K-Core Queries on Dynamic Graphs

Shantwana Dixit (2016MCS2913)
Mahdihusain Momin (2014CS50288)

Under the guidance of :
Prof. Sayan Ranu
Prof. Amitabha Bagchi

Introduction

What is a k-core?

- Let $G = (V, E)$ be a graph, where V is the set of vertices & E is the set of edges.

- $H = (W, E|W)$ induced by the set W is a k-core or a core of order k iff

  (i) $\forall v \in W, \quad \deg_H v \geqslant k$, and

  (ii) H is a maximum subgraph with this property

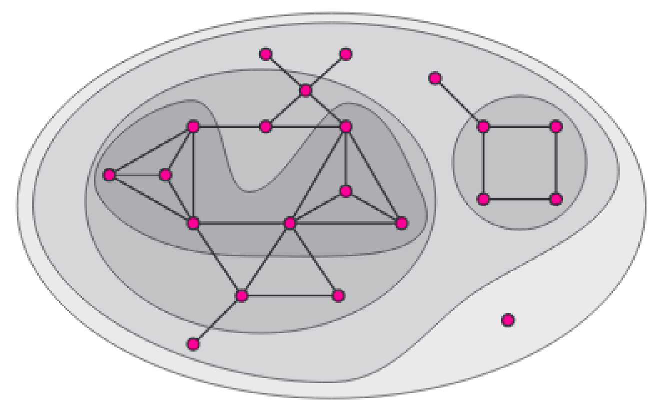


Figure 1: *0, 1, 2 and 3 core*

I. The cores are nested, $i < j \Rightarrow H_j \subseteq H_i$
II. Cores are not necessarily connected subgraphs.

A k-core is basically a maximal group of entities, all of which are connected to at least k other entities in the group.

Problem Statement -

- For a real-world dynamic network, given:
  - a threshold of k-core, i.e. θ
  - start time t_1 , and
  - a time duration Δt

- Find the nodes that lie in the θ-core for the entire duration Δt, starting from t_1.

Reference -

Incremental k-core decomposition : algorithms and evaluation (VLDB 2016)

- Ahmet Erdem Sariyuce, Bugra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, Umit V. Catalyurek

Methodology

Baseline Approach

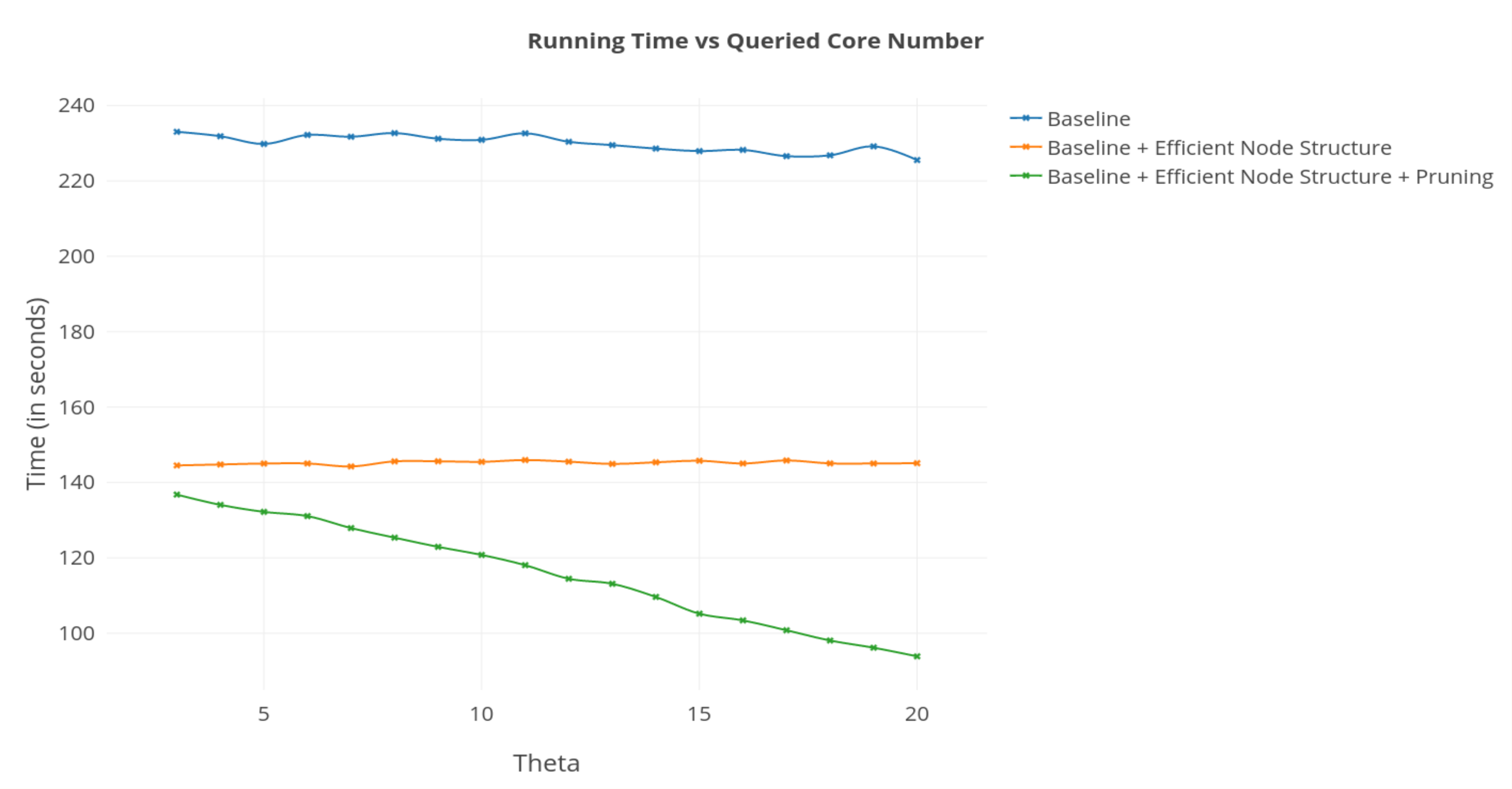
- Insert/Delete edges as required till time t_1
- Perform *Core Decomposition* at time t_1 and store the nodes in the result set that have their k-value equal to θ.
- For every edge insertion/deletion from time t_1 onwards, till Δt time duration, perform updation of k-values of nodes using the *incremental algorithms* of VLDB 2016 paper.
- At each time instant, if any of the nodes in the result set gets its k-value $< \theta$, remove that node from the result set.

Optimizations

- Use efficient node structure.
  - Map to store neighbours for O(1) access.
  - Store neighbour IDs in the bins corresponding to their k-values.

- Pruning using Upper Bound.
  - If a node u has its k-value < θ throughout the time window, this node does not impact the result set.
  - Use degree of the nodes as the upper bound, to find a subset of these nodes.
  - Two passes -
    - First pass : Find the subset of not required nodes.
    - Second pass : Discard all the edges that have any of their endpoints in this subset, and process only the remaining ones.

Comparison with Baseline - Stack Overflow Dataset (2464606 nodes, 17823525 edges)

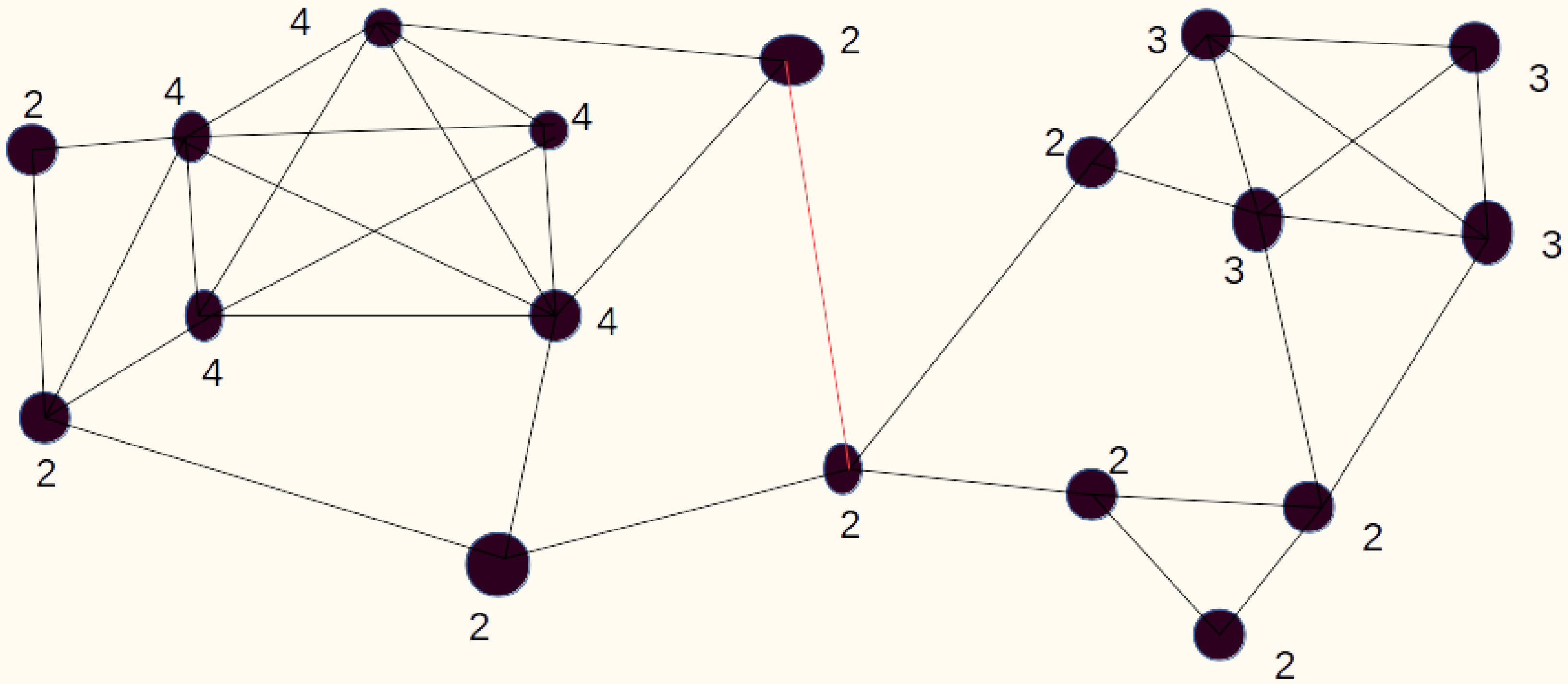


Iterative Recolor Algorithm - Insertion of (u,v)

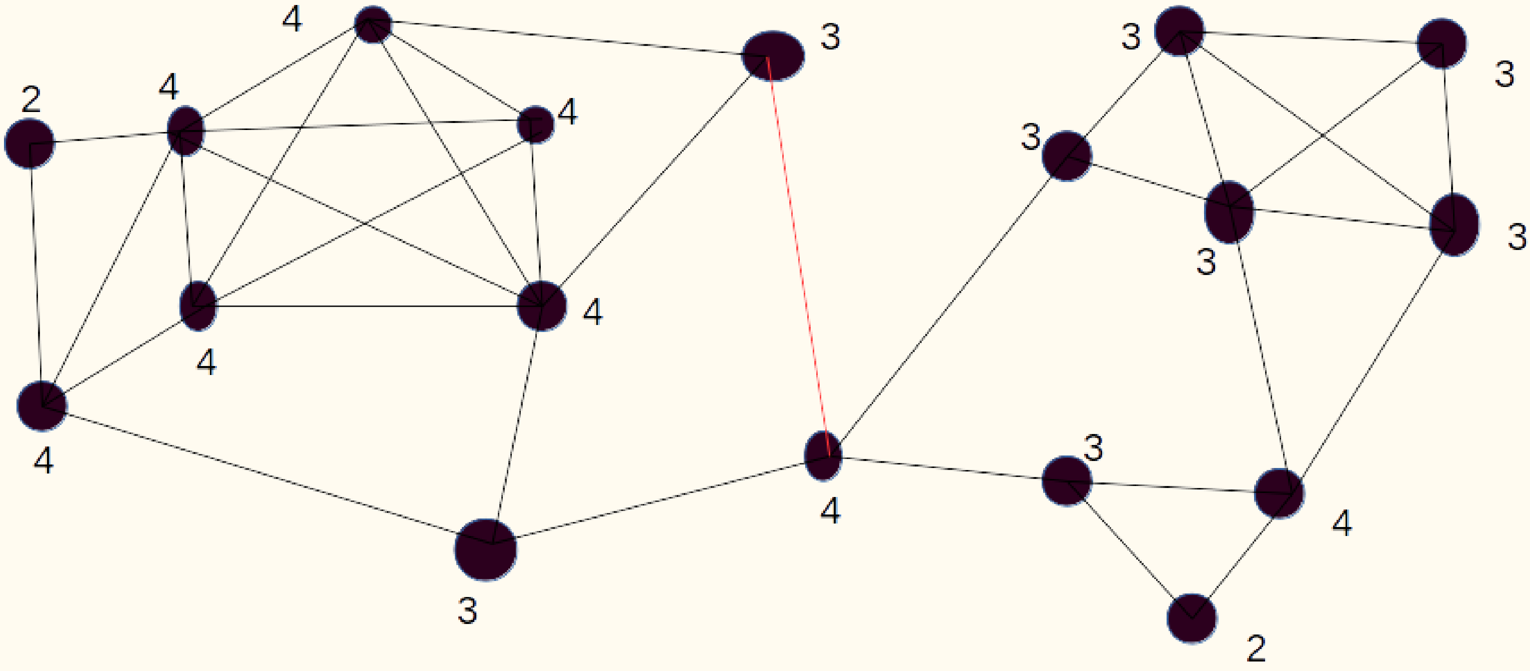
- Find the v_C set using Breadth First Search (Coloring) -
  - v_C set is the induced core subgraph of u, or v, or both, depending upon which of these has a lower core number. Let this core number be k.
  - Set the color of nodes in v_C to *true.*
- Recoloring Process -
  - Find the nodes in v_C whose k-values definitely remain unchanged; recolor these nodes *false.*
  - Start from those nodes in v_C whose MCD = k, and propagate here onwards to find all those nodes who cannot be a part of k+1-core
- Final Update -
  - Increase the core numbers of those nodes in v_C whose color is set *true,* to k+1.

MCD values are maintained by the algorithm itself, and need not be specially computed.

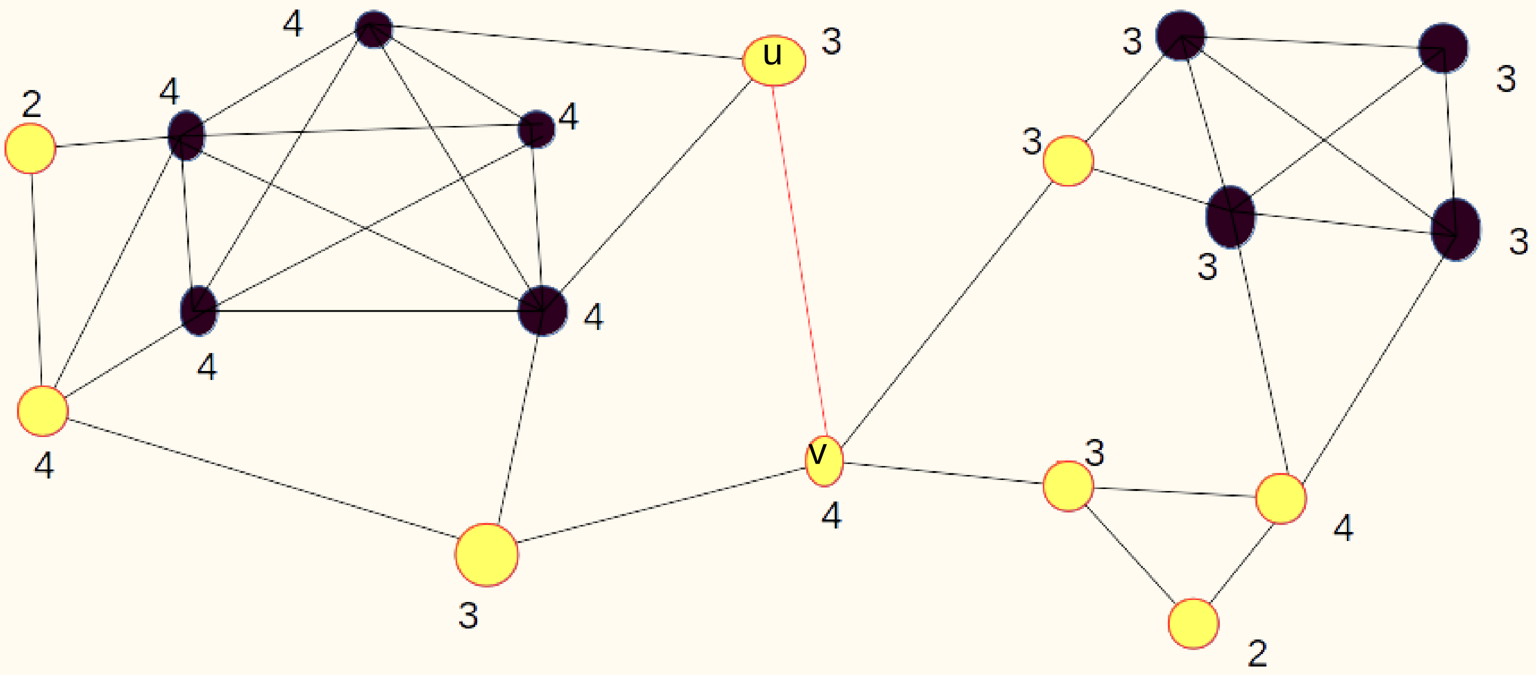
K Values



MCD Values

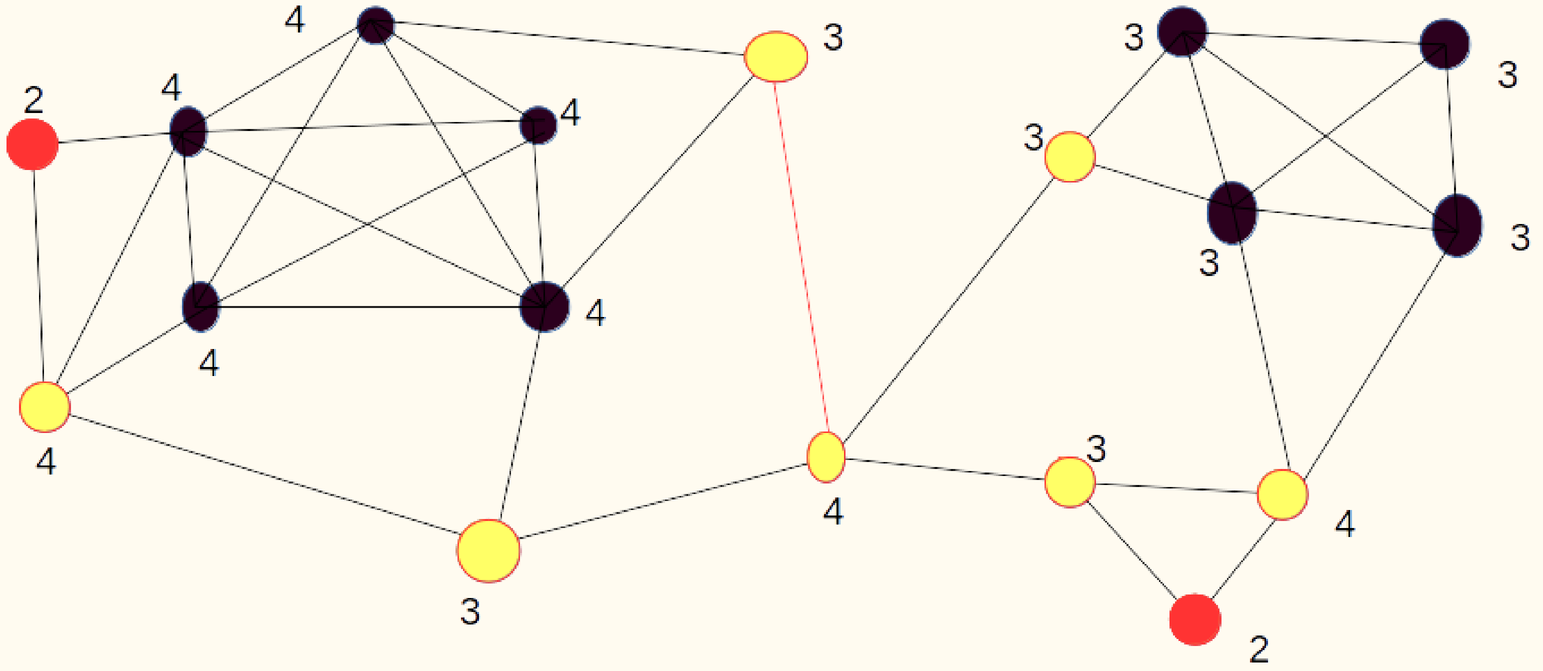


MCD Values

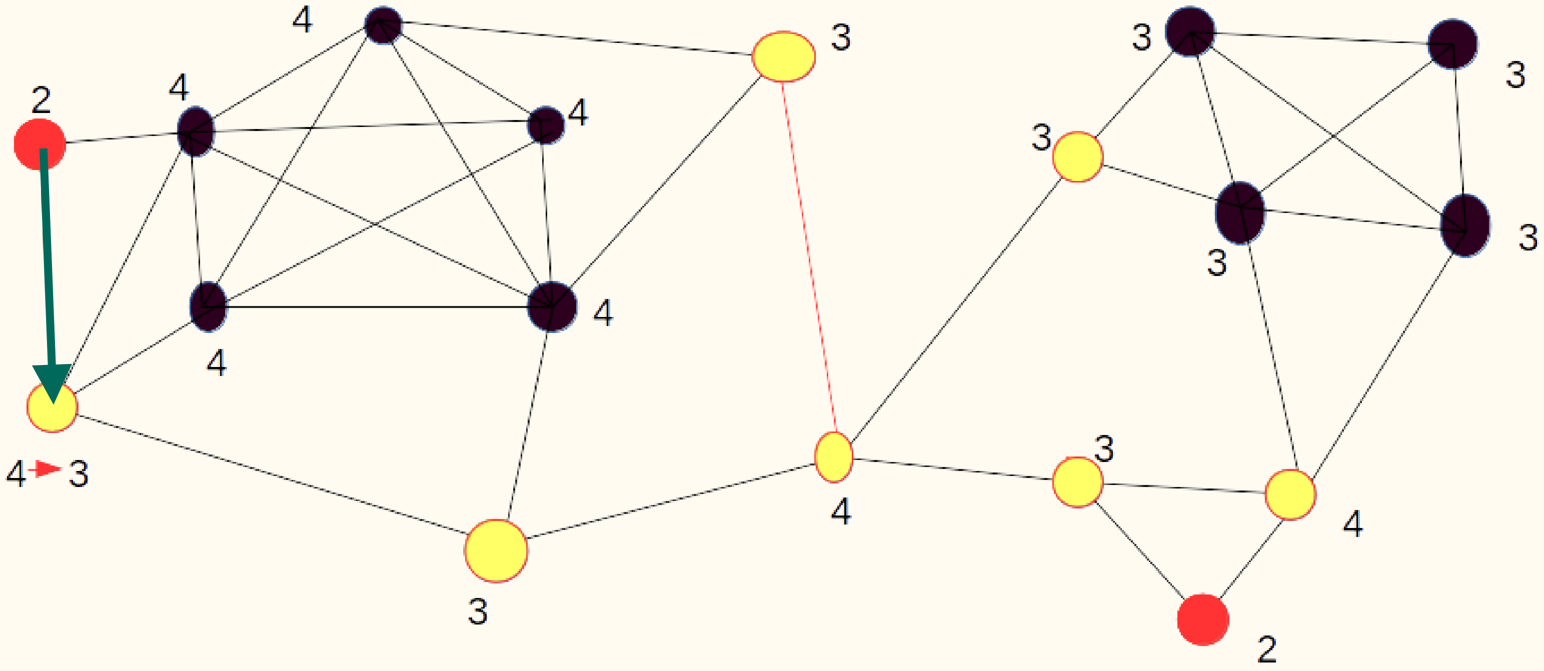


MCD Values

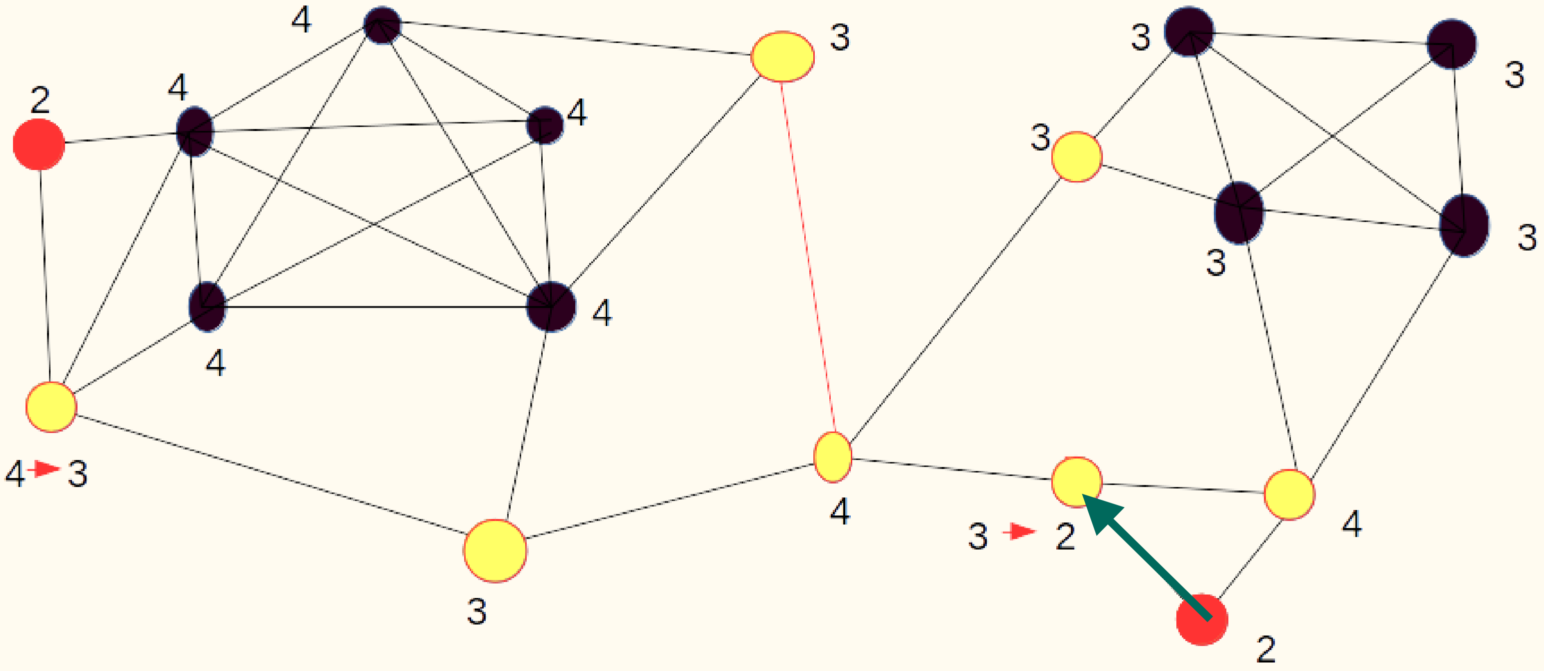
MCD>=K



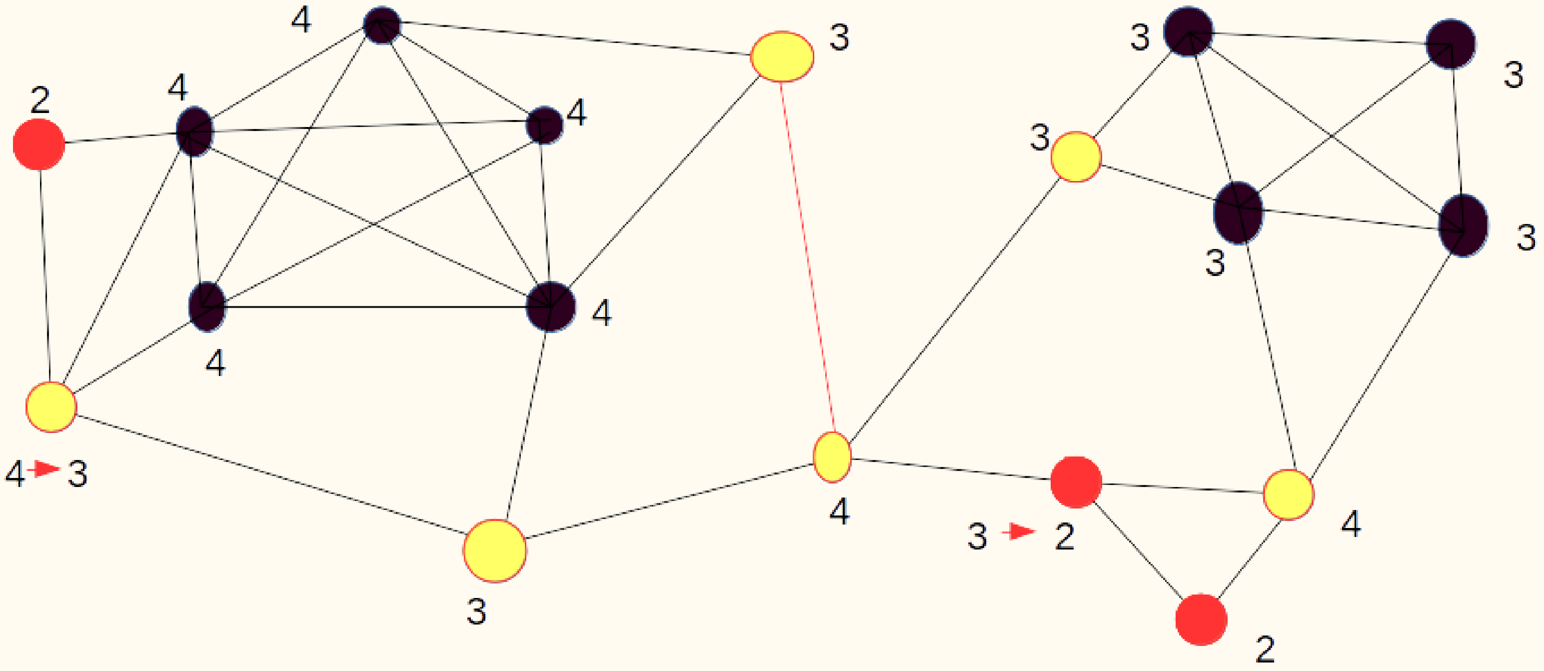
BFS



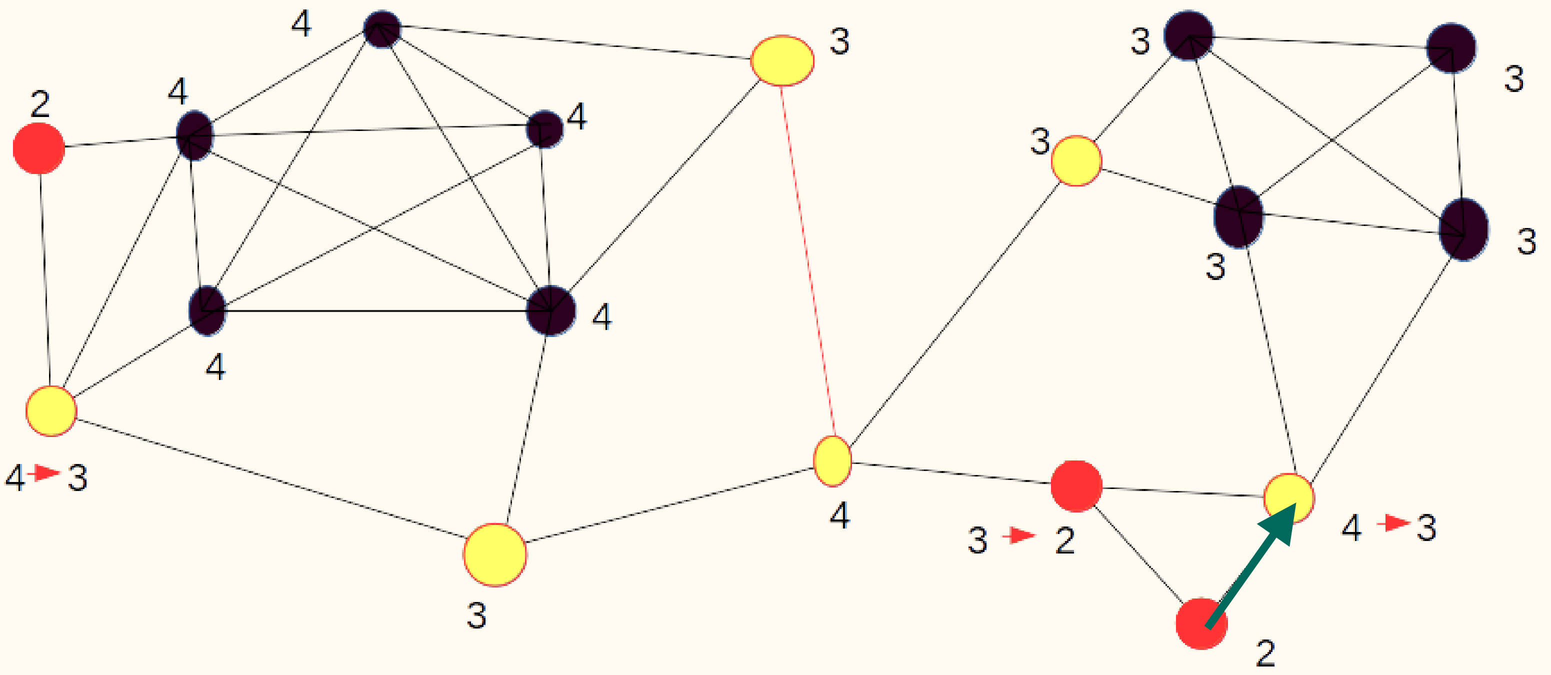
BFS



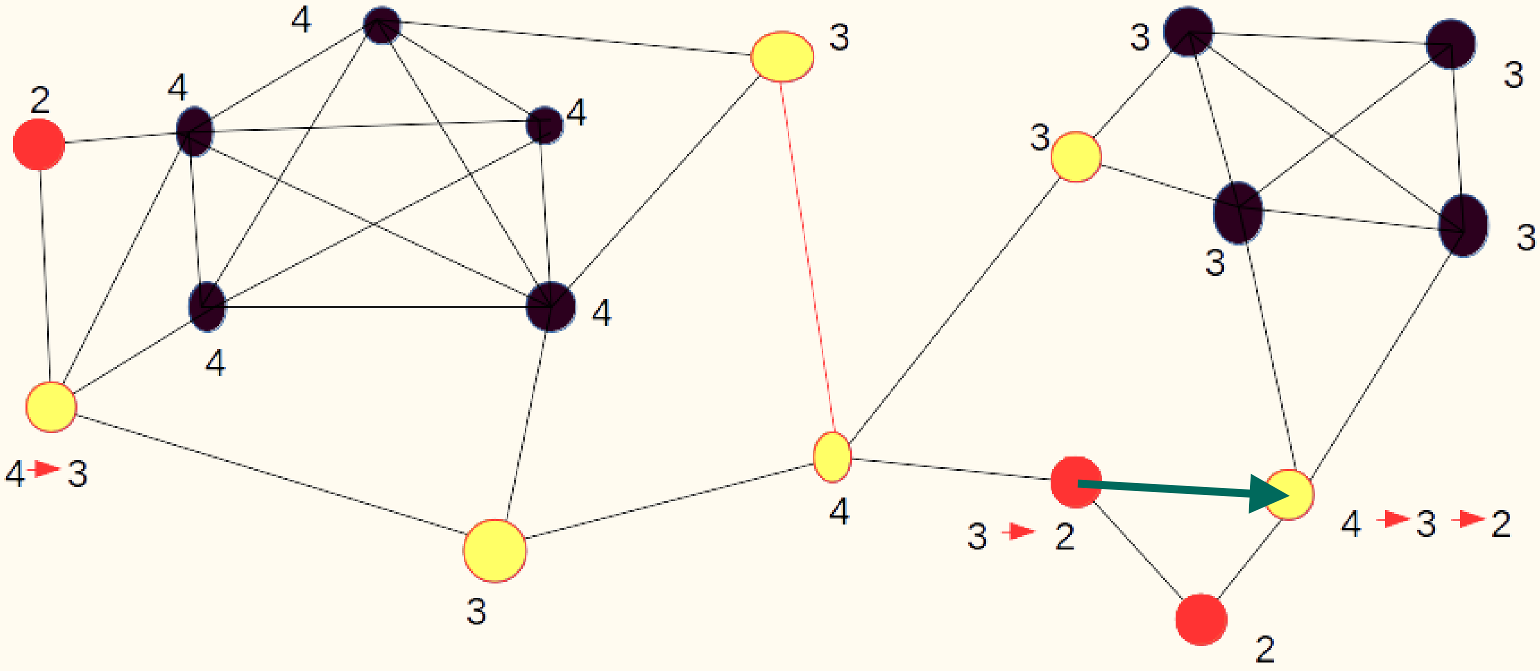
BFS



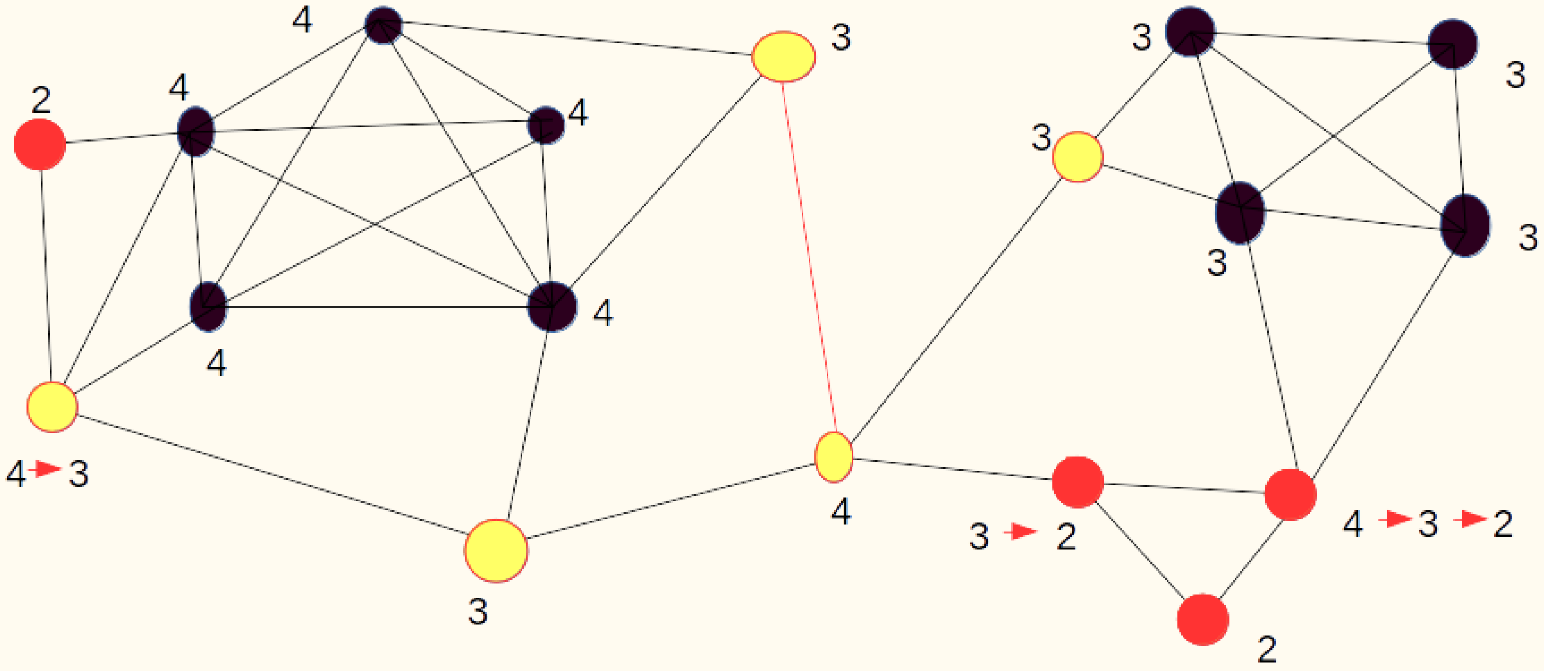
BFS



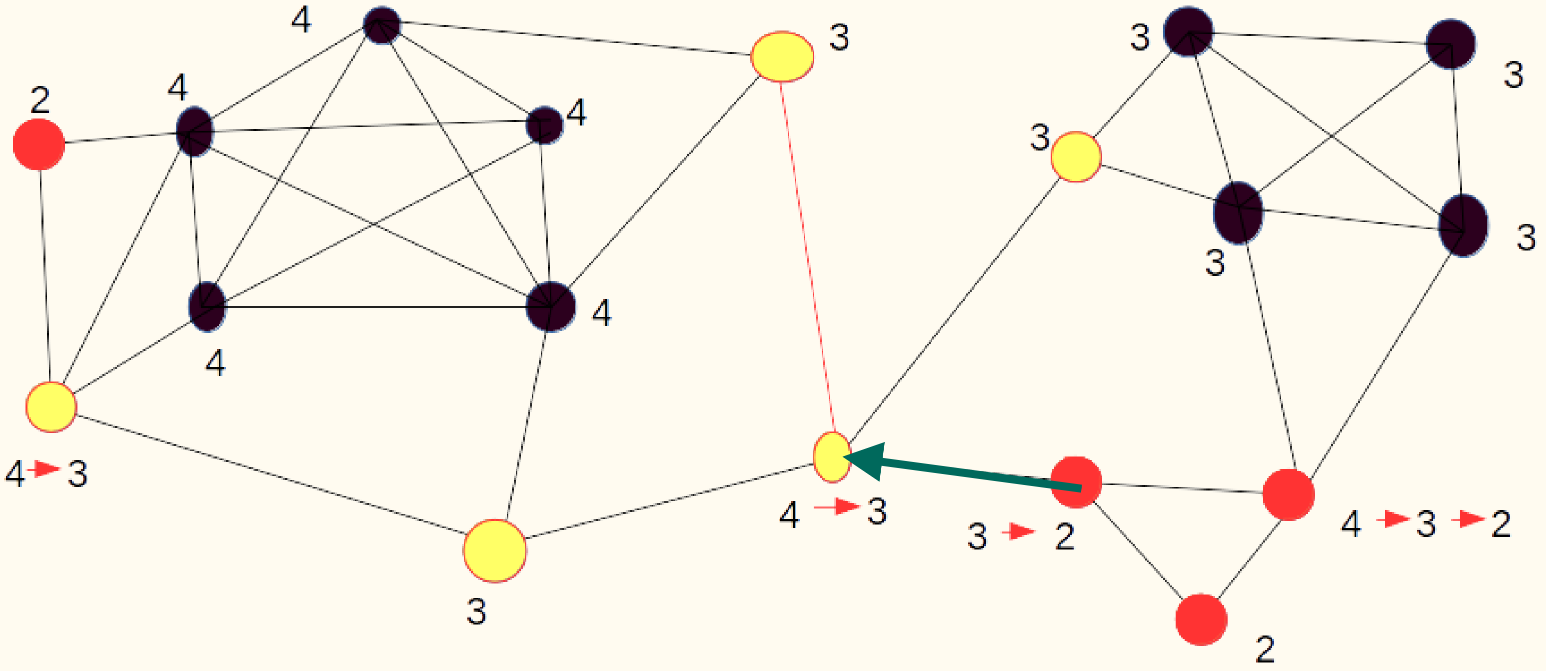
BFS



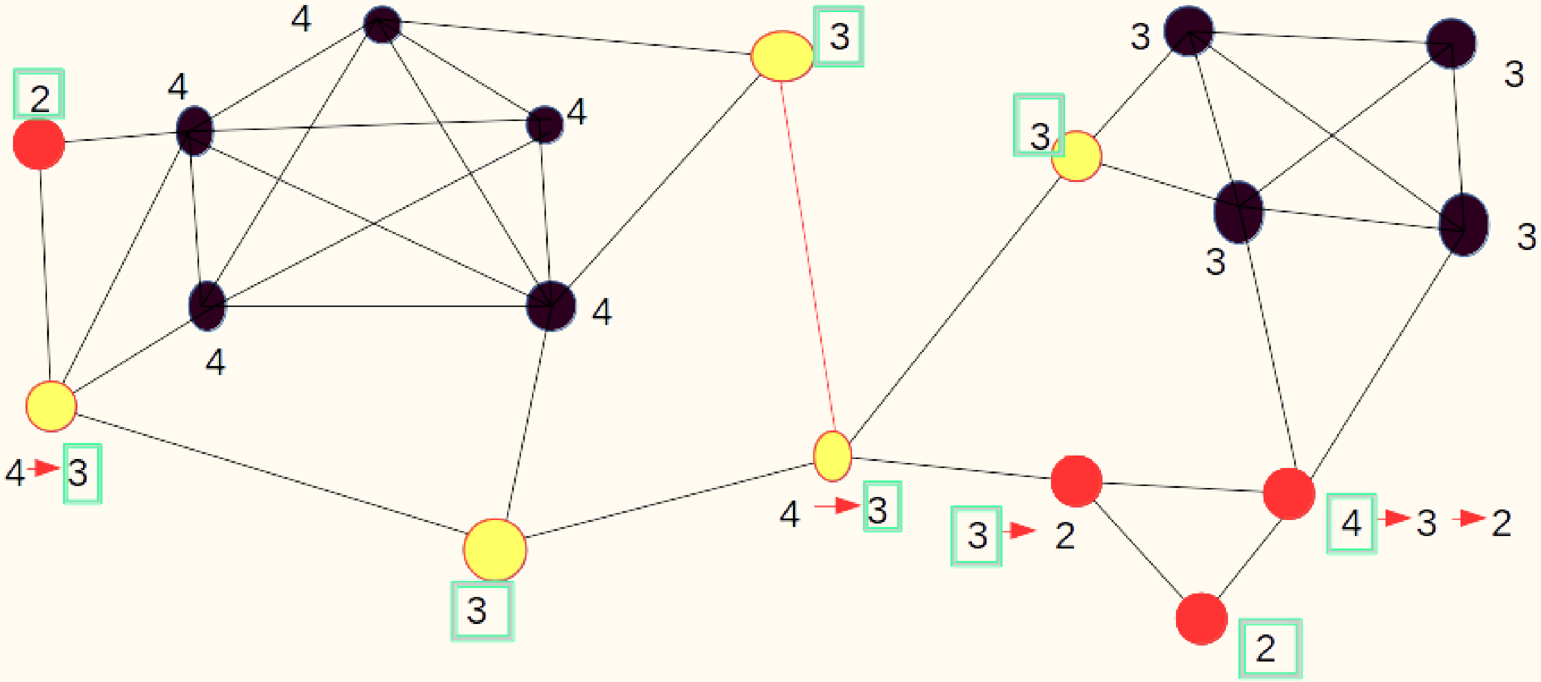
BFS



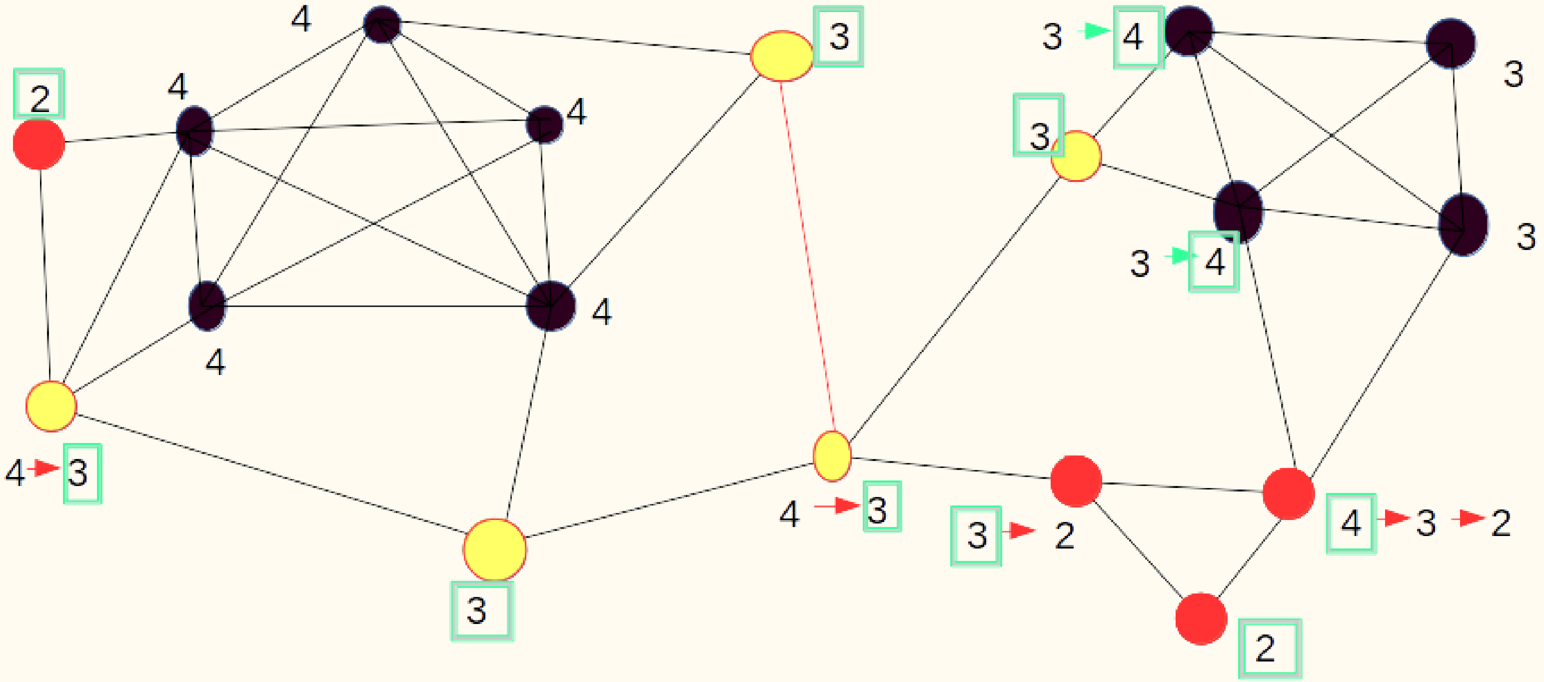
BFS



MCD Values



MCD Values



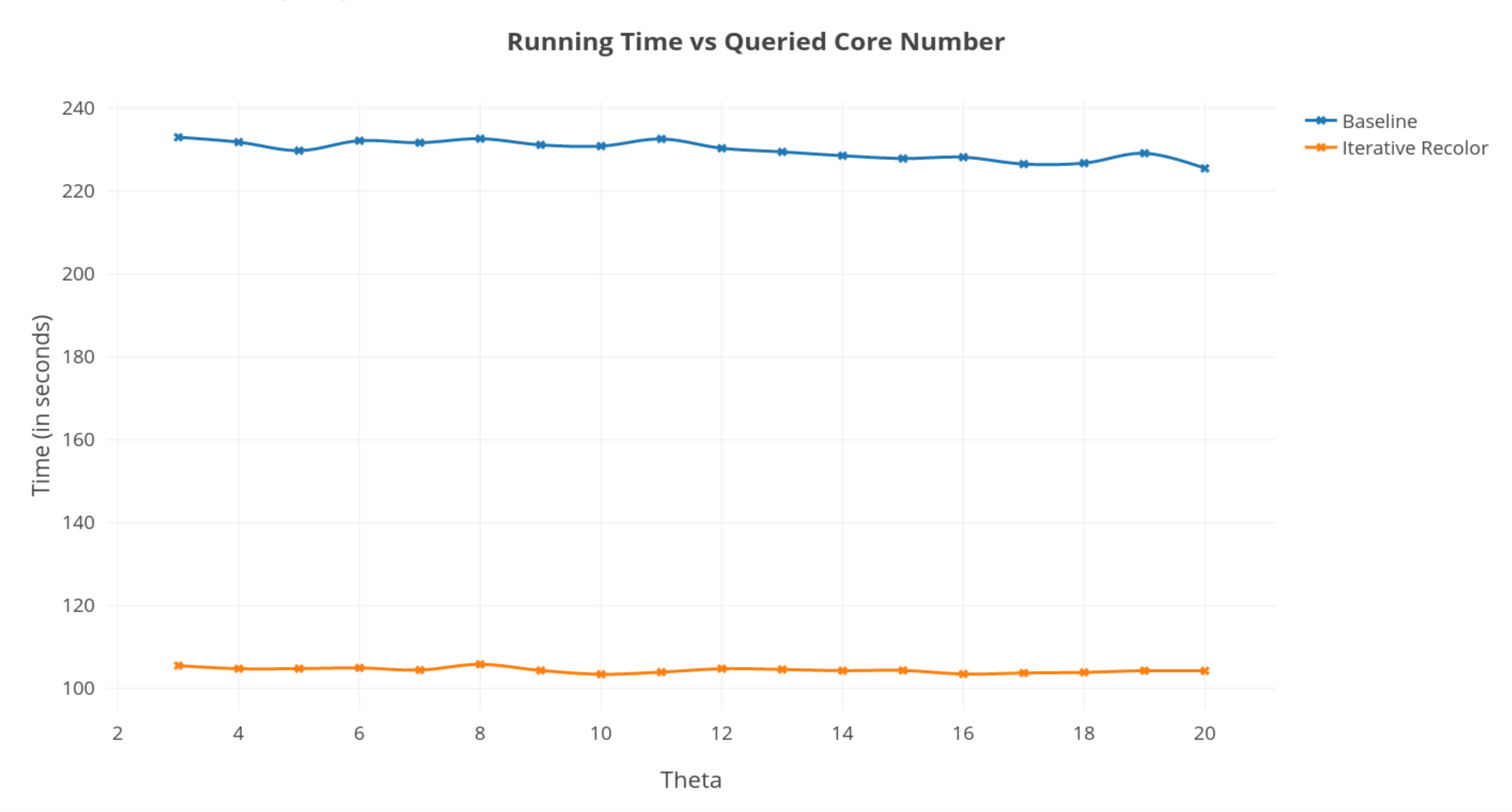
Iterative Recolor Algorithm - Deletion of (u, v)

- Coloring - Find the v_C set similarly using BFS

- Recoloring Process -
  - Find the nodes in v_c whose k-values definitely need to be updated; recolor these nodes *false.*
  - Start from u or v depending upon whose MCD = k-1, and propagate here onwards to find all those nodes who cannot remain in the k-core anymore.

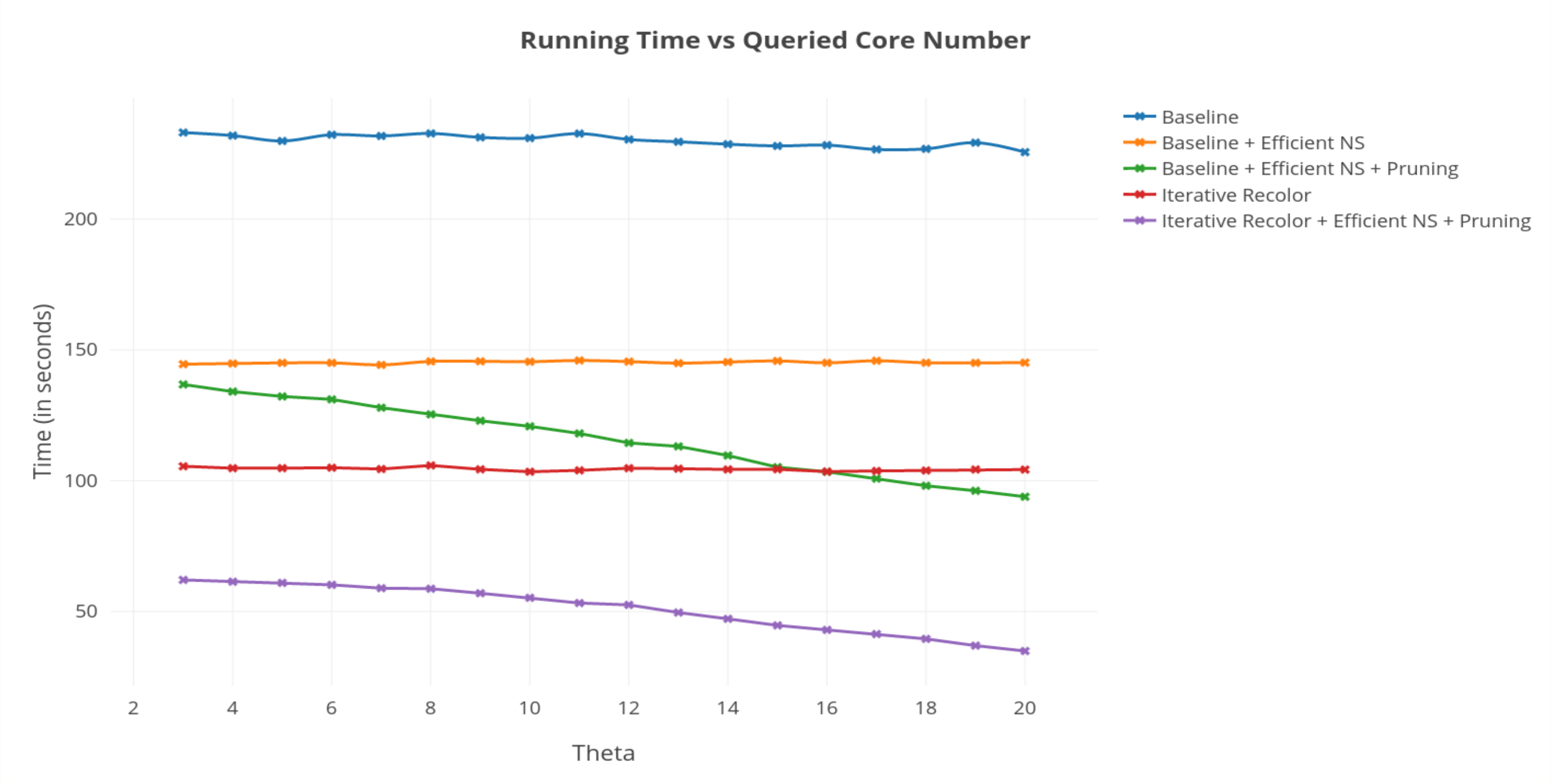
- Final Update -
  - Decrease the core numbers of those nodes in v_c whose color is set *false,* to k-1.

Comparison with Baseline - Stack Overflow Dataset (2464606 nodes, 17823525 edges)



Final Results

StackOverflow Dataset (2464606 nodes, 17823525 edges)

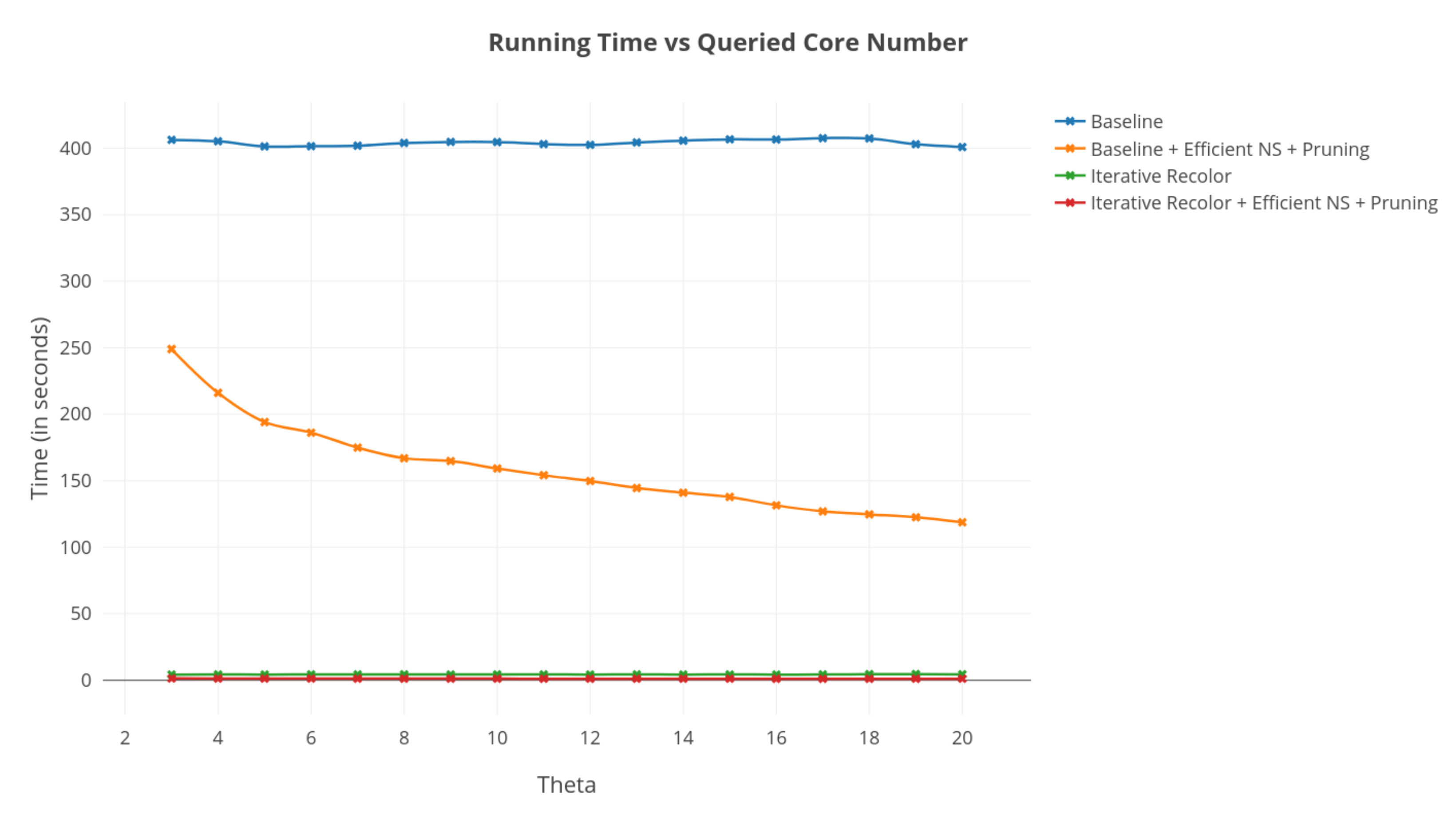


Observations

- Use of *efficient node structure* reduces the runtime by 37%.
- *Pruning strategies* reduce the runtime by 7-35% for θ varying from 3 to 20.
- With *Iterative Recolor Algorithm,* we are more than 2 times faster.

Combination of all our techniques makes us ***5 times*** *faster than the baseline.*

Wikipedia Dataset (1140149 nodes, 7833140 edges)



Future Work

- Reduce the runtime for smaller values of θ -
  - As per our query, we are not interested in knowing the exact k-value of a node.
  - The only requirement is to check whether a node has its k-value above θ throughout the given time window.
  - Design techniques that utilise this structure of the query, to reduce the time for small θ values.

- Checkpointing -
  - Propose space efficient techniques to store the graph at various time instants, serving as the checkpoints.
  - Depending upon the query start time, we can then choose the closest checkpoint to proceed with finding the answer set.

Thank You!