Project: Periodic Subgraph Mining in Dynamic Networks

Jingfang Liu

December 9, 2012

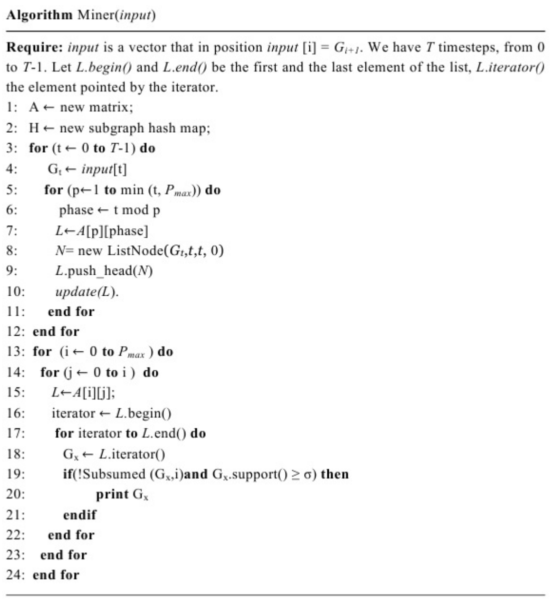
Problem Description and data structure

Given a dynamic networks $G =< G_0, ..., G_{T-1} >$ over uniquely labeled vertices for a set V and a minimum support threshold $\sigma \geq 3$, the periodic subgraph iming problem is to find the maximal subgraphs that recur at regular intervals in G. A maximal periodic subgraph means that it cannot be enriched by adding edges or expanding temporal span without losing support.

The data structure used in mining procedure includes:

- a hash table from a subgraph to a list of descriptors
- a list containing the subgraph at period p with phase m
- a matrix containing the list heads for different periods and phases.

The algorithm is like this

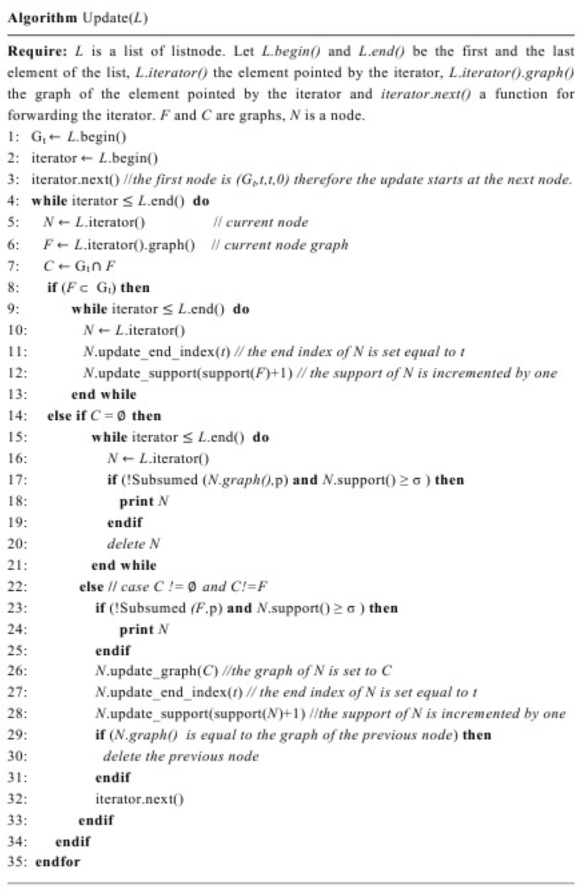


Algorithm Miner(*input*)

Require: *input* is a vector that in position *input* [i] = G_{i+1}. We have T timesteps, from 0 to T-1. Let *L.begin()* and *L.end()* be the first and the last element of the list, *L.iterator()* the element pointed by the iterator.

```
A ← new matrix;
H ← new subgraph hash map;
for (t ← 0 to T-1) do
    G_t ← input[t]
    for (p←1 to min (t, P_max)) do
        phase ← t mod p
        L←A[p][phase]
        N= new ListNode(G_t,t,t, 0)
        L.push_head(N)
        update(L).
    end for
end for
for (i ← 0 to P_max ) do
  for (j ← 0 to i ) do
    L←A[i][j];
    iterator ← L.begin()
    for iterator to L.end() do
      G_x ← L.iterator()
      if(!Subsumed (G_x,i)and G_x.support() ≥ σ) then
            print G_x
      endif
    end for
  end for
end for
```

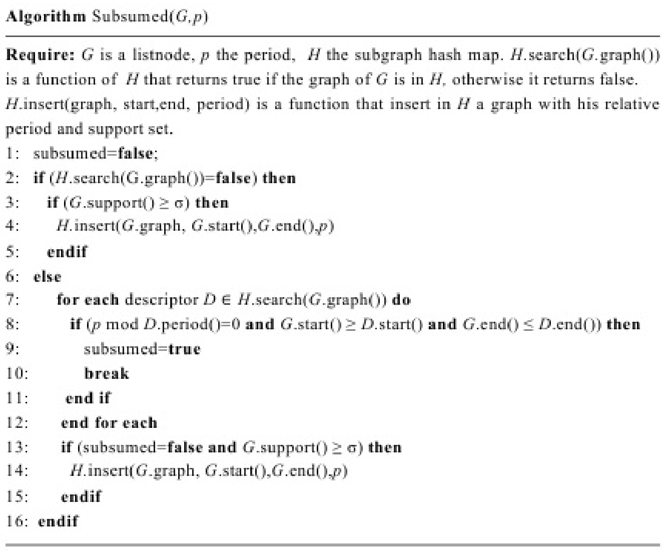
where the two functions Update and Subsumed are like this



Algorithm Update(L)

Require: L is a list of listnode. Let *L.begin()* and *L.end()* be the first and the last element of the list, *L.iterator()* the element pointed by the iterator, *L.iterator().graph()* the graph of the element pointed by the iterator and *iterator.next()* a function for forwarding the iterator. F and C are graphs, N is a node.

```
G_t ← L.begin()
iterator ← L.begin()
iterator.next() //the first node is (G_t,t,t,0) therefore the update starts at the next node.
while iterator ≤ L.end() do
  N ← L.iterator()              // current node
  F ← L.iterator().graph()   // current node graph
  C ← G_t ∩ F
  if (F ⊂ G_t) then
      while iterator ≤ L.end() do
         N ← L.iterator()
         N.update_end_index(t) // the end index of N is set equal to t
         N.update_support(support(F)+1) // the support of N is incremented by one
      end while
  else if C = ∅ then
          while iterator ≤ L.end() do
             N ← L.iterator()
             if (!Subsumed (N.graph(),p) and N.support() ≥ σ ) then
                print N
             endif
             delete N
          end while
        else // case C != ∅ and C!=F
           if (!Subsumed (F,p) and N.support() ≥ σ ) then
                print N
           endif
           N.update_graph(C) //the graph of N is set to C
           N.update_end_index(t) // the end index of N is set equal to t
           N.update_support(support(N)+1) //the support of N is incremented by one
           if (N.graph() is equal to the graph of the previous node) then
             delete the previous node
           endif
           iterator.next()
        endif
    endif
endfor
```



Algorithm Subsumed(G,p)

Require: G is a listnode, p the period, H the subgraph hash map. H.search(G.graph()) is a function of H that returns true if the graph of G is in H, otherwise it returns false. H.insert(graph, start,end, period) is a function that insert in H a graph with his relative period and support set.

```
1:  subsumed=false;
2:  if (H.search(G.graph())=false) then
3:     if (G.support() ≥ σ) then
4:        H.insert(G.graph, G.start(),G.end(),p)
5:     endif
6:  else
7:     for each descriptor D ∈ H.search(G.graph()) do
8:        if (p mod D.period()=0 and G.start() ≥ D.start() and G.end() ≤ D.end()) then
9:           subsumed=true
10:          break
11:       end if
12:    end for each
13:    if (subsumed=false and G.support() ≥ σ) then
14:       H.insert(G.graph, G.start(),G.end(),p)
15:    endif
16: endif
```

The algorithms are photocopied from [1]. I made several modifications in my code and these modifications include

- When $C = F$, check the graph of the previous node and the graph of current node. If they're equal, delete the previous node
- When Gt is an empty graph, update the []A[p][phase] by a sepcial update function which is similar to the case when C is empty without adding a node at the head of the list.

Here are some details for the data structures used in my code.

Hash Table

I creat hash table as an array of lists. In each lists, the graph and corresponding descriptors are stored.

For each graph this is a list of integers. I used list¡int¿ for each graph. Given a graph G, first derive a integer h from the list of integers representing the graph by following code

```
long int p = 193939;
    for(list<int>::iterator it=G.begin(); it!=G.end(); it++)
            p *= ( (*it) +1 ) % 193939;
    unsigned int h = (unsigned int) p%22111;
```

Then this h is the position for graph G in the hash table. When inserting and seaching are performed, the list at position h will be inserted or looked up.

listnode

I create a struct listnode containing five elemtns:

- a list of integers representing the graph
- start time
- end time
- support of the graph
- a pointer to the next listnode

Based on this struct, I also write a function to create a new listnode which returns a pointer to the listnode just created.

List

The list is based on listnode struct. I also write several functions to update the list, delete a particular listnode from a list, delete the whole list. For the functions like deleting a particular listnode, since I use pointers, the deletion could be done in constant time.

2D Array

I used 2D array to store the head the lists for each period and each phase. At the beginning of the mining, this 2D array is initialized so that each element is a NULL pointer. Then as reading new graph, corresponding lists get updated, the element in this 2D array will change correspondingly.

1 Complexity

1.1 time complexity

A projection $\pi_{p,m}$ of G is a subsequence of graphs $\pi_{p,m} =< G_{1+m}, G_{1+m+p}, G_{1+m+2p}, ... >$ where p is the period and m is the phase. For each period p, there are p different phases. The support of every projection is $\lceil (T-m)/p \rceil$. The maximum number of the listnode is also $\lceil (T-m)/p \rceil$. For every timestep t and for every list, in the worst case, the algorithm needs to calculate maximal common subgraph between Gt and every listnode in the list. So the number of maximal common subgraph is

$$\sum_{p=1}^{Pmax} \sum_{m=0}^{p} \sum_{j=0}^{\lceil (T-m)/p \rceil} j = O(T^2 \ln(Pmax)) \qquad (1)$$

Since the maximal common subgraph can be computed in $O(V+E)$ time, the total complexity of mining algorithm without subsumption is $O((V+E)T^2 \ln(Pmax)$. Since $Pmax = 40$, the total time complexity is $O((V+E)T^2)$.

For the algorithm containing subsumption, the hash table insert and hash table search only takes constant time, so it doesn't increase the time complexity, which is also $O((V+E)T^2)$.

1.2 space complexity

For each time step t, one listnode is add to a proper phase for each period. Thus for each period p, the number of listnode is increased by 1. So for every period p, the number of listnodes is $O(T)$ in the worst case. In every listnode, the subgraph is stored, the size is $V+E$ in the worst case. So the total space complexity is $O((V+E)TPmax)$. In the code, $Pmax = 40$, so the space complexity is $O((V+E)T)$.

2 Results

I implement the algorithm with subsumption.

The input file are in the source code Miner_.cpp, to use another dataset, just need to change the filename in the Miner function in the source code. Then compile and run. compile the code:

```
g++ Miner_new.cpp -o Miner
```

Run the code:

```
./Miner
```

The mining results are stored in the file "results.txt" and the statistics are stored in the file "results_stat.txt "

Testing the algorithm on the data "enron-small", "enron-large" and "enron-dated.itemset" with $Pamx = 10, 40$. The results files and statistics in the corresponding files like "results_small_stat_Pmax_10". My results match with the results provided quite well, here is the first page of my statistics of data "enron-small"

```
0-3=5
0-4=1
100-3=5
100-4=1
200-3=64
200-4=8
200-5=4
200-7=1
300-3=115
300-4=26
300-5=8
300-6=1
400-3=356
```

```
400-4=174
400-5=65
400-6=26
400-7=2
400-8=2
400-9=1
500-3=322
500-4=141
500-5=54
500-6=11
500-7=6
500-8=5
500-9=5
500-10=2
500-11=3
500-12=2
500-14=1
500-15=2
500-16=1
500-17=1
500-18=1
500-20=1
500-22=1
500-26=1
500-27=1
500-28=1
```

Compared with the provided statistics, it's the same. So the algorithm works correctly.

Reference

[1] Barbares Manuel. Periodic subgraph mining in dynamic networks. *undergradate thesis*