively complex relationship between them, a big graph is employed, in which the events are represented as vertices and the events relationship (such as, causal relationship, following relationship etc) are represented as edges. And the big graph is named the event relationship graph of the entity.

At the same times, both the structure of periodic traces and the mining of periodic traces become more complex. The problem of mining periodic traces is transformed into mining frequent subgraph from a big graph. For nowadays, the general methods of frequent subgraph mining from a single graph are based on edge-by-edge (i.e., incremental) pattern-growth [12], which are more fit for a small graph than our big graph with several thousand vertices.

In this paper, the problem of periodic traces mining is solved through the following steps:

Firstly, the cluster analysis method is adopted according to the similarity of the activity attribute of an event and each cluster gets a different label. Secondly, a novel method is proposed to mine all the vertice-edge-vertice patterns from the event relationship graph firstly and then all the Star patterns are mined based on the preceding patterns. Finally, an efficient method is proposed to merge all the Stars to get all the periodic traces of the entity.

In our paper, we address an important problem of mining periodic traces. In our algorithm, high efficiency is achieved through deviating from the existing edge-by-edge pattern-growth framework. At the same time, the heavy cost of the calculation of the support of a pattern is reduced and the production of lots of redundant patterns can be avoided. In addition, our algorithm could mine all the large periodic traces and most small periodic traces.

The rest of the paper is organized as follows. In Section 2, some related work is discussed. Section 3 gives the problem formulation. Section 4 and section 5 provides an outline of our algorithm to discover the periodic traces. We report our experimental results in Section 6, conclude our study in Section 7.

2 Related Work

At present, some studies [20] about how to organize events and events relationship have been investigated. However, less studies aim to mine some meaningful information from the events data of entities.

An event relationship migration graph was proposed by Zhaoman Zhong in [1] to organize events, in which a typical event is defined as a behavior sequence or a series of state changes.

A concept of event network is proposed by Zongtian Liu [3], which is used to organized the events and the relationship between them in an article.

Christopher C. Yang [4] introduces the concept of event evolution graph in which events are organized by the time sequence, which can help to efficiently browse the whole of event evolution process.

Heng Ji [5] proposes to identify the "centroid entities" which are frequently involved in events and then link the events involving the same centroid entity along a time line. However, no more studies are done based on the work ahead.

Zhenhui Li proposed an concept of periodic behavior in [8,10], which is mined from the spatiotemporal data over a long histoy and propose a two-stage algorithm, Periodica, to detect

the periods in complex movements and to mine periodic movement behaviors.

Similar to method in literature [3], all events of an entity is firstly linked according to three types of relationships between them. But the difference is that their goal is to create an event ontology but not to mine the meaningful information on the event ontology. However, our main goal is to organize the events of an entity for a long history and to mine the meaningful periodic traces, so much more vertices are involved than [3].

The model of a periodic trace adopted by us is a graph structure which is different from surprising periodic patterns [13] and the periodic behavior in [8,10] which is characterized as a probabilistic model.

SUBDUE [15] is probably the most well-known algorithm for mining frequent subgraphs in a single graph, however, it tends to mine small patterns with high frequency.The SpiderMine algorithm in [7] is proposed to mine top-K largest frequent patterns from a single massive network. SUBDUE and SpiderMine are both approximate algorithms, which aren't fit for our problem to mine all the periodic traces.Although MoSS is an algorithm to mine the complete pattern set, it suffers from a significantly runtime Complexity issue as the input graph size grows [7,17]. So we propose an efficient method fit for our problem. Our algorithm is to mine the complete pattern set, which achieves its efficiency through reducing the heavy cost of calculation of the support of a pattern and avoiding the production of lots of redundant patterns.

3 Problem Formulation

All the following work is based on the assumption that the event relationship graph $G = (V, E)$ has been established already.

The mining of periodic traces is to mine the periodic events series with three kinds of event relationship from the events data of an entity. With the help of the event relationship graph, the main task of periodic traces mining is now transformed to mine the frequent subgraph from the event relationship graph.

We formally define the notions related with the mining of periodic traces as follows:

Definition 1 (Event)**.** An event is something that happens at some specific time, and often some specific place, which is usual a phrase or sentence in the web pages including an activity which is the main word most clearly expressing an event occurrence. Similar to [2], in our paper, the event of an entity is defined as a model E with five attributes, described as follows:

$$E\langle \text{subject}, \text{activity}, \{\text{object}\}, \text{time}, \{\text{location}\}\rangle.$$

Among them, subject, activity and time elements are required.

Definition 2 (Event Relationship)**.** The dependent relationship from an event to another is named an event relationship. There are three kinds of relationships that we considered in this paper, which are causal relationship [9], part of relationship and following [19] relationship.

For these three kinds of relationships, we consider an event relationship as a logical dependency between two events.

If an event e_1 is a component of an event e_2, then there is a part of relationship between e_2 and e_1.

Besides, if the occurrence of an event e_2 always depends on the occurrence of an event e_1, then there must be a causal relationship or a following relationship from e_1 to e_2. If e_1 surely leads to the occurrence of e_2, then there is a causal relationship between e_1 and e_2. The rest are following relationships in which e_2 always occurs after e_1.

Definition 3 (Event Relationship Graph)**.** Event relationship graph is to link all the events of an entity according to the relationship between them. An event relationship graph is denoted as a directed graph $G = (V, E)$, where V are the vertices representing events and E are the directed edges representing event relationships. a fragment of an event relationship graph is shown in Fig 1, in which R_c represents causal relationship, R_f represents part of relationship and R_p represents part of relationship.

Definition 4 (Trace)**.** A trace is denoted as a directed graph $G^* = (V^*, E^*)$, which represent a behavior trajectory of the entity.

Definition 5 (Periodic Trace)**.** A periodic trace is also denoted as a directed graph $G\prime = (V\prime, E\prime)$ composed of periodicly happening events as the vertices $V\prime$ and periodicly happening event relationships as the directed edges $E\prime$ between vertices $V\prime$, where $V\prime \in V$, $E\prime \in E$. And the support of $G\prime$ in G is beyond σ, σ is the minimum support that we set.

The problem of mining periodic traces is to mine all the periodic traces $G\prime = (V\prime, E\prime)$ from an event relationship graph G, and the occurrence of $G\prime$ in G is beyond σ.

We will solve the problem in the following sections.

4 Find All the Event Classes

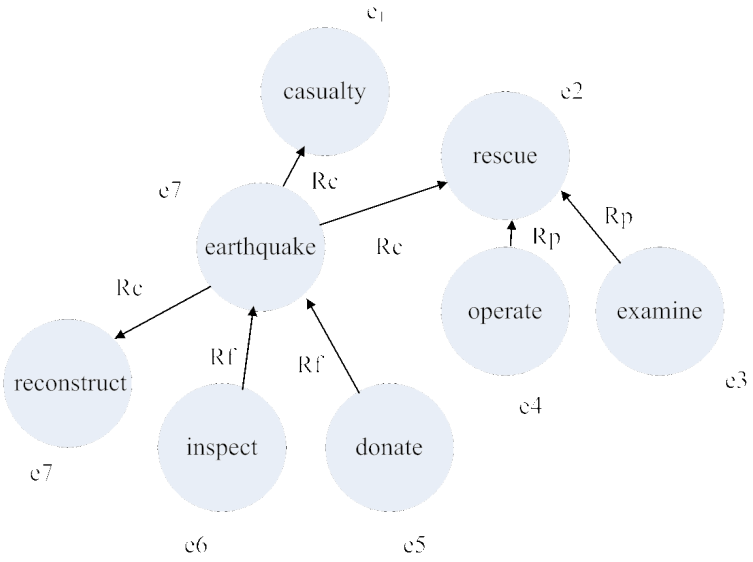

Figure 1: A fragment of an event relationship graph

A fragment of an event relationship graph is shown in Fig 1, in which all the events is represented by it's activity attribute. In this section, our main goal is to find all the event classes $EC = \{ec_1, ec_2, \ldots, ec_D\}$ from all the events, ec_i is an event class in which all the events has the similar activity and is represented like $ec_i = \{e_2, e_6, e_j, \ldots, e_n\}$.

In this section, a clustering method is employed. With the help of wordnet, all the events are clustered into groups according to semantic similarity of activity attribute of them, because the activity attribute could mostly express an event, hence the same kind of events get together and get an identical label, such as A, B etc. We name the events with the similar activity as an event class. For example, a launch event class, a promotion event class etc.

Then the event relationship graph is turned to a labeled directed graph, in which there are three kind of edges labeled as 1, 2, 3 etc.

5 Results and Discussion

In this section, we will describe the periodic trace mining algorithm, TracesMining, which could mine all the periodic traces from the event relationship graph.

The main algorithm of mining periodic traces is shown in Algorithm 1.

Algorithm 1 The periodic traces mining algorithm

Require: input all the event classes $EC = \{ec_1, ec_2, \ldots, ec_D\}$, the event relationship graph G of an entity, support threshold σ.

Ensure: all the periodic traces S.

1: Initialize $S \leftarrow \Phi$;
2: $T\prime \leftarrow \text{MinsMining}(G, EC, \sigma)$;
3: `/* mine all the minimum patterns- vertice-edge-vertice;*/`
4: $S\prime \leftarrow \text{StarsMining}(G, T\prime, \sigma)$;
5: `/* mine all the patterns-Stars based on` $T\prime$`;*/`
6: $S \leftarrow \text{StarsMerging}(S\prime, \sigma)$;
7: `/* Merge all Stars whenever possible*/`
8: Return S;

5.1 Mining All the Frequent Patterns of Vertice-Edge-Vertice

In the MinsMining(G, EC, σ) algorithm, for every two event classes,we should get all the patterns of vertice-edge-vertice and all their corresponding instances in G, here an instance of an pattern is just a physical occurrence of the pattern.

As is shown in Fig 2, $A \xrightarrow{R_f} B$ is a vertice-edge-vertice pattern, the frequency that the physical events in one event class A has the event relationship R_f with the physical events in another B in the event relationship graph G must be beyond the threshold σ, R_f is one of the three relationships we defined ahead.

The generation of a pattern $A \xrightarrow{R_f} B$ and it's instances is illustrated in Fig 2, in which e_i refers to an physical event, ec_i refers to an event class the label of which is B and σ is set to 2.

5.2 Mining All the Patterns of Stars

In the algorithm 1, the StarsMining$(G, T\prime, \sigma)$ algorithm is to generate all the patterns of Stars and their corresponding instances based on all the veritice-edge-vertice patterns produced ahead.

Just as Fig 3 illustrated, the so called Star is just a pattern the number of occurrence of which in the event relationship graph is beyond σ and in which a vertice serves as the center, 1 is the radius, the edge represent one of three kinds of event relationships and the direction of edge is optional which can point to or be away from the center. In Fig 3, the label of e_1 and e_2 is B, the corresponding label of e_7 and e_8 is D etc.

Then we propose an efficient method to mine all the stars based on all the vertice-edge-vertice patterns, as shown in Algorithm 2.

In the algorithm 2, in line 5, the function find(C, L_i) is to find all the vertice-edge-vertice pattern that have a vertice with the label L_i. In line 10, the function Count$(T[i], T[j])$ is to

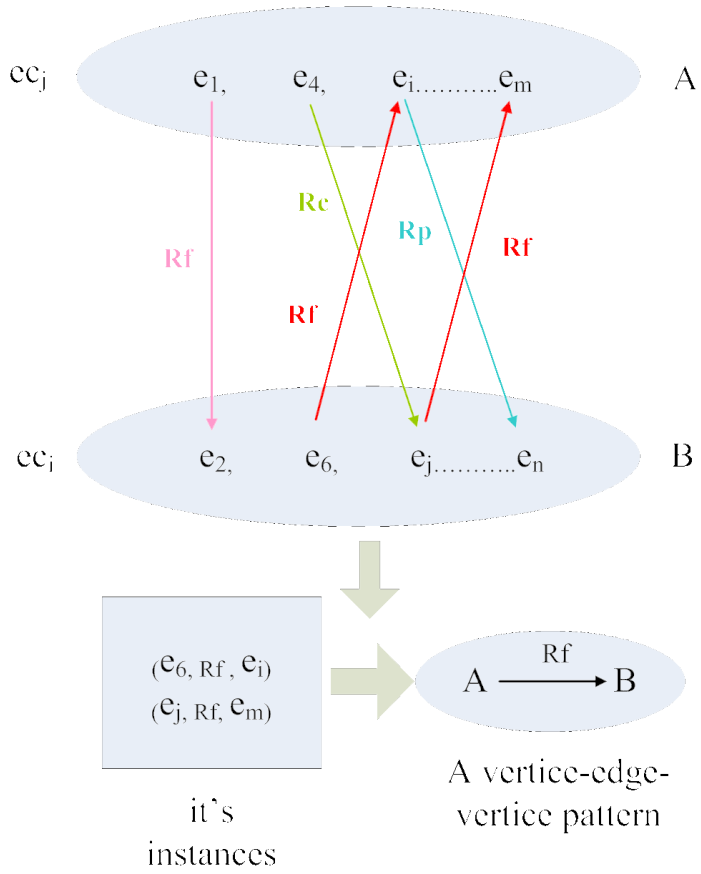

Figure 2: Generation of a vertice-edge-vertice pattern and it's instances

count the number of the same physical events that the label Li in T[i] and T[j] stands for and judge whether the number is beyond σ. In line 13, the function Merge$(T[i], T[j], T\prime)$ is to merge $T[i]$ and $T[j]$ if the number is beyond σ. In line 14, the function Check$(T, S\prime)$ is to put all the patterns which has never been merged into $S\prime$. Ultimately, the $S\prime$ stores all the stars centered by Li, for example, the stars centered by label A are named like Star(A1), Star(A2) etc according to the different physical events set that the label A stands for. In the algorithm 2, we shall finally find all the Star patterns.

In this paper, Stars could help efficiently mine the periodic traces due to the following reasons:

1. Stars reduce the heavy cost of the calculation of support of a pattern.
   Judging whether two patterns can be merged, what we only need to do is to determine which Star the merging bases on and further confirm the number of the same physical events that the center label of the Star stand for. The number is the support of the merged pattern, which could greatly reduce the heavy cost of the calculation of support of a pattern.

2. Stars reduce the complexity of merging.
   The inherent character of Stars with one label as the center makes the merging of pattern convenient. To merge a star, a pattern is just to judge whether one of the borders of the pattern can merge with the star. And the merging makes the patterns grow by Stars

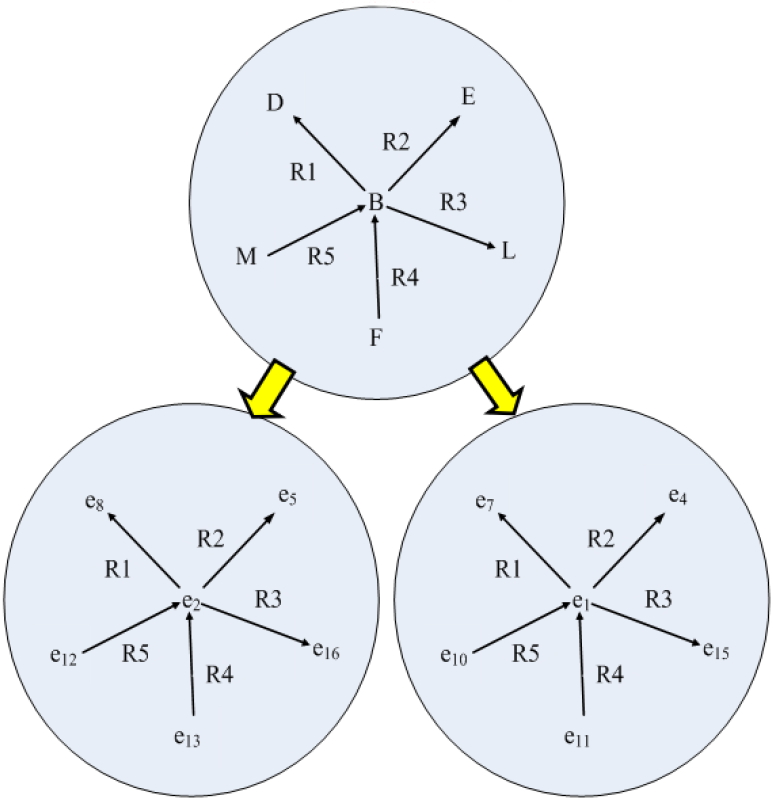

Figure 3: A Star pattern and it's instances

deviating from the existing edge-by-edge pattern-growth way [14], which greatly improve the efficiency of merging.

The last and most important step of generating the periodic traces is to merge the Stars, details is shown in 5.3.

5.3 Generating Periodic Traces

To discover all the periodic traces, the last step is to merge all the Stars. Then an effecient method StarsMerging() is proposed to merge all the Stars that can be merged until no more frequent patterns can be found. Although the number of underlying periodic traces is usually unknown, the algorithm StarsMerging() could at the same time determine the optimal number of periodic traces while generating all the periodic traces, details shown in. Algorithm 3.

S is implemented as a queue into which all the Stars are put. Star(S) (line 3) points to the current Star to be dealt with. p.border (line 6) points to all the boundaries of the pattern p. The function compare(b, s) (line 10) is to count the number of the same physical events that the vertice b and the center of s stand for.

At line 4–13, each current Star as far as possible merges the Stars in S through checking it's borders one by one until no stars can be merged to. The function cleanUp(S, σ) (line 21) is employed to clean all the stars that has been merged and the number of remainder physical events of the center vertice is not beyond σ from S.

Compared with the algorithm in literature [7], our algorithm is more effective and efficient. In Our algorithm, a pattern with 4 Stars just need 3 merging without additional patterns generated, which does not generated any redundant patterns and does not need any additional merging,

Algorithm 2 The StarsMining algorithm

Require: input all the vertice-edge-vertice pattern $C\{C_1, C_2, C_3, \ldots, C_i, \ldots, C_e\}$, all the instances of C, all the labels $L\{L_1, L_2, L_3, \ldots, L_i, \ldots, L_f\}$ appears in C, support threshold σ.

Ensure: all the Stars S.

```
1: Initialize S ← Φ;
2: for i = 1 to f do
3:   S′ ← Φ;
4:   flag = false;
5:   T ← find(C, L_i)
     /*find all the vertice-edge-vertice patterns that have a vertice with the
     label Li*/
6:   repeat
7:     T′ ← Φ;
8:     for i = 0 to T.length do
9:       for j = i + 1 to T.length do
10:        n ← Count(T[i], T[j]);
           /*to count the number of the same physical events that the label Li
           in T[i] and T[j] stands for*/
11:        if n ≥ σ then
12:          flag = ture;
13:          Merge(T[i], T[j], T; );   /*add the merged pattern to T′*/
14:        end if
15:      end for
16:    end for
17:    Check(T, S′)
       /*all the pattern in which the number of the physical events that label
       Li stand for is beyond σ and which hasn't been merged are put into S′*/
18:    T = T′
19:  until flag  /* the merger is completed*/
20:  S′ = S′ ∪ T;
21:  S = S ∪ S′;
22: end for
23: return S
```

while the algorithm in [7] averagely needs 5 additional patterns generated and needs 6 merging. With the number of Stars growing, more obvious advantages are shown in our algorithm.

More details of the comparison with [7] are shown in Section 6.

6 Experiments

In this section, we performed extensive experiments to evaluate the performance of our algorithm of mining the periodic traces on synthetic data. All experiments were done on a 2.8GHz Intel Pentium IV PC with 1GB main memory, operating system Windows XP. Our algorithm is implemented in Python 2.7.

The mining of periodic traces is experimented on a single graph which is generated by the Erdős - Rényi random network model. The Erdős - Rényi model is a well-known model to generate random graphs. Using the G(n, p) function, our synthetic single graph is constructed.

Algorithm 3 StarsMerging

Require: All the Stars queue S, support threshold σ.
Ensure: The periodic traces $S\prime$.

```
1: S' ← Φ;
2: while S.length > 0 do
3:   p = Star(S);  /* select one star from S*/
4:   while true do
5:     flag = false;  /*a flag whether merge happened*/
6:     border ← p.border;  /*check the borders of p one by one*/
7:     for b in border do
8:       ss = lookupS(b);  /*find all the stars of which the label of the center
         vertice is b*/
9:       for s in ss do
10:        n = compare(b, s)
           /*count the same physical events that the vertice b and the center
           vertice of s stand for*/
11:        if n > σ then
12:          p.merge(s);
13:          flag = true;
14:        end if
15:      end for
16:    end for
17:    if !flag then
18:      break;
19:    end if
20:  end while
21:  S' = S' ∪ p;  /*put these patterns that don't grow to S'*/
22:  cleanUp(S, σ);
23: end while
24: return S'
```

However, the graph generated by the Erdős - Rényi model is an undirected graph, which is different from our event relationship graph which is a directed graph. So some changes made based on the Erdős - Rényi model is that 0.5 probability is adopted to decide the direction of the edge. Besides, the labels of vertices and the labels of edges are distributed randomly under the condition that labels of any two adjacent vertices can't be identical.

In the experiment, in order to calculate the recall and precision ratio, a set of large patterns as well as a set of small patterns are injected into the graph. Our goal is to find all the large frequent patterns and all the small patterns from the big graph. Nowadays despite lots of studies in graph mining, few algorithms are capable of the mining task in a big single graph due to the exponentially high combinatorial complexity and support computation. And since Spidermine [7] has been compared with the other algorithms (SEuS [16] (version 1.0), MoSS [17] (version 5.3) and ORIGAMI [18]) and shows a tremendous advantage in efficiency and effectiveness, we just compare our algorithm with Spidermine and the well-known SUBDUE [15] (version 5.2.2), which is a classic approximate algorithm on a single graph.

Firstly, we generate 4 different data sets (labeled Data 1 to 4) with varied parameter settings referring to [7]. The description of the data sets is given in Fig 4. The details of the parameters is given as follows. $|V|$ is the number of vertices. l_V is the number of vertice labels and le is

Data	$\|V\|$	l_V	l_e	d	m	$\|V_L\|$	e_L	n	$\|V_S\|$	e_S
1	400	70	3	2	5	30	2	5	3	2
2	600	130	3	4	5	30	2	20	3	2
3	1000	250	3	4	5	30	2	5	3	20
4	1500	350	3	4	10	30	2	10	3	2

Figure 4: Data Sets

the number of edge labels. d is the average degree. $|V_L|$ (or $|V_S|$) is the number of vertices of each injected large (resp. small) pattern. m (or n) is the number of large (resp. small) patterns injected. e_L (or e_S) is the number of instances of each large (resp. small) pattern injected.

We implement the SpiderMine [7] according to it's algorithms in python. But a little changes are made to SpiderMine, because SpiderMine is designed for undirected graph without labels for it's edges while our graph is directed with labeled edges. At same times, SpiderMine is to mine top K large patterns from a big graph. Here we set $K = 5$, Dmax $= 4$.

Figs 5 to 8 show the distribution of patterns mined by SpiderMine, SUBDUE and our algorithm TracesMining for different data sets in Fig 4. The minimum support threshold is set to 2.

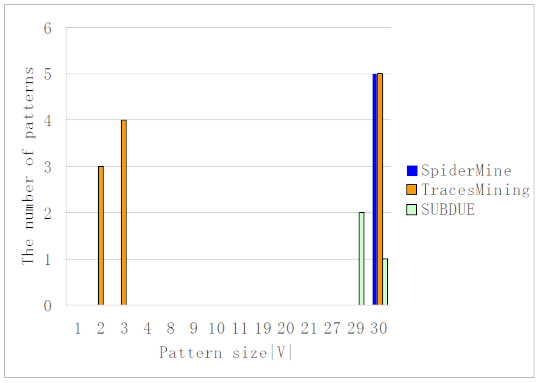

Figure 5: Data 1

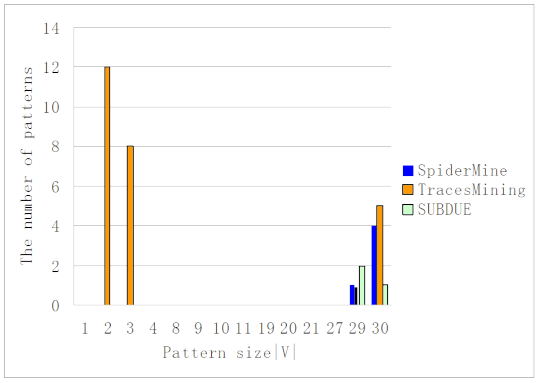

Figure 6: Data 2

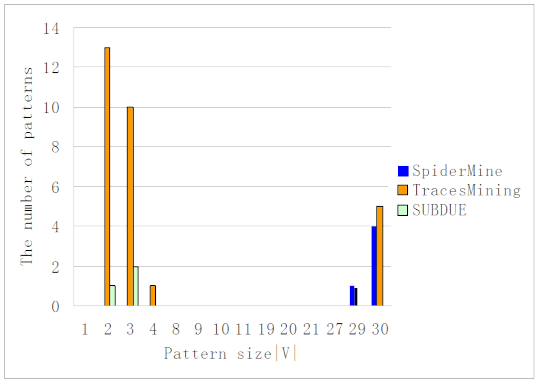

Figure 7: Data 3

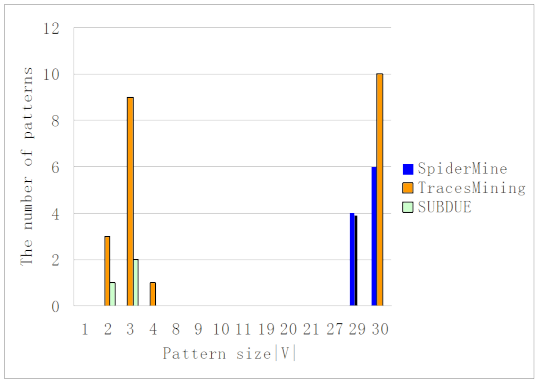

Figure 8: Data 4

In Figs 5, 6, 7 and 8, we can see that compared with SpiderMine() and SUBDUE, our algorithm, TracesMining could mine all the large periodic traces and most small periodic traces injected into the graph.While SpiderMine() focus on the large patterns in four different data sets and SUBDUE tends to mine smaller patterns .

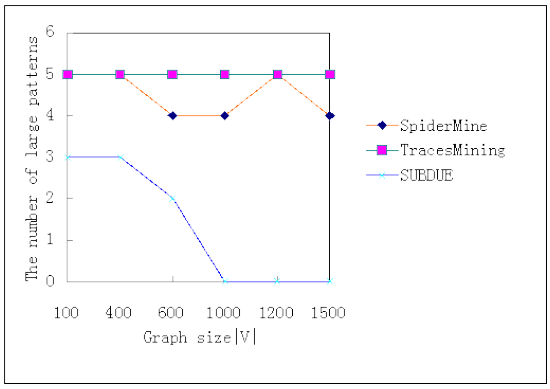

Figure 9: Large Pattern

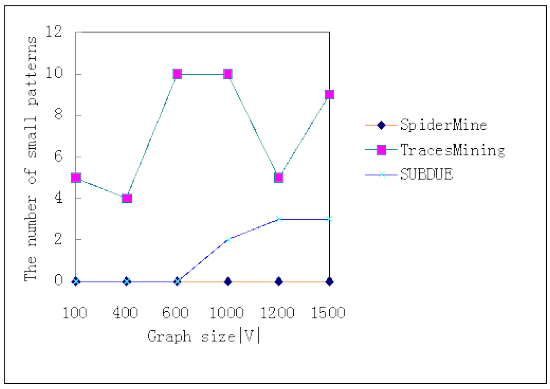

Figure 10: Small pattern

Run Time (seconds)		
Data	TracesMining	SpiderMine
1	0.686	0.704
2	2.248	3.951
3	10.089	13.041
4	13.190	16.272

Figure 11: Run Time

In Figs 9 and 10, TracesMining shows a tremendous advantage compared with SpiderMine and SUBDUE in both large and small patterns mining. SpiderMine is good at mining large patterns and SUBDUE tends to mine smaller patterns with the growing of graph, which fully illustrate that SpiderMine() and SUBDUE don't suit our task to find all the periodic traces.

Since SpiderMine has compare the run time with SUBDUE in [7], the runtime comparison of SpiderMine and TracesMining is shown in Fig 9 on the three data sets. We can see that in runtime, our algorithm has a clear advantage over SpiderMine.

7 Conclusion

In this paper, we address an important and difficult problem: Mining Periodic Traces of an entity on web. We propose a novel and efficient framework to solve the aforementioned problem. Our algorithm achieves its efficiency through deviating from the existing edge-by-edge pattern-growth framework and reducing the heavy cost of the calculation of the support of a pattern and avoiding the production of lots of redundant patterns. In addition, our algorithm could mine all the large periodic traces and most small periodic traces. Experiments demonstrate the efficiency as well as scalability of our algorithm.

Acknowledgement

The first author acknowledges the support by the Natural Science Foundation of China (No. 61303005, No. 61303089, No. 61202151, No. 61103117).

Bibliography

[1] Z. Zhong, Z. Liu, Z. Wen(2009), The Model of Event Relation Representation, *Journal Of Chinese Information Processing*, 23(6): 56–60.

[2] Z. Zhong, C. Li (2013), Web News Oriented Event Multi-Elements Retrieval, *Journal of Software*, 24(10):2366–2378.

[3] Z. Liu, M. Huang, W. Zhou(2009), Research on Event-oriented Ontology Model, *Computer Science*, 36(11): 191–195.

[4] C.C. Yang, X. Shi, C.P. Wei (2009), Discovering event evolution graphs from news corpora, *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(4): 850–863.

[5] H. Ji, R. Grishman, Z. Chen et al (2009), Cross-document Event Extraction and Tracking: Task, Evaluation, Techniques and Challenges, *RANLP*, 166–172.

[6] W. Liu, D. Wang, W. Xu et al (2012), A Sub-topic Partition Method based on Event Network, *The Seventh International Conference on Internet and Web Applications and Services*, 194–199.

[7] F. Zhu, Q. Qu, D. Lo et al(2011), Mining top-k large structural patterns in a massive network, *Proc. of the VLDB Endowment*, 4(11): 807–818.

[8] Z. Li, J. Han, B. Ding et al (2012), Mining periodic behaviors of object movements for animal and biological sustainability studies, *Data Mining and Knowledge Discovery*, 24(2): 355–386.

[9] S. Bethard, J.H. Martin (2008), Learning semantic links from a corpus of parallel temporal and causal relations, *Proc. ACL-HLT*, 177–180.

[10] Z. Li, J. Han, M. Ji et al (2011), Movemine: Mining moving object data for discovery of animal movement patterns, *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4): 37.

[11] Z. Guan, X. Yan, L.M. Kaplan (2012), Measuring two-event structural correlations on graphs, *Proceedings of the VLDB Endowment*, 5(11): 1400–1411.

[12] L. Gao, G.-M. Qin Gui, X.-F. Zhou (2008), An Overview of Algorithms for Mining Frequent Patterns in Graph Data, *Acta Electronica Sinica*, 36(8): 1603–1609.

[13] J. Yang, W. Wang, S.Y. Philip (2008), Mining surprising periodic patterns, *Data Mining and Knowledge Discovery*, 9(2): 189–216.

[14] M. Wörlein, T. Meinl, I. Fischer et al(2005), *A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston*, Springer Berlin Heidelberg.

[15] L.B. Holder, D.J. Cook, S. Djoko (1994), Substucture Discovery in the SUBDUE System, *KDD workshop*, 169–180.

[16] S. Ghazizadeh, S.S. Chawathe (2002), SEuS: Structure extraction using summaries, *Discovery science*, Springer Berlin Heidelberg, 71–85.

[17] M. Fiedler, C. Borgelt (2007), Support Computation for Mining Frequent Subgraphs in a Single Graph, *MLG*.

[18] M. Al Hasan, V. Chaoji, S. Salem et al(2007), Origami: Mining representative orthogonal graph patterns, *ICDM*, 153–162.

[19] Z. Li, F. Wu, M.C. Crofoot (2013), Mining Following Relationships in Movement Data, *ICDM*, 458–467.

[20] I.C. Resceanu, G.C. Călugăru, C.F. Resceanu et al(2012), Cooperative Robot Structures Modeled After Whale Behavior and Social Structure, *International Journal of Computers Communications & Control*, 7(5): 945–956.