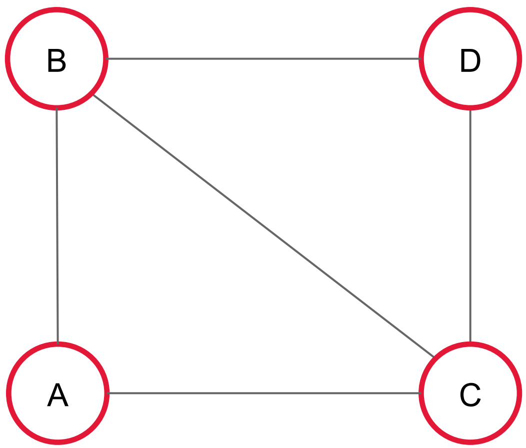
Efficient Mining of Active Components in a Network of Time Series

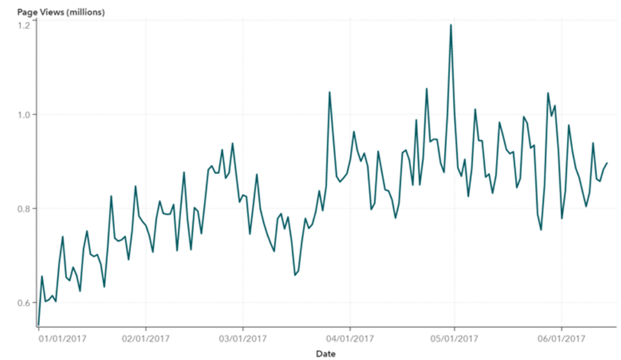
M.Sc. Thesis of Mahta Shafieesabet

York University, Toronto, Canada

Introduction

Networks and Time Series are Ubiquitous

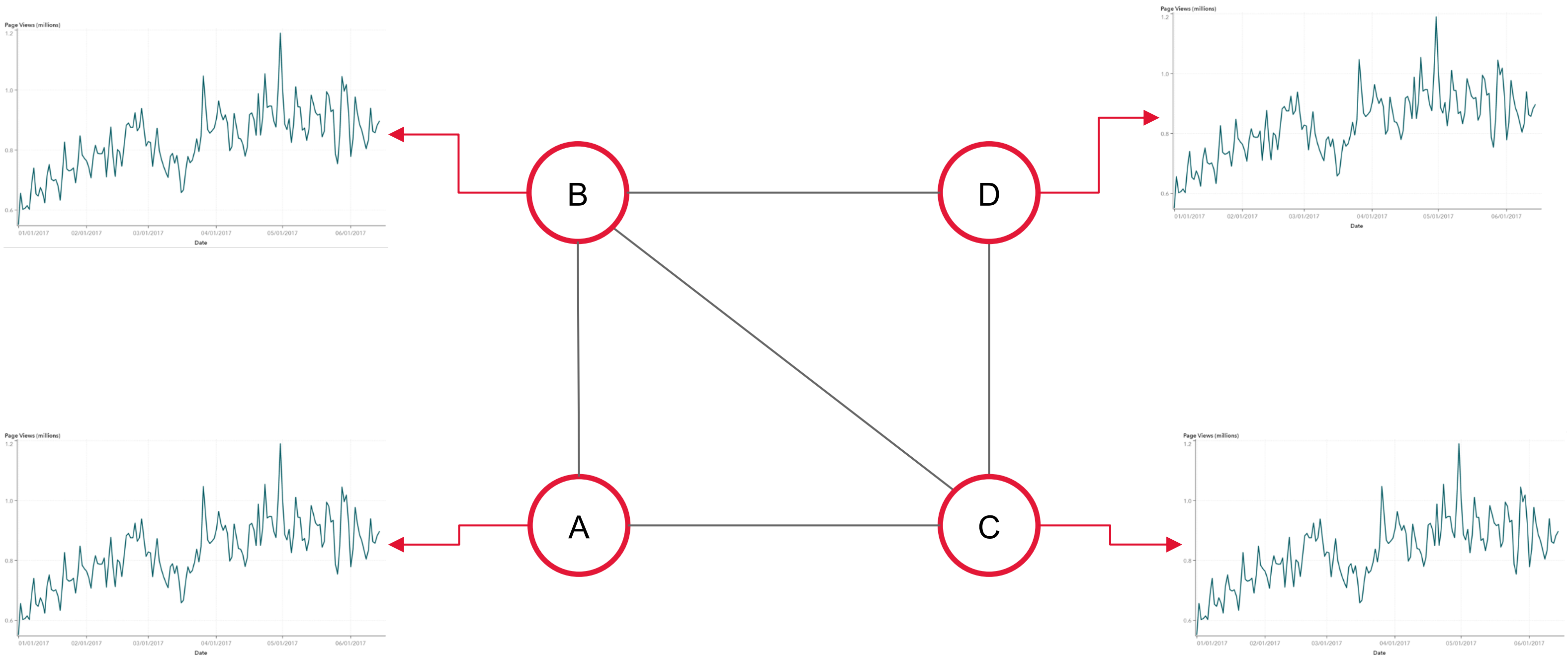




networks

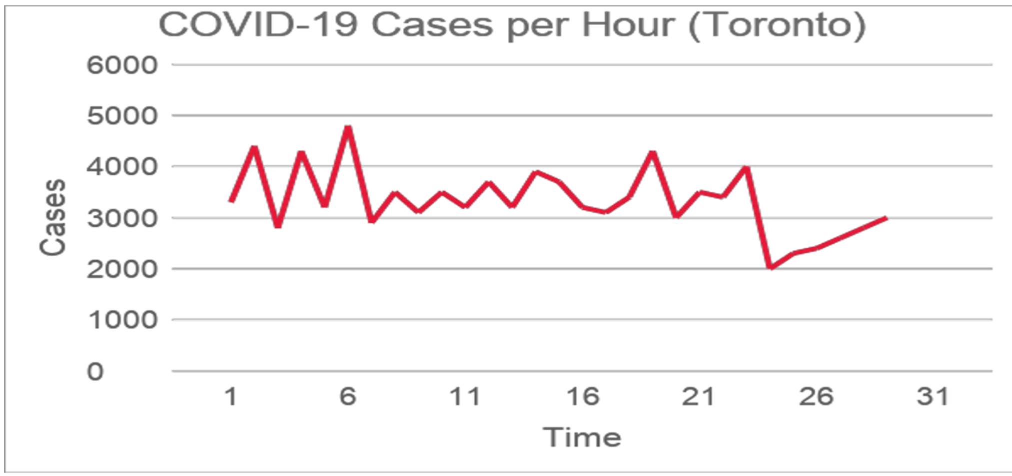
time series

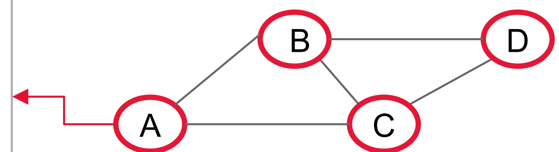
Network of Time Series

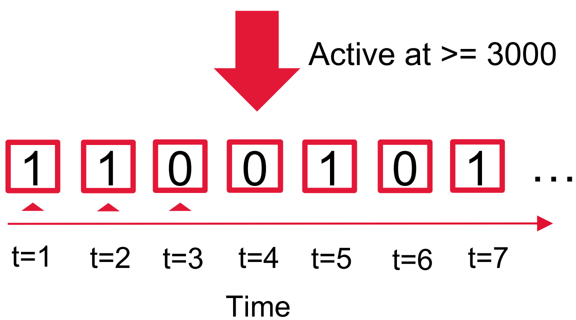


static topology

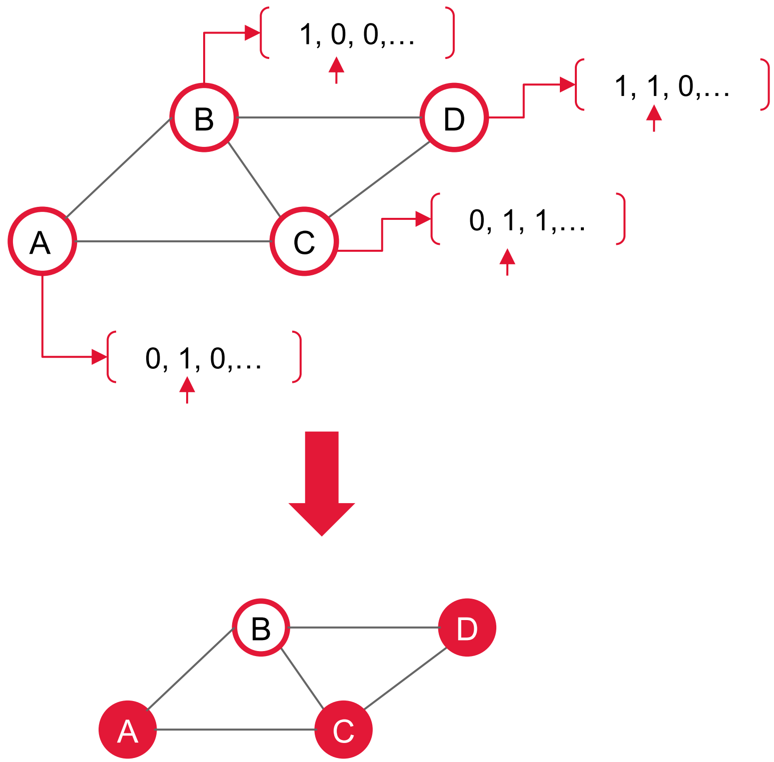
Binary Time Series





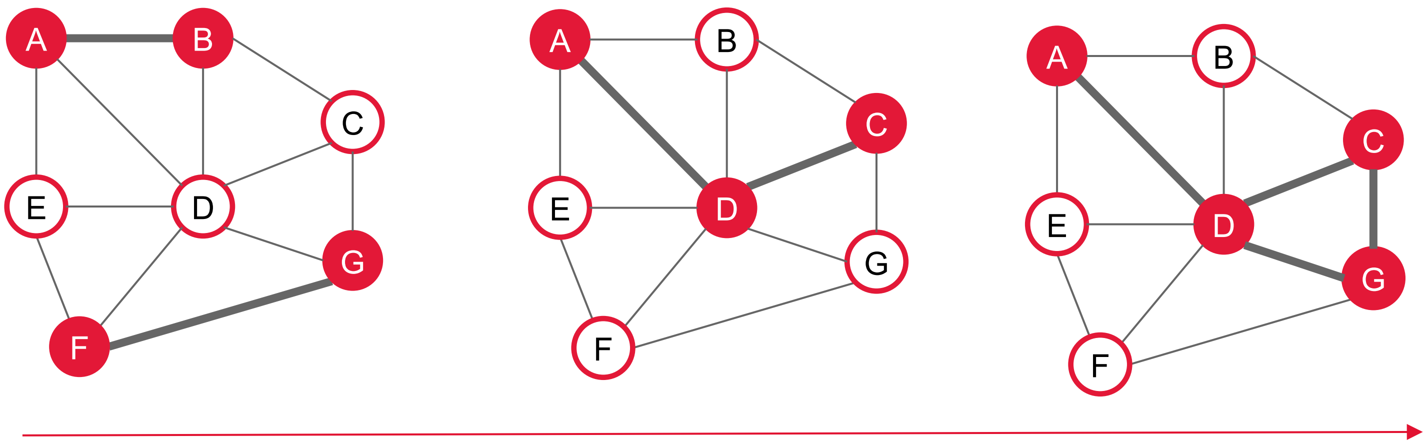


Active Nodes



Nodes at t = 2

Active Components



t = 1 t = 2 t = 3

{ {AB}, {FG} } {{ADC}} {{ ADCG}}

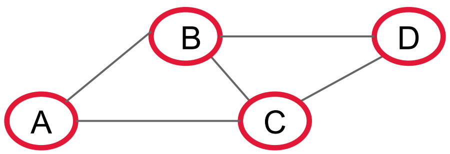
Problem Statement

Problem Statement

Given:

- A graph, G
- An $n \times h$ matrix X representing the time series, where X_{it} is the status of node i at time t

We want to compute C_t, the set of active components at time step t , for $t = 1, 2, \ldots, h$



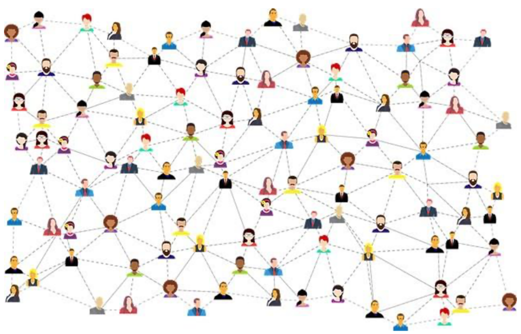
Time Steps

Vertices

	1	2	3	4	5
A	1	1	1	1	0
B	1	1	0	0	0
C	0	1	1	1	0
D	0	1	1	1	0
E	0	1	1	1	1
F	0	0	0	1	1

Why Do We Care?

Applications



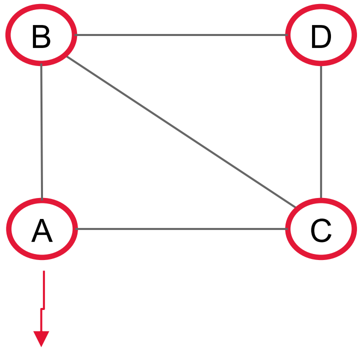
Social networks

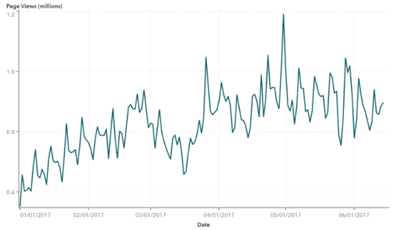
Technological networks

Biological networks

Road Networks

Road Network



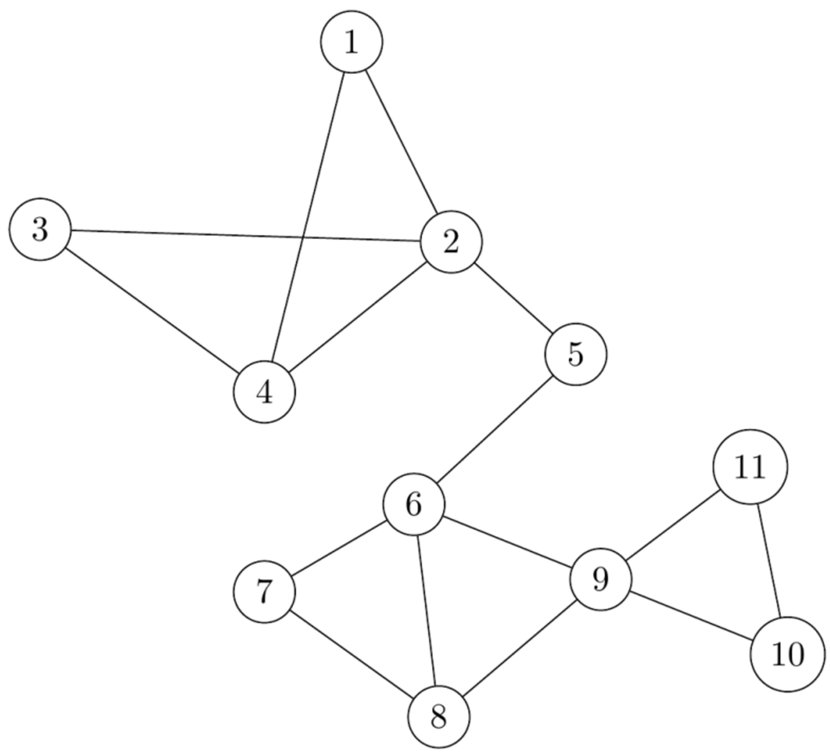


Background

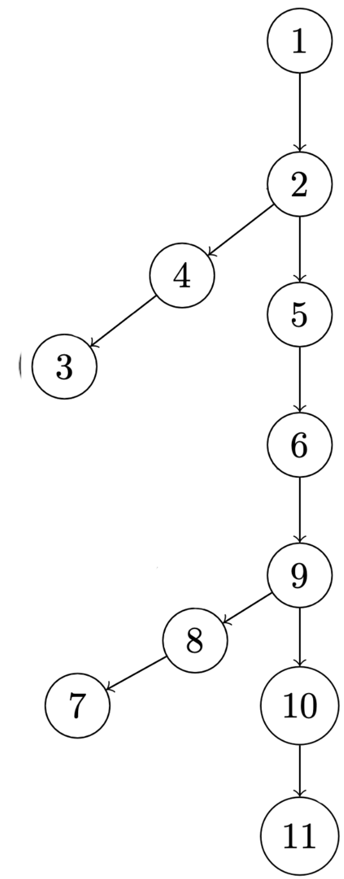
Graph Theory Background

- DFS-tree
- Heavy-Light Decomposition & Shallow Tree
- DFS-tree Enumeration

DFS-tree



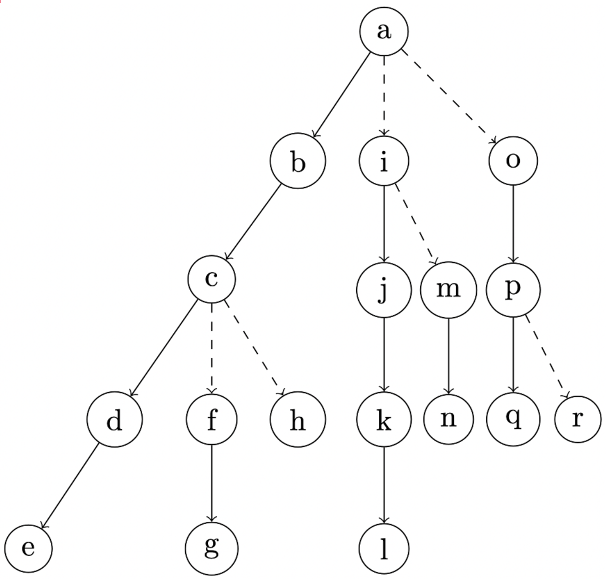
graph G



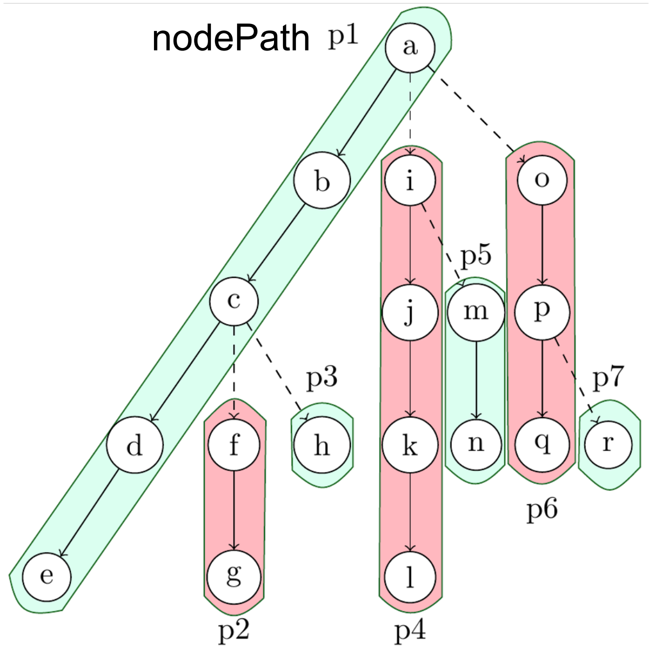
none-tree edges

DFS-tree of G

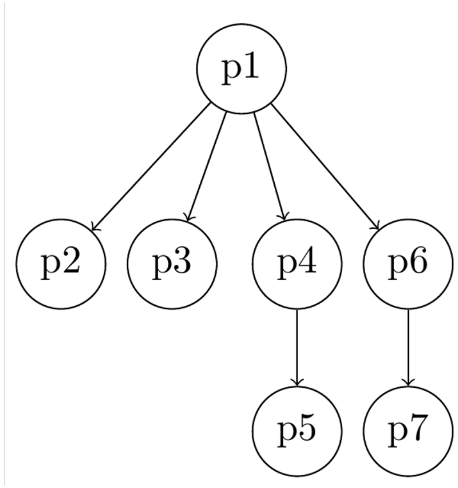
Heavy Light Decomposition & Shallow Tree



DFS-tree

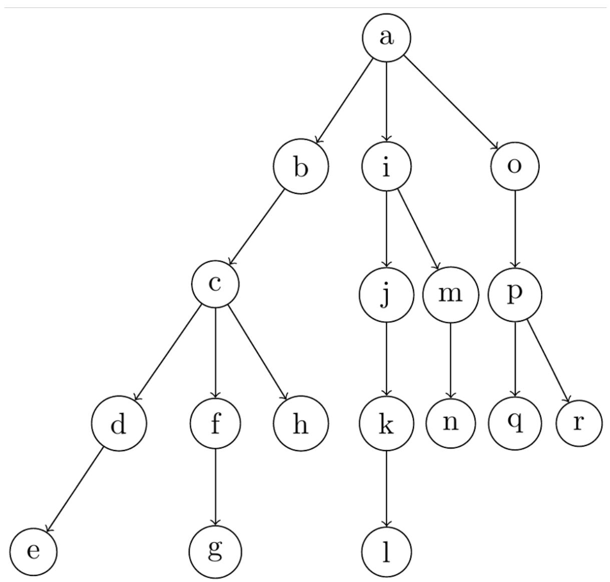


heavy light decomposition



shallow tree

DFS-tree Enumeration (dfs-id)

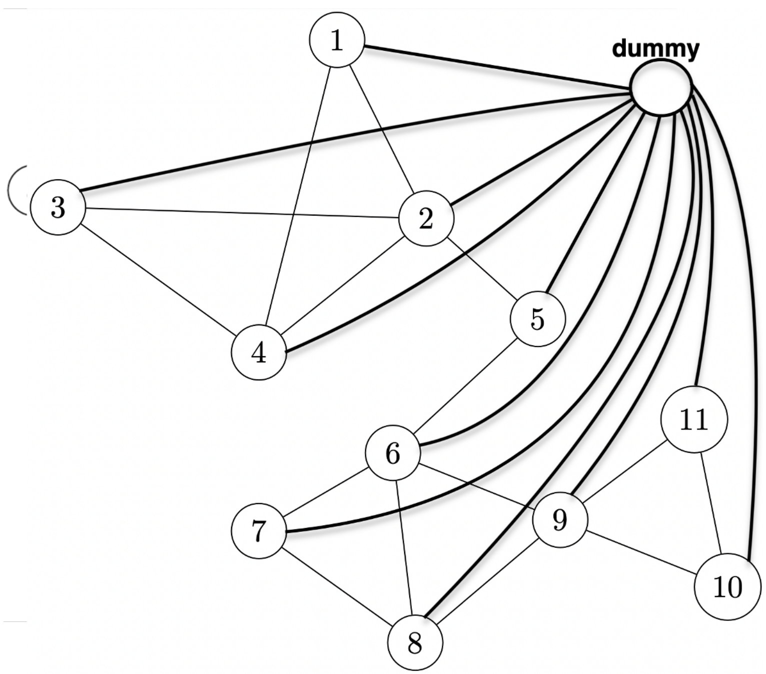


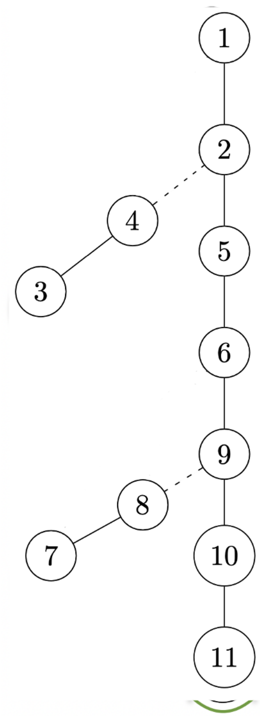
DFS-tree ***T***

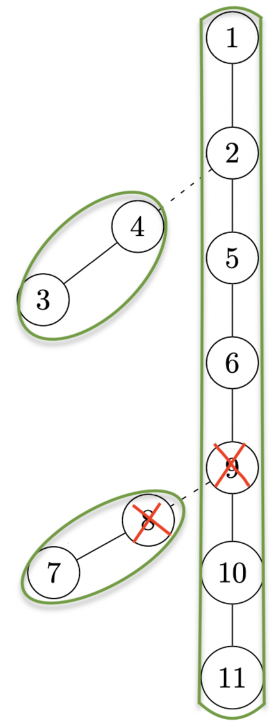
(a,1)
(b,2)
(c,3)
(d,4)
(e,5)
(f,6)
(g,7)
(h,8)
(i,9)
(j,10)
(k,11)
(l,12)
(m,13)
(n,14)
(o,15)
(p,16)
(q,17)
(r,18)

Methodology

Overview

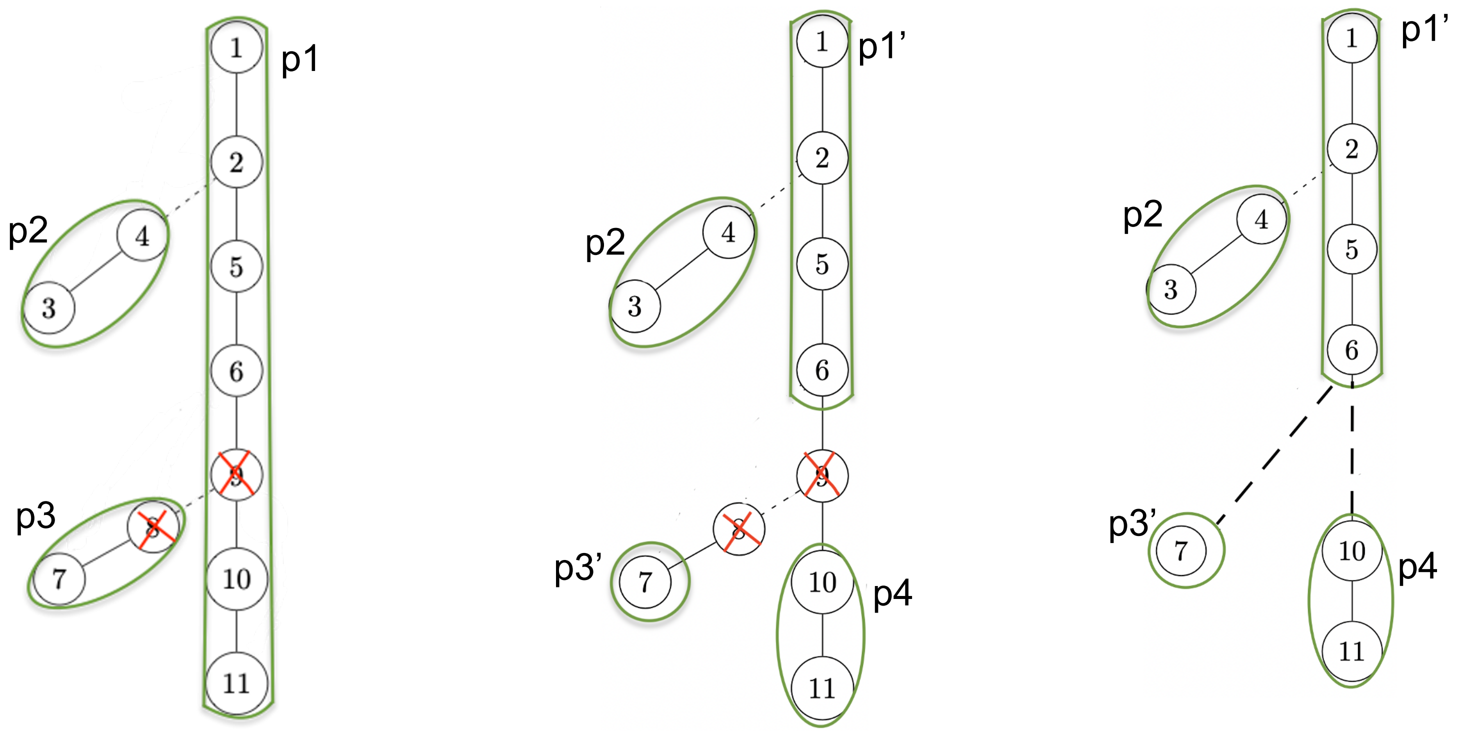






Updating the Shallow Tree

1. Delete inactive nodes from ***ST***
2. Set the ***nodePath*** for all active nodes
3. Set the parent of each ***nodePath***



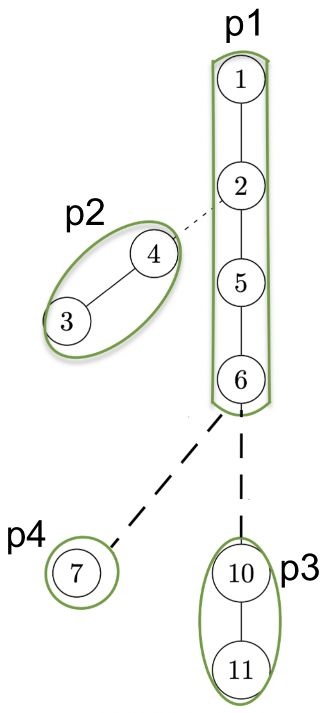
ActiveComp

ActiveComp(***G***, ***T***, ***T****, ***u***)

u.nodePath = ***p***

DFS on the shallow tree:

- Add the *nodePath* ***p*** to the partially grown spanning-tree ***T****
- Fill *Efficient_AL* for all the nodes on ***p***
- Continue with calling ActiveComp recursively on nodes in *Efficient_AL* for all the nodes on ***p***

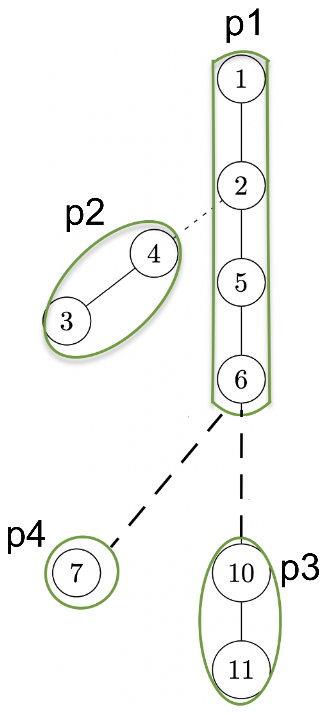


Query to the Data Structure

query(node ***u***, nodePath ***p***)

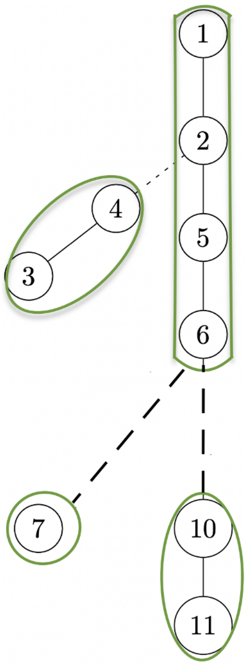
query(7,p1)

- returns a vertex on the ***p*** that is connected to ***u*** if there is any otherwise returns *nullptr*
- ***u*** has to be on one of the ***p***'s descendant



Computing Efficient Adjacency List

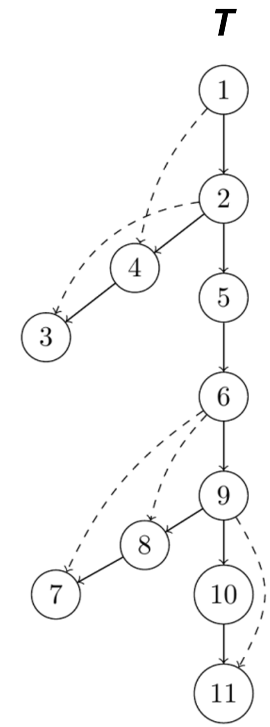
Vertices on a *nodePath* ***p*** can only have edges to the ***p***'s ancestors or descendants in the shallow tree



For each node ***u*** on the *nodePath* ***p*** that is being attached to ***T**** we need to make two categories of queries to fill its *Efficient_AL*:

1. queries from ***u*** to an ancestor of ***p*** which is not attached yet
2. queries from nodes on each descendant of ***p*** to ***p***

Data Structure



For each node ***u*** of the original graph and its corresponding DFS-tree ***T*** we save ***u***'s ancestors in ***T*** that ***u*** has an edge to.

For example:

dfs id:

$$(1,1),(2,2),(5,3),(6,4),(9,5),(8,6),(7,7),(10,8),(11,9),(4,10),(3,11)$$

Data structure *Anc_Nbr*:

node id	1	2	3	4	5	6	7	8	9	10	11
ancestors	{}	{1}	{2,4}	{1,2}	{2}	{5}	{6,8}	{6,9}	{6}	{9}	{9,10}

Time Complexity

Time complexity of *ActiveComp* is bounded by *Compute_Efficient_AL*

Compute_Efficient_AL time complexity is bounded by the number of calls to function *query*

For each node ***u*** we make a query to a path in the two following scenarios:

- ***u*** is not visited, one of ***u***.nodePath's ancestors is being attached to ***T****
- ***u***.nodePath is being attached to ***T**** and the query is from ***u*** to one of ***u***.nodePath's ancestors who are not attached

The height of shallow tree ***ST*** is d

Maximum number of calls to query from all ***n*** nodes is ***nd***

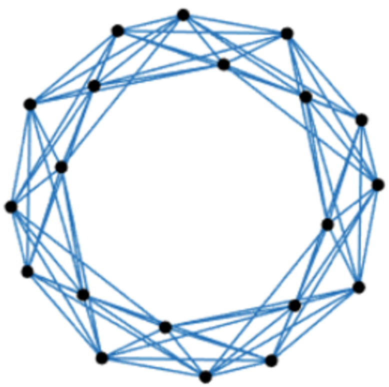
Each query takes *log*(n)

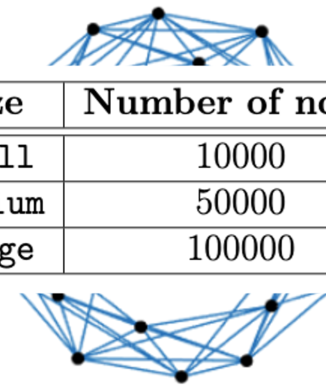
In total we the time complexity is $O(nd\log(n))$

Experimental Evaluation

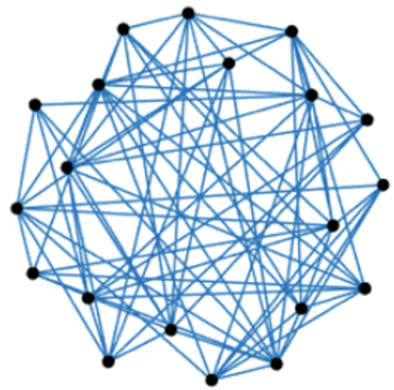
Synthetic Graph generation

Using *small-world* model





Size	Number of nodes
`small`	10000
`medium`	50000
`large`	100000



regular network

high average shortest path
high clustering coefficient

small-world network

low average shortest path
high clustering coefficient

random network

low average shortest path
low clustering coefficient

Synthetic Time Series Data Generation

$n \times h$ matrix X representing time series, nodes' status through time

Two scenarios:

1. Random
2. Forest Fire

Time Steps

Vertices

	1	2	3	4	5
A	1	1	1	1	0
B	1	1	0	0	0
C	0	1	1	1	0
D	0	1	1	1	0
E	0	1	1	1	1
F	0	0	0	1	1

Other Baselines

- Simple DFS
- Dynamic DFS [Baswana et al.,2019]
- Edge Deletion [Albert et al., 1997]

Percentage Active Nodes

We show the result in two different categories:

1 . High percentage active nodes

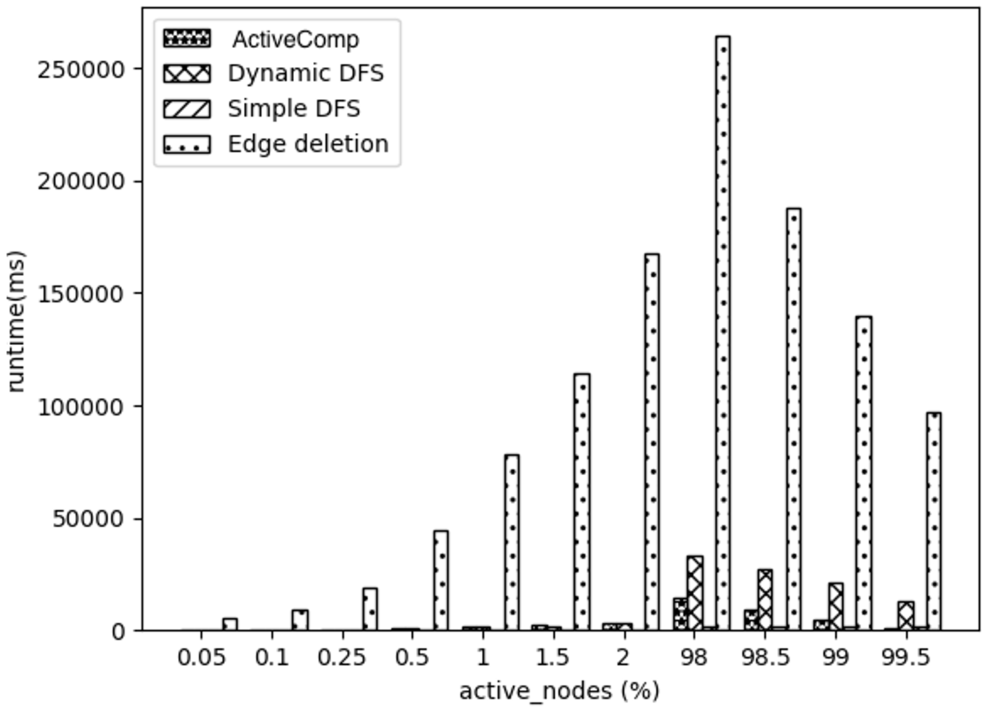
more than 99 percentage of nodes are on

2 . Low percentage active nodes

less than 1.5 percentage of nodes are on

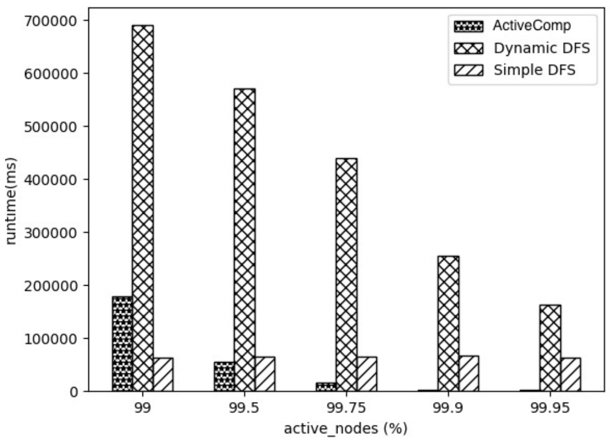
Performance of all Four Algorithms

- small-world network
- 10,000 nodes
- random scenario
- 1000 time steps

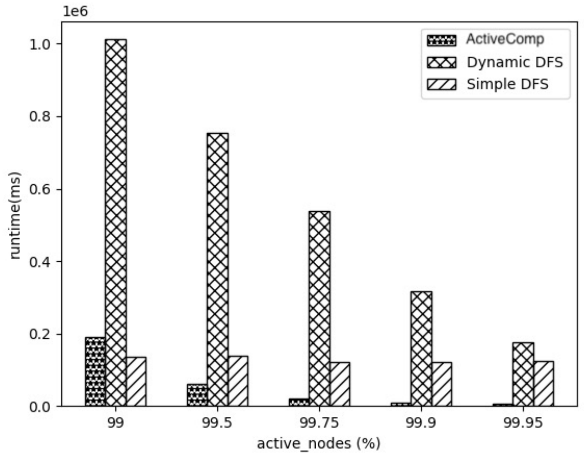


High Percentage Active Nodes

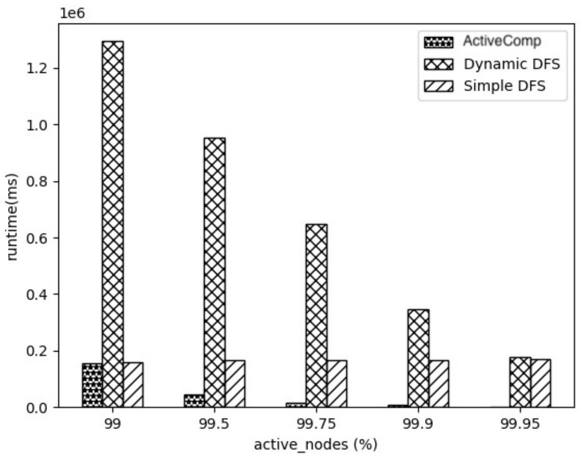
- small-world network with 50,000 nodes
- random scenario with 1000 time steps



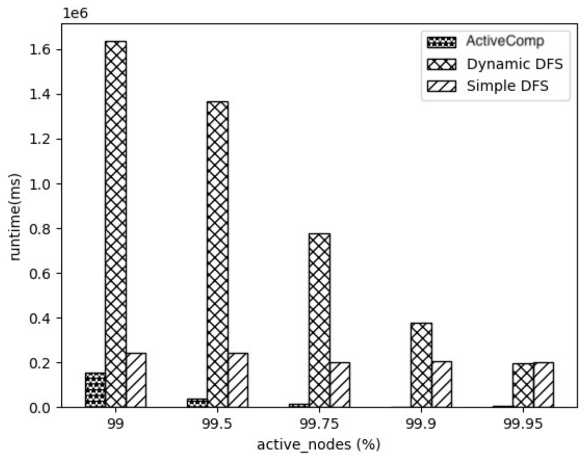
n=50k, k=250



n=50k, k=500

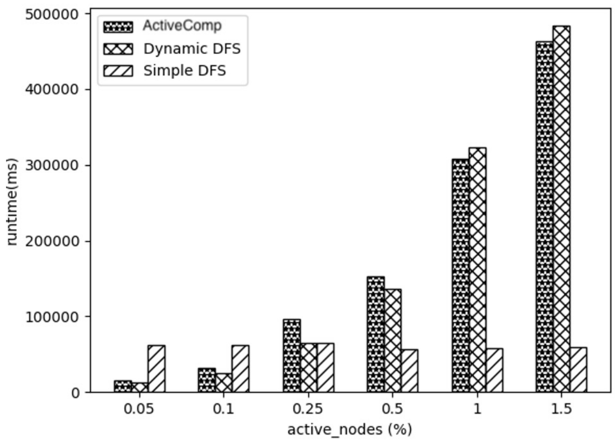


n=50k, k= 750

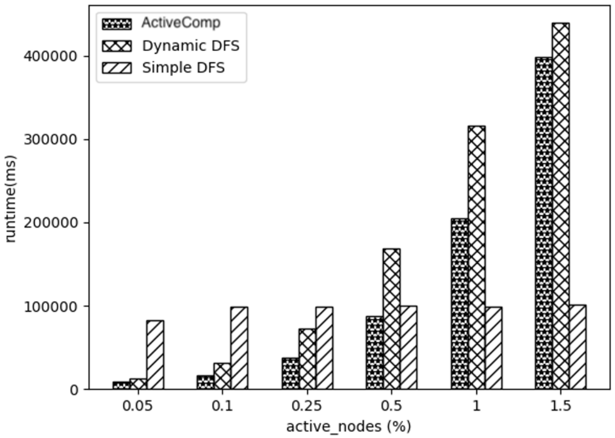


n=50k, k =1000

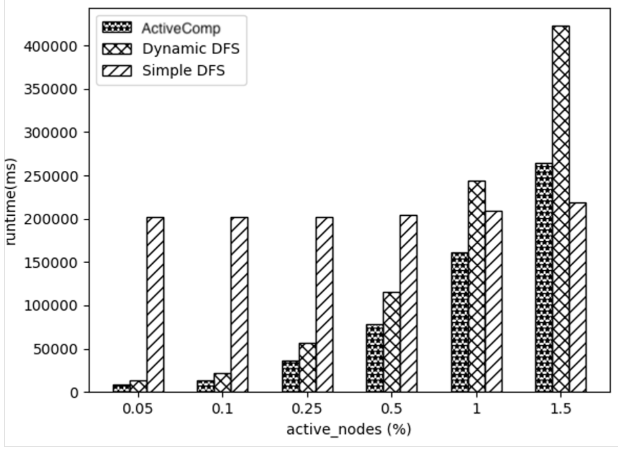
Low Percentage Active Nodes



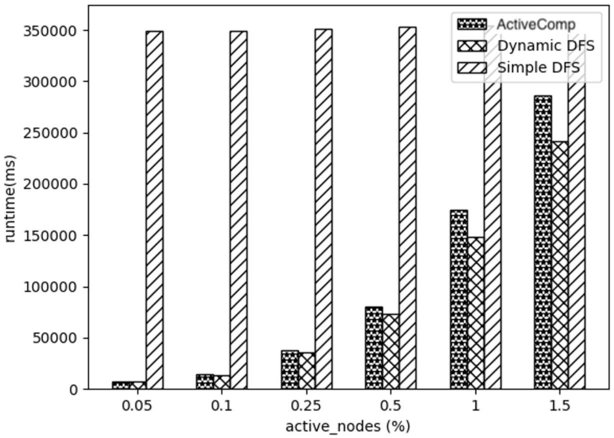
n=50k, k=250



n=50k, k =500



n=50k, k =750

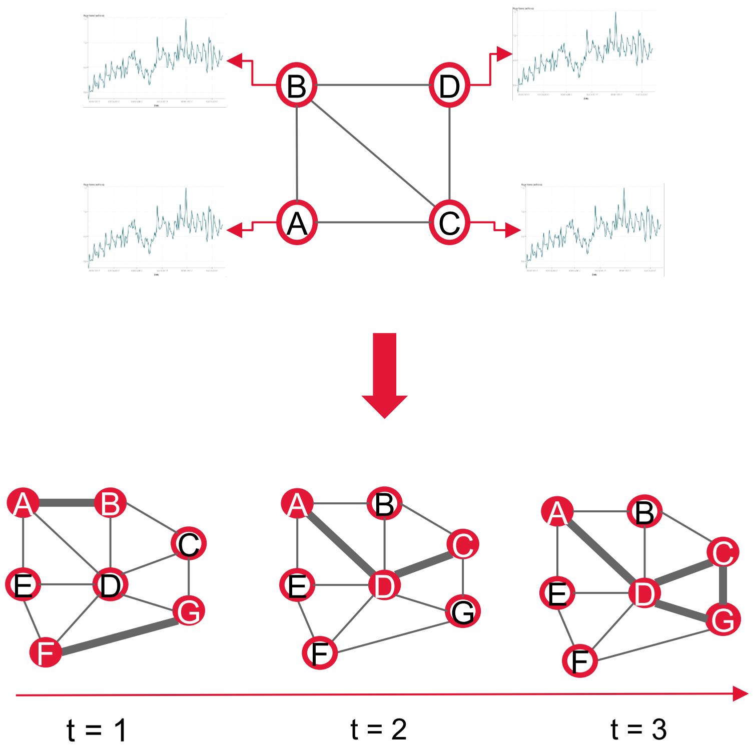


n=50k, k=1000

Conclusion, Limitations & Future Work

Conclusion

- Introduce a new problem of finding active components in a network of time series.
- Introduce ActiveComp
- Show the time complexity
- Empirically compare it to other baselines



Limitations

1. Scalability to Very Large Graphs
2. Underperforming in Specific Instances of the Problem

Future Work

1. Employing Parallel Computations
2. Fine-tuning to Accommodate Different Network Topologies
3. Extending the Comparative Empirical Analysis

Thank You