Critical Nodes Detection: Node Merging Approach

Hongbo Qiu
Zhejiang Gongshang University
hongboq.zjgsu@gmail.com

Renjie Sun
East China Normal University
renjie.sun@stu.ecnu.edu.cn

Chen Chen
University of Wollongong
chenc@uow.edu.au

Xiaoyang Wang
The University of New South Wales
xiaoyang.wang1@unsw.edu.au

Ying Zhang
Zhejiang Gongshang University
ying.zhang@zjgsu.edu.cn

ABSTRACT

Various cohesive models are widely employed for the analysis of social networks to identify critical users or key relationships, with the k-core being a particularly popular approach. Existing works, such as the anchor k-core problem, aim to maximize k-core by anchoring nodes (the degree of anchor nodes are set as infinity). However, we find that node merging can also enlarge the k-core size. Different from anchoring nodes, nodes merging can cause both degree increase and decrease which brings more challenges. In this paper, we study the core maximization by node merging problem (**CMNM**) and prove its hardness. A greedy framework is first presented due to its hardness. To scale for large networks, we categorize potentially influential nodes and provide a detailed analysis of all node merging pairs. Then, based on these analyses, a fast and effective algorithm is developed. Finally, we conduct comprehensive experiments on real-world networks to evaluate the effectiveness and efficiency of the proposed method.

CCS CONCEPTS

• **Information systems** → **Network data models**.

KEYWORDS

k-core; subgraph maximization; node merge

ACM Reference Format:
Hongbo Qiu, Renjie Sun, Chen Chen, Xiaoyang Wang, and Ying Zhang. 2024. Critical Nodes Detection: Node Merging Approach. In *Companion Proceedings of the ACM Web Conference 2024 (WWW '24 Companion), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3589335.3651485

1 INTRODUCTION

Graph analysis has received tremendous attention in recent years. The cohesive subgraph is one of the most important tools in analyzing graph data. Among all cohesive subgraph models, k-core [2] is the most widely used and requires that the nodes in the subgraph have degree at least k. Recently, a lot of studies have tried to enlarge k-core by anchoring nodes [12] or adding edges [14], etc. However, merging nodes [7] as another perspective that can expand the k-core is completely ignored by previous studies. When two nodes are merged, the degree of the "new" node is larger than these two nodes, and it will be possible to expand the k-core. Merging nodes also has many real-world applications, such as merging logistics entities [4] and team formation [3], etc. For example, in [4], they focus on the case of merging several separate logistics entities, and highlight its contribution to supply optimization.

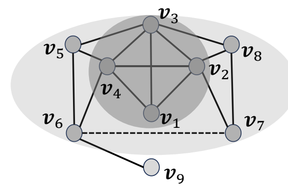


Figure 1: Motivation example

To fill the gap, in this paper, we propose and investigate the core maximization by node merging problem (**CMNM**). Specifically, given a graph G, an integer k and a budget b, we aim to find b pairs of nodes to merge to maximize the size of k-core. For example, in Figure 1 with 9 nodes, the graph in dark grey is a 3-core. After merging v_7 and v_9, v_7 has a new neighbor v_6 (indicated by a dashed line). Then the whole graph in light grey becomes 3-core. To the best of our knowledge, we are the first to investigate the CMNM problem. The main challenges of the problem are two folds. Firstly, we need to consider every pair of nodes in the graph and this search space is huge. Secondly, we prove the problem is NP-hard. In this paper, we first present a greedy search framework and then analyze different types of merging cases to reduce the search space. Finally, experiments on real datasets are conducted to demonstrate the effectiveness and efficiency of the proposed method.

Related work. Many cohesive models are used to study critical nodes or key relationships detection of the graph, such as k-core [8, 14], k-truss [9] and clique [10]. Existing works often use adding edges [14], anchoring nodes [12] to enlarge the corresponding k-core or k-truss. [11, 14] study the k-core maximization problem by adding edges. [12] studies k-core maximization problem by anchoring nodes while [6] tries to consider the coreness improvement globally rather than only enlarging k-core. In [1], Bu et al. study k-truss maximization by merging nodes. However, the techniques in the above research cannot support the problem in this paper, since they do not consider the node merging or only apply to truss.

2 PRELIMINARIES

we consider an undirected graph $G = (V, E)$ where V (resp. E) represents the node (resp. edge) set in G. Given a subgraph $S =$

WWW '24 Companion, May 13–17, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0172-6/24/05
https://doi.org/10.1145/3589335.3651485

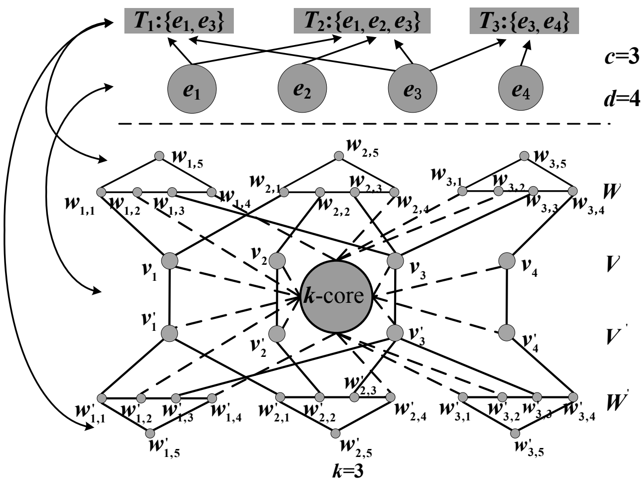


Figure 2: Construction example for NP-hard proof, k=3

$(V(S), E(S))$ in G, we denote the neighbor set of u in S by $N(u, S) = \{v | (u, v) \in E(S)\}$, and the degree of u in S by $d(u, S) = |N(u, S)|$.

Definition 2.1 (k-core). Given a graph G, a subgraph S is a k-core of G, denoted by $C_k(G)$, if (i) S satisfies degree constraint, i.e., $d(u, S) \geq k$ for every node $u \in V(S)$; and (ii) S is maximal, i.e., any supergraph of S cannot be a k-core.

Definition 2.2 (coreness). Given a graph G, the coreness of a node $u \in V$, denoted by $c(u, G)$, is the largest k such that $C_k(G)$ contains u, i.e., $c(u, G) = \max\{k | u \in V(C_k(G))\}$.

For any two nodes v_1 and v_2, the merger operation [7] for v_1 and v_2 is to "shift" the edges incident to v_2 to v_1 without adding multiple edges or self-loops, and remove v_2 from G. Specifically, after the merger between the node pair v_1 and v_2, we can obtain a new graph $G' = (V', E')$, where $V' = V \backslash \{v_2\}$ and $E' = E \cup \{(u, v_1) | u \in N(v_2, G) \wedge u \neq v_2\} \backslash \{(u, v_2) | (u, v_2) \in E\}$. For simplicity of description, we use G_{v_1,v_2} to represent the graph after merging v_1 and v_2. After merging nodes in the graph G, the graph's structure changes, consequently affecting the $C_k(G)$. We call the nodes in $V(C_k(G_{v_1,v_2})) \backslash V(C_k(G))$ are *followers* of merging v_1 and v_2. Hence, we delineate the problem addressed in this paper as follows.

Problem statement. Given a graph G, an integer k and a budget b, the problem of core maximization by node merging (**CMNM**) is to merge b pairs of nodes to maximize the size of k-core.

Theorem 2.3. *The* CMNM *problem is NP-hard for all* $k \geq 3$.

Proof. We reduce the MC problem [5], which is proved as NP-hard, to our CMNM problem. The MC problem aims to find some sets that cover the largest number of elements with a given budget b. Consider an arbitrary instance H of MC with c sets $\{T_1, T_2, \cdots, T_c\}$ and d elements $\{e_1, e_2, \cdots, e_d\} = \bigcup_{1 \leq i \leq c} T_i$. Then, We construct a corresponding instance of CMNM problem.

The node set of constructed graph G includes five parts: W, W', V, V' and a k-core. W (resp. V) and W' (resp. V') is symmetrical in graph G. We therefore focus on W and V. W contains c sub-parts $\{W_1, W_2, \cdots, W_c\}$ which correspond to each set T_i in MC instance H. Each sub-part W_i contains $d+1$ node, i.e., $W_i = \{w_{i,1}, w_{i,2}, \cdots, w_{i,d+1}\}$. We first connect these nodes by a circle. $w_{i,j}$ is corresponding to the element e_j in T_i for $1 \leq j \leq d$. V contains d nodes, i.e., $V = \{v_1, v_2, \cdots, v_d\}$, corresponding to each element e_j. Then if T_i contains the element e_j, we add an edge $(w_{i,j}, v_j)$. After that, we need to make sure $d(w_{i,j}, G) = k$ for $1 \leq j \leq d$ and $d(w_{i,j}, G) = k-1$ for $j = d+1$. If the degree is not enough, we add edges between $w_{i,j}$ and k-core to satisfy the restriction. For each node v_j, we add $k-2$ edges to nodes in k-core. Then we do the same thing for the symmetrical W' and V', and add an edge between v_j and v'_j. Consequently, the construction is completed. Figure 2 shows an example for $d = k = c = 3$.

Based on the above construction, we can have the following properties: (i) $w_{i,d+1}$ will be deleted during the k-core decomposition in the first round, then all nodes will be deleted accordingly expect k-core. So the coreness of $w_{i,j}$ and v_j is $k-1$. (ii) The new node after merging the symmetrical nodes $w_{i,d+1}$ and $w'_{i,d+1}$ will not be deleted during the core decomposition. (iii) Among all node pairs in G, only choosing symmetrical nodes $w_{i,d+1}$ and $w'_{i,d+1}$ to merge can make the most nodes stay in k-core. The optimal merging set for CMNM problem corresponds to the optimal set collection C for MC problem. Hence, the CMNM problem is NP-hard for $k \geq 3$. □

3 ALGORITHMS

Due to the NP-hardness of the problem, in this paper, we tend to greedy heuristic. A straightforward greedy strategy involves iterating through b rounds. In each round, we select a pair of nodes (v_1, v_2) from all possible pairs in V that will yield the largest $|C_k(G_{v_1,v_2})|$. Note that, after each round, G will be G_{v_1,v_2}. However, this greedy method suffers from massive unnecessary node pair searches and is inefficient in handling larger graphs. Therefore, in this section, we analyze various node pair scenarios to identify the node set with the greatest potential to maximize the k-core. We then greedily select b node pairs from this node set (which is much smaller than V). Before presenting the details, we first introduce an interesting conclusion in the following lemma.

Lemma 3.1. *Given a graph G and any two nodes v_1 and v_2, after merging these two nodes, the coreness of the other nodes will change by at most 1, i.e., $|c(v, G) - c(v, G_{v_1,v_2})| \leq 1$ for each $v \in V \backslash \{v_1, v_2\}$.*

Proof. Note that merging nodes can cause $c(u, G)$ increase or decrease. In the decrease case, merging a pair of nodes can decrease $d(u, G)$ by most 1 for each common u. Therefore the current k-core at least satisfying $(k-1)$-core, the coreness decreases by most 1. For the increase case, we consider the opposite operation (split node) and show the decrease is limited. We split v_1 in G_{v_1,v_2} back to v_1, v_2 in G. And regarding the coreness of each node, such an operation is no worse than deleting a node. So the degree and coreness of a node can only be reduced by 1. Hence, the lemma holds. □

Definition 3.2 (k-shell). Given a graph G and an integer k, the k-shell of G, denoted by $S_k(G)$, is the set of nodes with coreness k, i.e., $S_k(G) = \{v \in V | c(v, G) = k\}$.

Based on Lemma 3.1 and Definition 3.2, only the nodes in $(k-1)$-shell can be the followers of merging two nodes. Therefore, we only need to focus on nodes that have neighbors in the $(k-1)$-shell. These nodes can be divided into three parts: the nodes in k-core (core-node), the nodes in $(k-1)$-shell (shell-node), and the

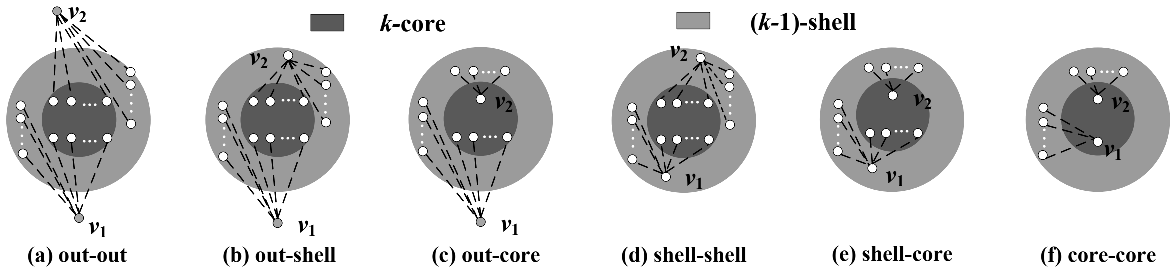


Figure 3: Merging example

remaining nodes (out-node). Based on the categorization of nodes, we can obtain 6 types of merge pairs as shown in Figure 3.

out-out merge. As shown in Figure 3(a), v_1 and v_2 are out-nodes. After merging v_1 and v_2, if v_1 stay in the k-core of G_{v_1,v_2}, for each node u in $\{u \in S_{k-1}(G) | u \in N(v_1, G) \vee u \in N(v_2, G)\}$, u has a new neighbor v_1 in $C_k(G_{v_1,v_2})$, which may increase the coreness of u. Thus, when merging the out-node, the more neighbors it has in the $(k-1)$-shell, the more potential followers it has.

out-shell merge. As shown in Figure 3(b), v_1 and v_2 are out-node and shell-node, respectively. According to [13], when computing k-core by core decomposition, there is a removing order, denoted as $\preceq$, for the nodes in $(k-1)$-shell. $u \preceq v$ iff u is removed before v by core decomposition. After merging v_1 and v_2, if v_1 stay in the k-core of G_{v_1,v_2}, for each neighbor node u in $(k-1)$-shell of v_2 with $u \preceq v_2$, v_2 will not contribute to u joining k-core, as u is removed before v_2 in $\preceq$. While, for the neighbor node u in $(k-1)$-shell of v_2 with $v_2 \preceq u$, u may increase its coreness, this is because v_2 will always be the neighbor of u when core decomposition. Thus, when merging the shell-node, the more neighbors with removing order after shell-node, the more potential followers it has. The effect of v_1 is the same as out-out merge.

out-core merge. As shown in Figure 3(c), v_1 and v_2 are out-node and core-node, respectively. Since v_2 is already in k-core, v_2 will not contribute to its neighbor joining k-core when merging v_1 and v_2. After merging, v_1 will become a k-core node, and v_1's $(k-1)$-shell neighbors have a new k-core neighbor, so the coreness of them can increase. However, the effectiveness of this merger only relies on v_1, i.e., out-node.

shell-shell merge. As shown in Figure 3(d), v_1 and v_2 are both shell-nodes. Similar to out-shell merge, after merging v_1 and v_2, if v_1 survive during core decomposition, for each neighbor node u in $(k-1)$-shell of v_1 and v_2 with $v_1 \preceq u$ or $v_2 \preceq u$, they do not be deleted. Thus, the more neighbors with removing order after v_1 and v_2, the more potential followers they have.

shell-core merge. As shown in Figure 3(e), v_1 and v_2 are shell-node and core-node, respectively. After merging, v_1 becomes a k-core node, thus each v_1's shell neighbor has a new neighbor in k-core, which may lead to the coreness increase. However, before and after merging, $d(u, C_{k-1}(G)) = d(u, C_{k-1}(G_{v_1,v_2}))$ for each u in $\{u \in S_{k-1}(G) | u \in N(v_1, G) \wedge u \preceq v_1\}$, thus the coreness of u does not change. The more neighbors with removing order after shell-node v_1, the more potential followers it has.

Note that the above two situations in Figure 3(d) and (e) can also lead to the coreness loss for the node in $(k-1)$-shell if v_1 and v_2 have a common neighbor in $(k-1)$-shell.

Algorithm 1: CMNM Algorithm

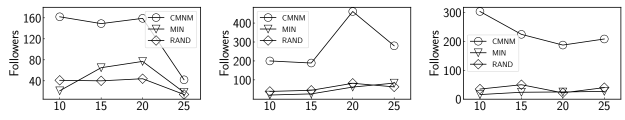
Input : G: a graph, k: degree constraint, b: budget, x: the number of out-node, y: the number of shell-node.
Output : P: the pairs of nodes to be merged

```
1  while |P| < b do
2      max ← 0; BP ← ∅;
3      compute (k − 1)-shell and the corresponding removing order ⪯;
4      X ← x out-nodes with most neighbors in (k − 1)-shell;
5      Y ← y shell-nodes with most neighbors in ⪯ after it;
6      for each node pair (v1, v2) in X ∪ Y do
7          if |C_k(G_{v1,v2})| + |S_{k−1}(G_{v1,v2})| > max then
8              max ← |C_k(G_{v1,v2}))| + |S_{k−1}(G_{v1,v2})|;
9              BP ← (v1, v2);
10     P ← P ∪ {BP}; G ← G_BP;
11 return P;
```

core-core merge. As shown in Figure 3(f), both v_1 and v_2 are core-nodes. Note that their neighbors in the $(k-1)$-shell are deleted before them during the core decomposition. Furthermore, the core-core merge can lead to degree loss if v_1 and v_2 have some common neighbors in k-core or $(k-1)$-shell. This can lead to a decrease of k-core size or $(k-1)$-shell size. Thus, we prune this situation.

Based on the above analysis, we introduce a new method to select nodes to merge. Instead of selecting the node pair from the whole node set V, we first extract a subset $X \subseteq V$ which contains x out-nodes that have the most neighbors in the $(k-1)$-shell, and a subset $Y \subseteq V$ which contains y shell-nodes that have the most neighbors in $\preceq$ after it. Then we only select the node pair from $X \cup Y$. x and y are set by users to control computational cost. In addition, according to Lemma 3.1, the upper bound for expanding k-core is the size of $(k-1)$-shell. The merger will enlarge or decrease the size of $(k-1)$-shell, so we need to consider them both to retain or increase $(k-1)$-shell nodes as more as possible. The pseudocode of the algorithm is shown in Algorithm 1.

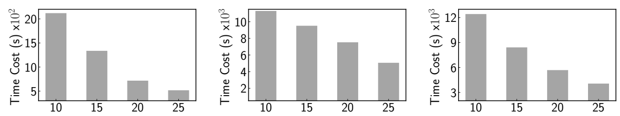
We find the best merge pairs until exhaust budget b (lines 1-10). In each round, we use *max* and *BP* to record the best effect to enlarge k-core and best merge pair respectively (line 2). We first apply core decomposition to compute the $(k-1)$-shell and the corresponding removing order $\preceq$ (line 3). After that, we collect x out-nodes with most $(k-1)$-shell neighbors and y shell-nodes with most neighbors in $\preceq$ after it (lines 4-5). Then, we extract the best node pair *BP* from $X \cup Y$ that can enlarge k-core and $(k-1)$-shell (lines 6-9). At the end of each round, we add *BP* into P and update the graph G after merging *BP* (line 10).



(a) Brightkite (k) (b) Cithepph (k) (c) Gowalla (k)

(d) Brightkite (b) (e) Cithepph (b) (f) Gowalla (b)

Figure 4: Effectiveness evaluation



(a) Brightkite (k) (b) Cithepph (k) (c) Gowalla (k)

(d) Brightkite (b) (e) Cithepph (b) (f) Gowalla (b)

Figure 5: Efficiency evaluation

4 EXPERIMENTS

Algorithms. To the best of our knowledge, there is no existing work for the problem of core maximization by node merging. Thus we implement and evaluate the following algorithms.

- **CMNM**. Our proposed CMNM algorithm (i.e., Algorithm 1).
- **RAND**. In CMNM, we randomly extract x out-nodes and y shell-nodes nodes from V in each iteration.
- **MIN**. In CMNM, we extract x out-nodes and y shell-nodes with most neighbors in $(k-1)$-core.

Datasets and workloads. We employ 3 real-world networks, i.e., Brightkite (58228 nodes and 214078 edges), Cithepph (36692 nodes and 420877 edges) and Gowalla (196591 nodes and 950327 edges). All datasets are publicly available on SNAP (http://snap.stanford.edu). All the programs are implemented in C++. We vary k in $\{10, 15, 20, 25\}$, and vary b in $\{5, 10, 15, 20\}$. The default values of k and b are 15 and 10, respectively. We choose 20 shell-nodes and 20 out-nodes in each round.

Effectiveness evaluation. To evaluate the effectiveness of our method, in this experiment, we report the number of followers of CMNM, RAND and MIN by varying k and b. The results are shown in Figure 4. Note that, in these experiments, we use the default setting for another unchanged parameter. As we can see, CMNM outperforms RAND and MIN on all settings, which demonstrates the effectiveness of our proposed method.

Efficiency evaluation. In this experiment, we evaluate the efficiency of our algorithm CMNM by varying k and b. The results are reported in Figure 5. As we can see, CMNM can finish within a reasonable time on all settings. When k increases, the algorithm runs faster due to the smaller search space. With the increase of b, the response time increase, because we need to perform more iterations to select merger pairs.

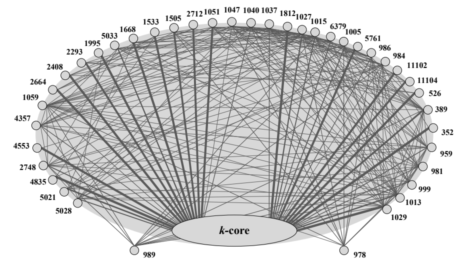


Figure 6: Case study on Brightkite with $k = 20$ and $b = 1$

Case study. We also conduct a case study on Brightkite dataset when $k = 20$ and $b = 1$. As shown in Figure 6, each number represents the node id. And we chose node 989 and node 978 by using our algorithm, which can bring 40 new nodes into k-core.

5 CONCLUSION

In this paper, we discuss the k-core maximization problem by node merging. We prove the problem is NP-hard. To solve the problem, we develop a greedy algorithm and analyze different merging scenarios that may enlarge k-core. Then, we propose the CMNM algorithm which can precisely choose nodes to merge. Experiments are conducted on real-world datasets to demonstrate the advantages of the proposed method.

ACKNOWLEDGMENTS

This work was supported by ARC DP230101445, DP240101322 and ZJNSF LY21F020012. Xiaoyang Wang is the corresponding author.

REFERENCES

[1] Fanchen Bu and Kijung Shin. 2023. On Improving the Cohesiveness of Graphs by Merging Nodes: Formulation, Analysis, and Algorithms. In *SIGKDD*.
[2] Chen Chen, Qiuyu Zhu, Renjie Sun, Xiaoyang Wang, and Yanping Wu. 2021. Edge manipulation approaches for k-core minimization: Metrics and analytics. *TKDE* (2021).
[3] Meenal Chhabra, Sanmay Das, and Boleslaw Szymanski. 2012. Team formation in social networks. In *Computer and Information Sciences III*.
[4] Istabrak Daoud and Racem Mellouli. 2015. Network design and planning with resource pooling: The context of merging two logistics entities. In *SOLI*.
[5] Richard Karp. 1972. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*.
[6] Qingyuan Linghu, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2020. Global Reinforcement of Social Networks: The Anchored Coreness Problem. In *SIGMOD*.
[7] James G. Oxley. 2006. Matroid Theory (Oxford Graduate Texts in Mathematics).
[8] Renjie Sun, Chen Chen, Xijuan Liu, Shuangyan Xu, Xiaoyang Wang, and Xuemin Lin. 2022. Critical nodes identification in large networks: the inclined and detached models. *World Wide Web* (2022).
[9] Renjie Sun, Yanping Wu, and Xiaoyang Wang. 2022. Diversified Top-r Community Search in Geo-Social Network: A K-Truss Based Model. In *EDBT*.
[10] Renjie Sun, Yanping Wu, Xiaoyang Wang, Chen Chen, Wenjie Zhang, and Xuemin Lin. 2023. Clique Identification in Signed Graphs: A Balance Theory based Model. *TKDE* (2023).
[11] Xin Sun, Xin Huang, and Di Jin. 2022. Fast Algorithms for Core Maximization on Large Graphs. In *VLDB*.
[12] Fan Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, and Xuemin Lin. 2017. OLAK: an efficient algorithm to prevent unraveling in social networks. *VLDB* (2017).
[13] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2017. A fast order-based approach for core maintenance. In *ICDE*.
[14] Zhongxin Zhou, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Chen Chen. 2020. K-Core Maximization: An Edge Addition Approach. In *IJCAI*.