EIA Algorithm

August 27, 2016

1 Problems in Traditional Cycle detection

There are several problems in the literature of traditional cycle detection, and one of the major topics is the detection of "inner cycle". Suppose P_k is a path with distance k. An inner cycle was defined as an cycle C_d with distance d which is included in P_k where $d < k$. From Fig.1(a), we can observe that a 3-distance cycle $C \rightarrow D \rightarrow E \rightarrow C$ was included in P_6 where $P_6 = A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow C \rightarrow F$. Traditional method utilizing matrix computing or other detection algorithm did not take inner cycle into consideration. The extension or application based on these inner cycles may result in a waste of memory and produce additional computational burden.

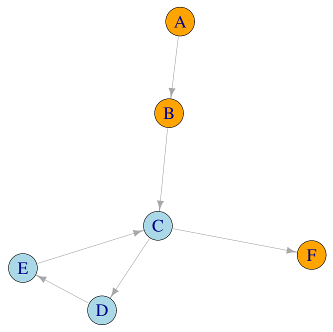
The other problem "repeated cycle". A cycle might be recorded repeatedly since that each cycle could be represented in different ways by selecting different nodes as the "head". Consider a k-distance cycle $C_k = (P_1 \rightarrow P_2 \rightarrow ... \rightarrow P_{k+1})$. Define the node set $P = \{P_1, P_2, ..., P_{k+1}\}$. C_k could be represented as a cycle started from any nodes in set P. That is, $C_k$also could be represented as $(P_{k,2} \rightarrow P_{k,3} \rightarrow ... \rightarrow P_{k+1} \rightarrow P_1)$. In general, a k-distance cycle C_k has k different representations. We take a 4-distance cycle C_4 for an example, the structure was presented in Fig.1(b). Different representations was classified in Table.1. The deletion of repeated cycle could reduce great amount of memory and improve computing efficiency.

2 Data Preprocessing

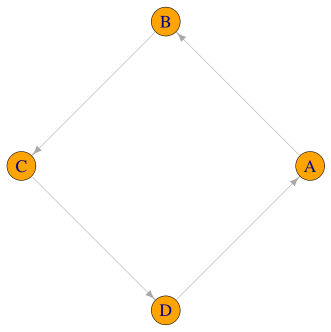
While analyzing large scale network, redundant data might slow down the computational speed. Thus, we have to process the data before applying the algorithm.

Table 1: Different Representations of C_4 in Fig.1(b)

A → B → C → D → A
B → C → D → A → B
C → D → A → B → C
D → A → B → C → A



(a) Example of an Inner Cycle



(b) Repeated Cycle

Figure 1: Two Major Problems in Traditional Cycle detection

2.1 Remove the isolate nodes

In the first step of data preprocessing, we have to delete the isolate nodes among the directed network. The node could be a head of a cycle only when it is a receiver and deliver simultaneously. As a result, we selected this nodes to keep in the directed network and deleted others. In our experience, when the connecting probability is 80%, the amount of the nodes in the directed network could be reduced by 10% to 15%. We take the trading network in Fig.2 for an example. The red nodes should be removed due to they are not both a seller and buyer.

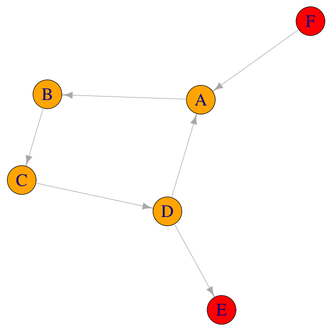


Figure 2: Removing the isolate points from directed network

2.2 Strongly Connected Component

In the mathematical theory of directed graphs, a graph is said to be strongly connected if every vertex is reachable from every other vertex. The strongly connected components of an arbitrary directed graph form a partition into subgraphs that are themselves strongly connected. It is possible to test the strong connectivity of a graph, or to find its strongly connected components, in linear time.

Call two nodes u and v of a directed graph $G = (V; E)$ connected if there is a path from u to v, and one from v to u. As such, it partitions V into disjoint sets, called the strongly connected components of the graph. If we shrink each of these strongly connected components down to a single node, and draw an edge between two of them if there is an edge from some node in the first to some node in the second, the resulting directed graph has to be a directed acyclic graph (dag) that is to say, it can have no cycles. The reason is simple: A cycle containing several strongly connected components would merge them all to a single strongly connected component. We can restate this observation as follows: Every directed graph is a dag of its strongly connected components. From the Fig.3, the network could be split into two parts.

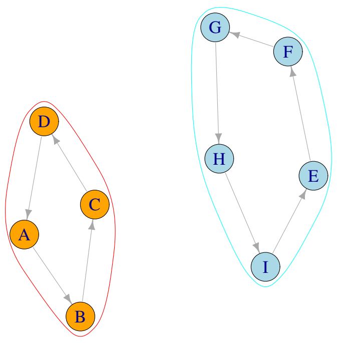


Figure 3: Split the network by strongly connected component

3 Algorithm

We defined a directed labeled graph as $G = (V, E)$, where V is the set of vertices with ordered labele $V == \{v_1, v_2, \ldots, v_m\}$ and E is the set of directed edges $E = \{e_1, e_2, \ldots, e_m\}$.

Notice that G could contain parallel edges in opposite directions but cannot have self-loops. Also, isolated vertices mentioned in last chapter should be removed from the graph due to it is impossible for them to become a member of cycle. A simulated graph G with 4 vertices and 7 edges was shown in the figure.4.

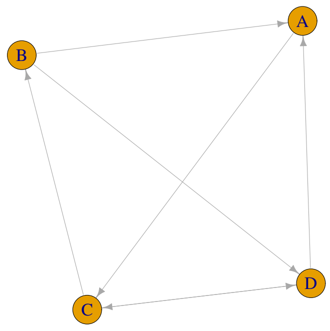


Figure 4: The simulation netowrk G

Definition

In general, this algorithm contains three types of sets. The first one is the set of cycle with distance k, defined as C_k.

Another is the set of "ascending order path" with distance k which is defined as $P_k(I)$. $P_k(I)$ contains $p_k(I)$ where in each $p_k(I)$, the leader's label is smaller than that of the end.

The other is the set of "descending order path" with distance k which is defined as $P_k(D)$. $P_k(D)$ contains $p_k(D)$ where in every $p_k(D)$, the leader's label is smaller than that of the end.

Definition

In addition, two major operators were used to operate two different paths; the first one is the "merge" operator $\bigotimes$ which is used to construct a new path when one path's head and end is same as the end and head of another path, respectively. Another operator is the "extend" operator $\bigoplus$ which is used to build a path if one path's end is the head of another path. How these two operators work were illustrated in the following example. Suppose there are three paths, $p_1 = (D \to C), p_2 = (D \to E \to A), p_3 = (A \to B \to D)$.

$$p_3 \bigotimes p_2 = (A \to B \to D \to E \to A), \text{which is a length 4 cycle}$$

$$p_3 \bigoplus p_1 = (A \to B \to D \to C), \text{which is an extended length 3 path.}$$

We now introduced the algorithm with an application on the simulated network G. All edges E are were shown in Table.2 and was classified to $P_1(D)$ and $P_1(I)$. The algorithm first apply $p_1(I) \bigotimes p_1(D)$ then $C_2 = (C \to D \to C)$ was generated.

Next, we extend the path $p_1(I)$ to length 2 by operating $p_1(I) \bigoplus E$ and obtain $P_2(I) = \{(A \to C \to B), (A \to C \to D)\}$. Notice that each $p_k(I)$

Algorithm 1 EIA algorithm

```
procedure EIA($G = (V, E)$)
    Initially group the edge set $E$ into two disjointed sets $P_1(I)$ and $P_1(D)$,
        and $E = P_1(I) \bigcup P_1(D)$.
    In general, this algorithm contains three phases with each given distance.
    for $k = 1$ to $K$ do
        I. Find Cycles by $k \bigotimes k$:
            Detect all length possible $2k$ cycles by $P_k(I) \bigotimes P_k(D)$.

        II. Extend Paths of Ascending Order Path Set and Find Cycles by $k + 1 \bigotimes k$:
            Extend paths by $P_{k+1}(I) = P_k(I) \bigoplus E$.
            Detect all length possible $2k + 1$ cycles by $P_{k+1}(I) \bigotimes P_k(D)$.
        III. Extend Paths of Descending Order Path Set:
            Extend paths by $P_{k+1}(D) = P_k(D) \bigoplus E$.

    end for
    return The set of cycles $C$.
  ▷ After running $K$ times, we can get all possible cycles with lengths from 2
        to $2K + 1$.
end procedure
```

Table 2: All edges E of the simulated network G

Head		End
A	$\rightarrow$	C
B	$\rightarrow$	A
B	$\rightarrow$	D
C	$\rightarrow$	B
C	$\rightarrow$	D
D	$\rightarrow$	A
D	$\rightarrow$	C

could be contained in $P_k(I)$ only when its end is larger then head. Also, the paths containing inner cycles should be removed as well. After obtaining $P_2(I)$, we apply the $\bigotimes$ for length 2 cycle set. Thusly, C_3 could be generated from $p_2(I) \bigotimes p_1(D)$ where $C_2 = \{(A \to C \to B \to A), (A \to C \to D \to A)\}$. During the operation, inner cycles should also be eliminated.

We now extend the descending order path $p_1(D)$ to length 2. In this step $P_2(D) = \{(C \to B \to A), (C \to D \to A),), (B \to D \to A)\}$ was generated from $E \bigoplus p_1(D)$. Due to $P_2(D)$ should follows descending order, the head of the every path $p_2(D)$ in $P_2(D)$ should larger then the end otherwise it should be eliminated. Also, inner cycles should be removed from the set in this step.

Further, we take $P_2(D) and P_2(I)$ as input and start from step I again to obtain $C_4, P_3(I), C_5, P_3(D)$. The iteration continues until our reaching our selected length cycles or one of the set $P_k(I) or P_k(D)$ is a null set.