Accelerating Community Detection by Using K-core Subgraphs

Chengbin Peng* Tamara G. Kolda† Ali Pinar‡

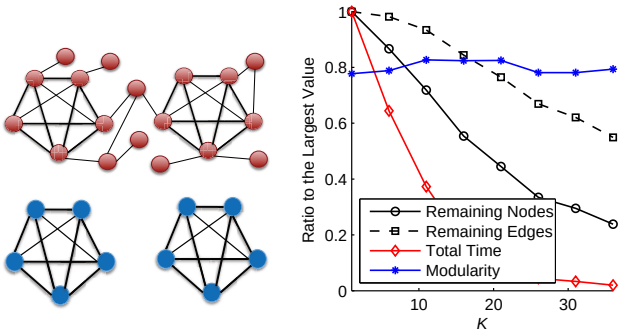
Abstract

Community detection is expensive, and the cost generally depends at least linearly on the number of vertices in the graph. We propose working with a reduced graph that has many fewer nodes but nonetheless captures key community structure. The K-core of a graph is the largest subgraph within which each node has at least K connections. We propose a framework that accelerates community detection by applying an expensive algorithm (modularity optimization, the Louvain method, spectral clustering, etc.) to the K-core and then using an inexpensive heuristic (such as local modularity maximization) to infer community labels for the remaining nodes. Our experiments demonstrate that the proposed framework can reduce the running time by more than 80% while preserving the quality of the solutions. Recent theoretical investigations provide support for using the K-core as a reduced representation.

1 Introduction

Understanding community structure in a graph is a major algorithmic challenge, with a wide spectrum of applications across many disciplines. Different definitions of community can be proposed, but an essential notion is that nodes within the same community are highly connected to one another. Many real world networks have inherent community structure, including social networks, transportation networks, biological networks, etc. [11, 12, 33]. Community structure has many applications, such as in network dynamics where it is used to understand the spreading of a disease [28]. Many aspects on community detection problems have also been explored recently, such as the impact of the degree assortativity [3], the contribution of node attributes [35], and the detection for community outliers [8].

There are many algorithms developed for the community detection problem [7], and here we focus on non-overlapping community detection without node attributes. Example methods include Bayesian methods that maximize the likelihood of a stochastic blockmodel [13], spectral algorithms [20], and fast algorithms using modularity optimization [1, 18] . These methods all share a common feature: their expense is *at least* linear in the number of nodes and oftentimes much more expensive. Our goal is to work with a reduced representation that has fewer nodes but nonetheless maintains the community structure of the original.



(a) Reduction to 4-core removes spurious edges and nodes

(b) Sample results

Figure 1: Effectiveness of K-core subgraph community detection

The fundamental observation of this paper is that community structure is generally preserved in the K-core for reasonable values of K. Recall that the K-core is the largest subgraph in which each node has at least K edges [6]. For example, to compute the 4-core: remove all nodes of degree less than 4, update the degrees of all remaining nodes, and repeat until only nodes of degree 4 or higher remain. See Fig. 1a for an example of a graph and its 4-core. Algorithms for finding the K-core are efficient and amenable for parallelization [29]. We propose to run community detection *only on the much smaller K-core* and then use a fast heuristic to find community labels for the remaining nodes.

In Fig. 1b, we show sample results for the ego-Facebook network (4k nodes, 88k edges). This is an interesting example because the network is fairly dense, so community structure is retained for fairly large values of K. We use modularity optimization [21] as an exemplar community detection method. For $K = 1$, we see the results of applying the community detection method to the full network, and this serves as the baseline. The runtime is 876 seconds. As we increase K, the number of nodes and edges decrease so that we have 20% of nodes and 50% of edges remaining for

*King Abdullah University of Science and Technology, KSA, chengbin.peng@kaust.edu.sa

†Sandia National Laboratories, CA, tgkolda@sandia.gov

‡Sandia National Laboratories, CA, apinar@sandia.gov

$K = 40$. The number of edges does not decrease as quickly as the number of nodes since we are removing low-degree nodes. Because modularity optimization is an expensive procedure, the time reduces faster even than the number of nodes, to less than 20% of the total time for the baseline. The number of communities is determined automatically, but the highest number of communities found in this example is 14. We measure the quality in terms of modularity. In this case, the solution is always at least as good as the baseline.

1.1 Our Contributions

- **General K-core Community Detection Framework**: We propose a fairly intuitive but nonetheless novel method for accelerating community detection by focusing the bulk of the effort on the K-core. We suggest a fast heuristic for providing labels to nodes outside the K-core.
- **Time Reduction & Quality Preservation**: We show positive results (similar to Fig. 1b) for a variety of popular community detection algorithms on various real-world data sets. Overall, we find that we get substantial decreases (up to 90%) in runtime without sacrificing quality, measured in terms of modularity, normalized cut, and NMI.
- **Theoretical Justification**: We show how recent theoretical developments support the idea that the K-core preserves the community structure and also how this understanding can be used to select a good value for K.

1.2 Related Work Much research has been devoted to community detection algorithms [19, 7]. However, scalability is a major issue. One strategy is to split the graph into very small subgraphs and optimize each small problem locally [24, 14], in which case the quality is usually compromised. Alternatively, another strategy is to include additional phases for global optimization [18, 1, 9], but the running time becomes longer.

Perhaps the most closely related work to our focuses on removing unimportant edges [30]. However, this method preserves all the nodes and at least one edge per node, so it requires input graphs with relatively high average degree to achieve a good performance.

Scalability can also be achieved via high-performance implementations such as multi-threaded algorithms [25, 2, 36]. Some other fast algorithms rely on additional information, such as node attributes [27, 32]. Our K-core framework is compatible with these improvements.

Some papers discuss the existence condition of a K-core subgraph [15, 26]. Some study the relationship between the K-core and the percolation dynamics [31], and it is worth noting that the bond percolation is quite similar to the label propagation in community detection [24]. Recently, K-core is also used to evaluate the robustness of communities, which reflects the collaborative nature in co-authorship graphs [10].

2 Algorithm

Our framework is compatible with any community detection method and comprises three steps. The first step is to reduce the whole graph to a K-core. The second step uses an existing algorithm to generate community labels for nodes in the K-core. The third step is to use find community labels for the remainder of the graph via a fast algorithm. Algorithm 1 is the pseudocode of the framework; here, G is the original graph, K is the desired core number, G_K is the K-core subgraph, g_K is the labels for the K-core, and g is the labels for all nodes. Note that we discuss the choice of K in Sec. 4.

Algorithm 1 Accelerated K-core Community Detection

```
1: input Graph G, Parameter K
2: output Labels g
3: (1) G_K ← Kcore(G, K)
4: (2) g_K ← CommunityDetection(G_K)
5: (3) g ← Recover(G, g_K)
6: return g
```

2.1 Find the K-core Subgraph There are many algorithms available to find the K-core of a graph [29, 15]. A standard approach [15] is to traverse all the nodes and remove those connected by less than K edges. The edges connected to the removed nodes are removed as well. The traversal and removal may be repeated multiple times, until all the remaining nodes have degrees at least K. The pseudocode is presented in Algorithm 2.

Algorithm 2 Algorithm to Find the K-core Subgraph

```
1: input Graph G, Parameter K
2: output Subgraph G_K
3: G_K ← G
4: while G_K is not a K-core do
5:    Find nodes in G_K whose degree is less than K
6:    Remove those nodes and their incident edges
7:    Update the node degrees for the remaining nodes
8: end while
9: return G_K
```

2.2 Apply a Community Detection Algorithm on the K-core Applying a community detection algorithm to the K-core is the main part in our framework, and any existing algorithm can be used here. The biggest performance gains will be for algorithms with high quality and high time complexity.

The main algorithm detects the community structure only for nodes in the K-core, which is the main reason for the running time reduction. Here we discuss briefly the effect of our framework on different algorithms. For algorithms with high quality and fairly fast running time, our framework can reduce the running time significantly with the quality preserved. Those algorithms include the Louvain method [1], Martelot's method [18], and modularity optimization [21]. For very fast algorithms such as label propagation [24], our framework does not improve the running time except for very large graphs. Some algorithms like the Bayesian method [13] and spectral clustering [34] need to specify a community number before running, but our framework can increase the community number (in the "recover" step) from the predefined value and to improve the quality.

We note that any method may be used for community detection, though each may rely on a different merit function. This is discussed further in the next subsection.

2.3 Recover the Community Structure for the Whole Graph When the community structure g_K of the K-core is known, the third step is to recover the node labels for all the nodes in and outside the K-core. With the knowledge of g_K, it becomes possible to use a simple algorithm to achieve high quality. In fact, the exact details of the recovery algorithm do not have much impact on the overall success of the method. Low-degree nodes are generally not critical for community structure.

The recovery step is composed of two stages: inferring community labels for nodes outside the K-core and optimizing the structure for the whole graph, as shown in Algorithm 3.

Algorithm 3 Algorithm to Find the Community Structure of the Whole Graph

```
input Graph G, Labels g_K
output Labels g
s ← set of unlabeled nodes, roughly sorted in descending order according to the proportion of their neighbors inside the K-core
repeat
  for all nodes i ∈ s do
    g(i) ←plurality vote of labeled neighbors of i
    Remove i from s
  end for
until no more nodes can be labeled
g ← SecondStageAlgorithm(A, g)
return g
```

In the first stage, we sort the unlabeled nodes in a descending order according to the ratio of their labeled neighbors. Sorting is necessary to make sure that the community structure information of the K-core is sufficiently utilized, so it runs at the beginning of this stage.

A sorting algorithm typically has a time complexity of $O(n \log n)$. However it can be reduced to $O(n)$ by using a coarse sorting instead. Since the ratio of neighboring nodes is in the interval of $[0, 1]$, in coarse sorting we can divide the interval into a number of evenly-sized bins and sort accordingly. For instance, if we have 10 bins where bin 1 has the highest ratio (i.e., (0.9,1]) and bin 10 has the lowest ratio (i.e., [0,1.0]) the nodes with ratios 0.85, 0.5, and 0.45 and put into bins 2, 6, and 6, respectively.

By the order from the sorting above, we assign community labels of the unlabeled nodes according to their most frequent neighboring label (ties are broken randomly). Each assignment may impact subsequent labels. Additionally, we repeat the procedure (without resorting) until we have labeled every connected node. Any nodes that are not connected to the K-core are unlabeled. Each unlabeled node is put into its own single-node community.

In the second stage, we optimize the community structure for the whole graph. Although it can be fast by repeatedly adjusting the labels of nodes according to their most frequent neighboring labels using label propagation [24], we propose a local modularity optimization algorithm to produce a better quality. This method updates the labels according to the modularity gain by the following analysis.

The modularity [4] is defined as follows. Let A be the adjacency matrix of G and m be the number of edges. Define $e_r = \frac{1}{2m} \sum_{ij} A_{ij} \delta(g(i), r) \delta(g(j), r)$ to be the fraction of edges that join vertices within community r and $a_r = \frac{1}{2m} \sum_{ij} A_{ij} \delta(g(j), r)$ to be the fraction of edges that are attached to vertices in community r. The modularity is

$$Q = \sum_r (e_r - a_r^2). \tag{2.1}$$

Therefore, when a node i is moved from community r to community s ($r \neq s$), the change of modularity is as

follows.

$$\begin{aligned}\Delta Q =& Q^{(2)} - Q^{(1)} \\ =& \sum_i [e_i^{(2)} - (a_i^{(2)})^2] - \sum_i [e_i^{(1)} - (a_i^{(1)})^2] \\ =& [(e_r^{(1)} - d_r) - (a_r^{(1)} - d)^2] \\ &+ [(e_s^{(1)} + d_s) - (a_s^{(1)} + d)^2] \\ &- [e_r^{(1)} - (a_r^{(1)})^2] - [e_s^{(1)} - (a_s^{(1)})^2] \\ =& (-d_r + d_s) + 2d(a_r^{(1)} - a_s^{(1)}) - 2d^2. \end{aligned} \tag{2.2}$$

where $d_r = \frac{1}{2m}[\sum_{j \neq i}(A_{ij} + A_{ji})\delta(g(j), r) + A_{ii}]$ and $d = \frac{1}{2m}\sum_j A_{ij}$. For undirected graphs without self-loops, $d_r = \frac{1}{m}\sum_j A_{ij}\delta(g(j), r)$.

In local modularity optimization, we traverse all the nodes for multiple rounds, and compute the potential ΔQ for each node over all the possible community labels. If the maximum potential ΔQ for a node is larger than zero, we change the community label of that node correspondingly; otherwise, the label of the node remains unchanged.

There is somewhat of a mix of objectives in the different steps of our method. In Step 2, the community detection optimizes according to the selected method. But in Step 3, the recovery first uses label propagation as a criteria and then switches to optimizing modularity. It is certainly possible to make all the steps consistent, but we have selected very inexpensive heuristics for Step 3. Perhaps surprisingly, our experiments show that quality is not negatively impacted by this mixture of objective functions. It may be due to the fact that they are different ways of measuring the same fundamental structure.

2.4 Time Complexity The running time of the algorithm in the first and third step grows linearly with respect to the size of the graph. Let b be some constant. Let the graph size be N, and a corresponding K-core subgraph size be N_K. If a community detection algorithm has time complexity $O(N^b)$ to find the community structure, it only cost $O(N_K^b)$ in the second step of our framework. Thus, the complexity of the algorithm using our framework is reduced to $O(N_K^b + N)$. Therefore, the speedup is $(\frac{N}{N_K})^b$ asymptotically.

3 Experiments

In this section, we demonstrate that, on both synthetic and real world data, our approach can reduce the computing time significantly, while the quality of the detected communities changes little and sometimes even becomes better.

Table 1: Data Sets

Data Set Name	Number of Nodes	Number of Edges
com-Youtube	1,134,890	2,987,624
com-Amazon	334,863	925,872
Email-Enron	36,692	367,662
ego-Facebook	4,039	88,234
ca-AstroPh	18,772	396,160
ca-CondMat	23,133	186,936
ca-GrQc	5,242	28,980
ca-HepPh	12,008	237,010
ca-HepTh	9,877	51,971
oregon1_010331	10,670	22,002
oregon1_010421	10,859	22,747
oregon1_010428	10,886	22,493

3.1 Data The synthetic graph (LFR_N1e4) is generated by LFR benchmark [16] using the following parameters: 10000 nodes with average degree 10 and maximum degree 50, mixing parameter of 0.1, exponent of degree distribution = -2, exponent of community size distribution = -1, and the minimum and maximum community sizes as 20 and 200.

The real world graphs listed in Table 1 are from the Stanford Large Network Dataset Collection at SNAP[1]. The ego-Facebook network also has community assignments available.

3.2 Implementation Details All the results are produced in Matlab R2013b on a Dell Precision T7500 with Dual Intel Xeon X5650 2.66GHz Six-core CPU and 48GB of RAM. The results are averaged from five runs.

To demonstrate the effectiveness and flexibility of our framework, we provide results for a variety of community detection algorithms. The details of the methods and implementations is given below. The methods automatically determine the number of communities except for the last two as noted.

- *Louvain method* [1]: Initializes each node as a single community, and shifts the community label of each node according to the modularity gain, until the labels converge. Then, it considers each community as a node to merge some of them again according to the modularity gain. We use the MATLAB implementation called `cluster_jl` by A. Scherrer with default settings[2].
- *Martelot's method* [18]: A multi-scale algorithm with two phases, optimizing a global and a local criterion respectively. In our experiment, we use

[1] SNAP: Stanford Network Analysis Platform, `http://snap.stanford.edu/`

[2] `http://perso.uclouvain.be/vincent.blondel/research/Community_BGLL_Matlab.zip`

the modularity criterion. We use the MATLAB implementation called `fast_mo` by E. Le Martelot[3].

- *Modularity optimization* [21]: An agglomerative method that merges nodes into bigger and bigger communities hierarchically, using the modularity criterion. We use the MATLAB implementation called `fast_newman` by E. Le Martelot[3].
- *Label propagation* [24]: This is a fast method that simply updates the label of a node according to the plurality vote of its neighbors (according to their labels, randomly breaking ties). It runs until the labels cease changing (or 1000 iterations). It is appropriate for large networks, though the quality is usually compromised. Also, the update order impacts the results, so in the experiments run for five times using random orders, and the results are averaged. We implemented this algorithm in MATLAB ourselves.
- *Bayesian method* [13]: A variational approach solves the parameter inference problem in a stochastic block model. We use the MATLAB implementation called `vbmod_restart` by J. Hofman[4]. In this case, the number of communities must be specified by the user, which is a disadvantage but not relevant for our discussion here. We use the results of the prior experiments to come up with reasonable starting values[5].We set `opt.NUM_RESTARTS` to be one (default is ten) in order for the code to run relatively quickly.
- *Spectral clustering* [34]: Considers the graph as a similarity matrix, and solves a data clustering problem where each cluster is a community. It contains two phases. First, it maps the data onto a lower dimensional space formed by eigenvectors of the Laplacian matrix; second, it uses k-means to cluster the reduced data. We use the MATLAB implementation `SpectralClustering` by I. Buerk[6] with the Jordan/Weiss normalization scheme (`type=3`), except we have modified it to run the K-means step ten times and choose the solution that minimizes the sum of the within-cluster sums of point-to-centroid distances. Like for the Bayesian method, the user needs to specify an appropriate number of communities in advance, and we use the same values again[5].

In the recovery step of our algorithm (Step 3), the maximum number of iterations for labeling nodes is set to ten, and the maximum number of modularity optimization iterations is set to 10.

3.3 Results and Analysis — Synthetic Data

Fig. 2 shows performance analysis using the synthetic LFR graph. In this graph, the minimum degree is 4; therefore, the 4-core is the same as the 1-core which is the original graph, so all the results for $K \leq 4$ are identical.

Fig. 2a provides a few different bits of information. The black lines show the proportion of nodes and edges in the K-core as K increases. The 7-core contains only 40% of the nodes and edges of the complete graph. The red lines in Fig. 2a give a sense of the breakdown of the runtime for a specific method (Louvain method [1]). The diamonds show the time relative to the longest time, in this case for the full graph. The other lines show the proportion of that time that went to each step. Observe that Step 2 takes the majority of the time because it has higher complexity, whereas Steps 1 and 3 take only a small proportion of the time, growing as the number of nodes outside the K-core increases.

Fig. 2b shows the total run time for four different algorithms in Step 2. Here we only consider the methods that automatically determine the number of communities. The run times are relative to the longest run time for that method and are not directly comparable *across* the different methods. When $K = 7$, the total running time is reduced by 75–97% for all methods except label propagation. The label propagation is extremely efficient, so it is hard to improve its run time for small graphs.

In terms of quality, we consider the modularity [4, 22] in Fig. 2c. Observe that the modularity is nearly unchanged from 0.9 as K increases.

To give another view of quality, we consider two cases for NMI [5] in Fig. 2d and Fig. 2e. Fig. 2d compares the results to ground truth (i.e., the true clusters). Again, the value is little changed for $K = 5, 6, 7$, though here there is some small decrease. If we did not know the ground truth, we could consider the NMI with respect to the original solution as shown in Fig. 2e. While a negative result is not damning, an NMI near 1 w.r.t the baseline result will indicate no major changes in the solution. We use the definition of NMI in Ref. [5]:

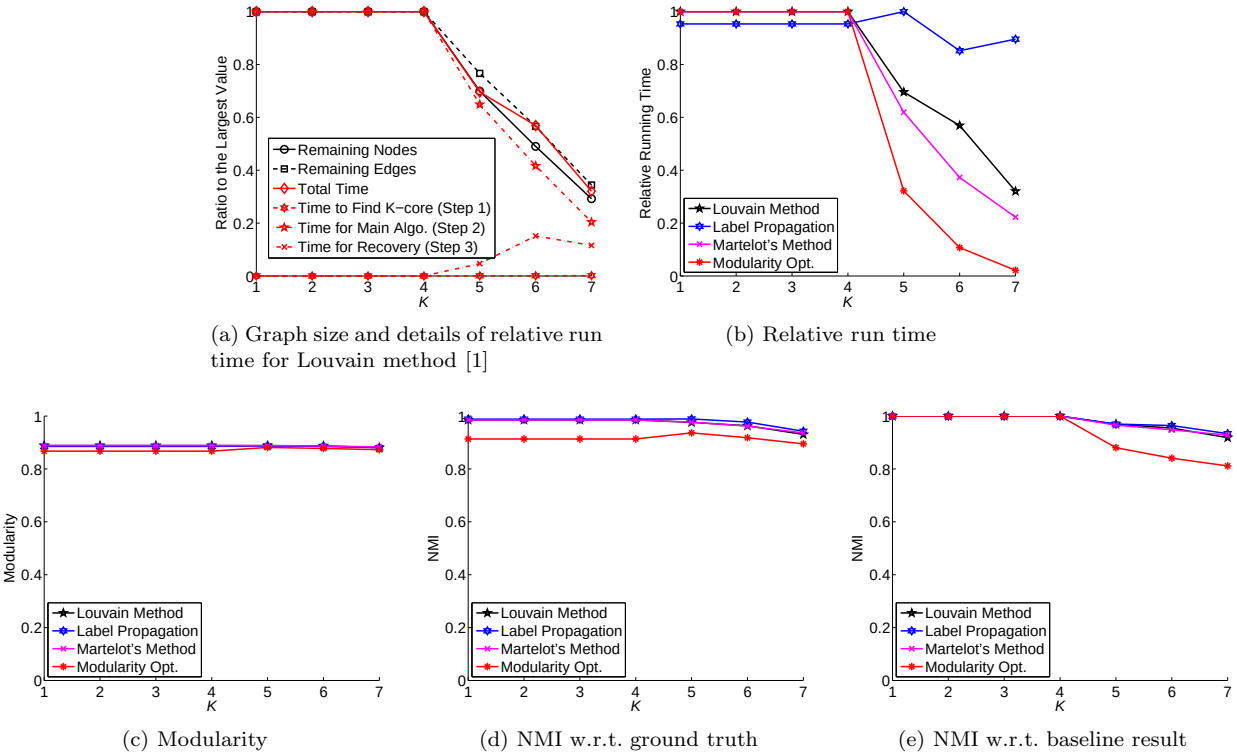
$$I(A,B) = \frac{-2\sum_{i=1}^{c_A}\sum_{j=1}^{c_B} N_{ij}\log(N_{ij}N/N_{i\cdot}N_{\cdot j})}{\sum_{i=1}^{c_A} N_{i\cdot}\log(N_{i\cdot}/N) + \sum_{j=1}^{c_B} N_{\cdot j}\log(N_{\cdot j}/N)}, \tag{3.3}$$

[3] `http://www.elemartelot.org/index.php/programming/cd-code`

[4] http://vbmod.sourceforge.net/

[5] Specified number of communities: Email-Enron=485, ego-Facebook=10, ca-AstroPh=604, ca-CondMat=1835, ca-GrQc=391, ca-HepPh=608, ca-HepTh=904, oregon1_XXXXXX=30.

[6] `http://www.mathworks.com/matlabcentral/fileexchange/34412-fast-and-efficient-spectral-clustering/content/files/SpectralClustering.m`



(a) Graph size and details of relative run time for Louvain method [1]

(b) Relative run time

(c) Modularity

(d) NMI w.r.t. ground truth

(e) NMI w.r.t. baseline result

Figure 2: Results of K-core acceleration on LFR_N1e4 (artificial LFR network)

where the number of real communities is denoted c_A, the number of found communities is denoted c_B, N_{ij} is the number of nodes in the real community i that appear in the found community j, the sum over row i of matrix N_{ij} is denoted $N_{i\cdot}$, and the sum over column j is denoted $N_{\cdot j}$.

3.4 Results and Analysis — Social Network Data Fig. 3 shows analogous results to Fig. 2, except that it is on the ego-Facebook graph, in which we consider the ego networks constituting the graph as the ground-truth communities.[7]

In Fig. 3a, the highest runtime for the Louvain method is actually for $K = 7$, approximately 25% more expensive than for the full graph. However, as K continues to increase, the runtime reduces to less than 50% of the runtime for the full graph. Once again, Step 2 requires the majority of the computation time but eventually reduces for the smaller K-cores.

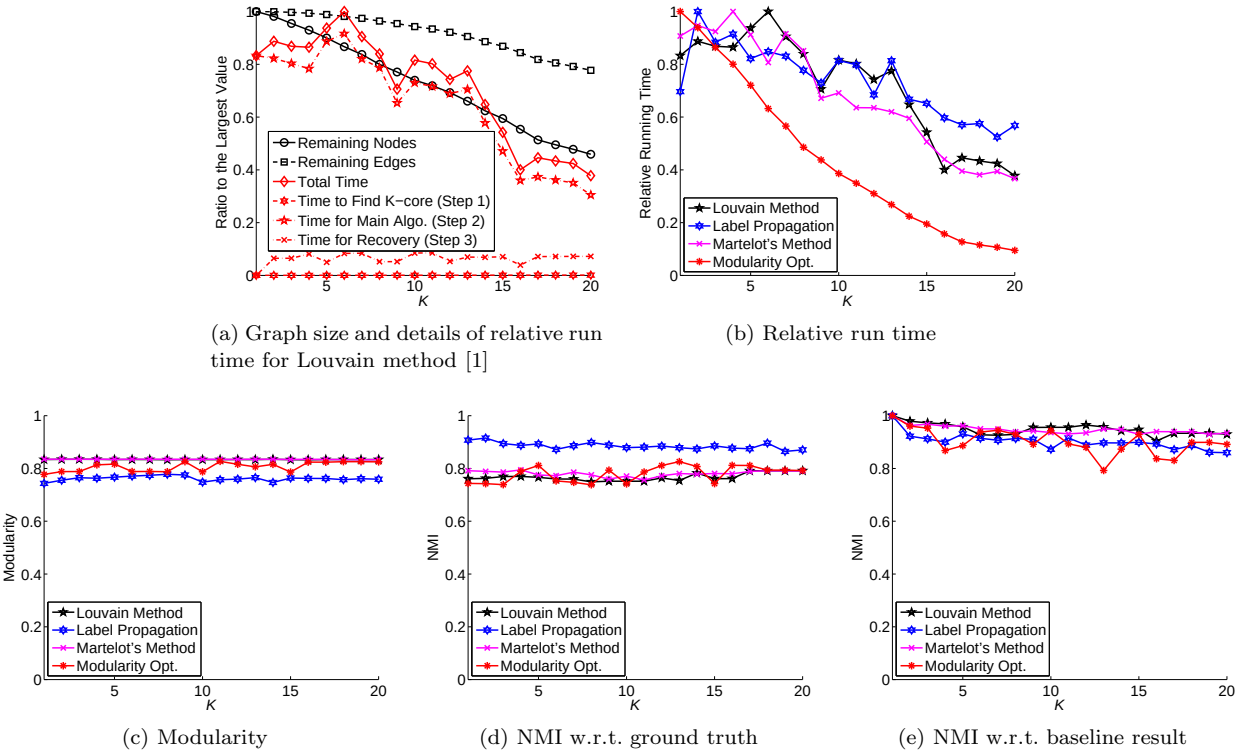
In Fig. 3b, we see that all but one of the methods have some fluctuation in runtimes for lower values of K. The reason is that, as K increases, the number of iterations may change even though the cost per iteration is less. For example, the number of iterations in the Louvain method [1] increases from 9 to 11 when K is increased from 1 to 5. Nevertheless, the total running time will decrease eventually. Therefore, although the running time fluctuates within a small range, the general trend is decreasing as K grows.

In terms of quality, the Figs. 3c–3e show no degradation. The NMI w.r.t. the baseline in Fig. 3e shows that there is little change w.r.t. the original solution as well. We also note that label propagation has the lowest score with respect to modularity but the highest score w.r.t. NMI compared to ground truth.

3.5 Results and Analysis — Large-scale networks Fig. 4 demonstrates how our K-core framework accelerates the (linear-time) label propagation algorithm [24] on com-Youtube and com-Amazon data. Since these are large graphs, we skip the modularity optimization (i.e., the second stage algorithm) in Step (3). Any unlabeled nodes are assigned to singleton communities.

The plots on the left of Fig. 4 show that the quality (based on modularity) is unchanged as K increases. The plots on the right of Fig. 4 show details on the

[7] We have access to overlapping community information. If a node is in multiple communities, we assign it to the community to which is has the most connections, breaking ties randomly.



(a) Graph size and details of relative run time for Louvain method [1]

(b) Relative run time

(c) Modularity

(d) NMI w.r.t. ground truth

(e) NMI w.r.t. baseline result

Figure 3: Results of K-core acceleration on ego-Facebook

method. In these cases, the label propagation algorithm used in Step 2 is so inexpensive that the cost of Step 3 (even without the modularity optimization) eventually dominates the total time. Nevertheless, the overall run time is significantly reduced even when the original algorithm is only linear!

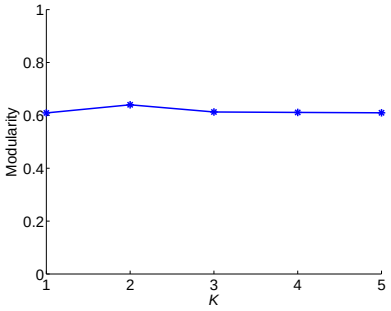
Louvain method [1] is also a linear time algorithm , but the average running time for each node is much longer than label propagation. Thus, we only run those algorithms with $K \geq 3$ on the com-Amazon network. By Fig. 4, the ratio of remaining nodes drops from 76% in 3-core to 50% in 4-core, 17% in 5-core, and 0.2% in 6-core. In Table 2, the running time column contains the running time in seconds, and those in the bracket behind are the relative running times with respect to 3-core. We show that by our approach, the modularity computed through both algorithms does not change much from 3-core to 5-core, but the running times have significant reduction. At 6-core, because the remaining nodes are too few, the modularity is no longer a satisfactory.

3.6 Results and Analysis — Multiple real-world networks Finally, we consider a collection of results on nine more real-world graphs. In Fig. 6, 7 and 5, label propagation [24] is omitted because the quality is usually much worse than other algorithms. Additionally, the Bayesian method [13] and modularity optimization [21] are omitted in two comparisons because they run too slowly.

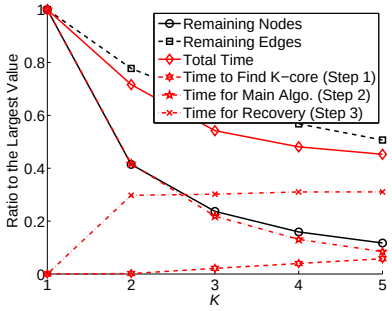
Table 2: Algorithms Using K-core on com-Amazon network

K	Modularity	Time (Relative to 3-core)
3	0.923	6.3e4 (1.00)
4	0.917	2.6e4 (0.41)
5	0.890	3.2e3 (0.05)
6	0.788	5.5e2 (0.01)

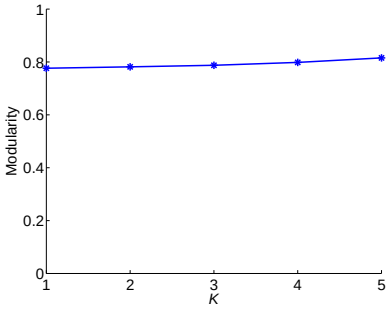
In Fig. 5, the maximum running time for each algorithm is normalized to one. Generally, the overall run time is substantially reduced. The main exception is that the runtime increases for spectral clustering on the oregon networks (which are quite small and sparse). Otherwise, the running time reduction can be as much as 80% for $K = 2$ and over 95% for larger values of K. The small fluctuations in running time are due to the same phenomenon as we encountered for Facebook (Fig. 3b) where the number of iterations in the clustering method changes for some reason.



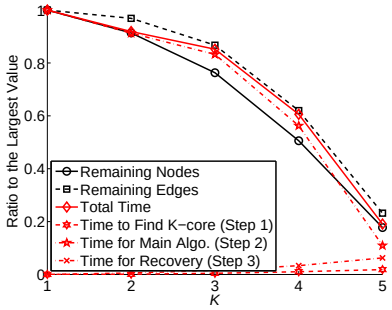
(a) Modularity for com-Youtube



(b) Graph size and details of relative run time for label propagation on com-Youtube



(c) Modularity for com-Amazon



(d) Graph size and details of relative run time for label propagation on com-Amazon

Figure 4: Results of K-core acceleration on com-Youtube and com-Amazon for Community Detection

In terms of quality, Fig. 6 shows the modularity as K increases. Generally, the modularity is little changed and sometimes even improves. For instance, the time to run Modularity Optimization on ca-HepTh is over 1000 seconds on the full graph, but that time reduces by 95% for $K = 4$ with no reduction whatsoever in modularity score.

For another quality measure, we show the normalized cut [19] in Fig. 7 (lower is better). Once again, we see almost no degradation in quality even when runtimes are reduced by 50-90% or more.

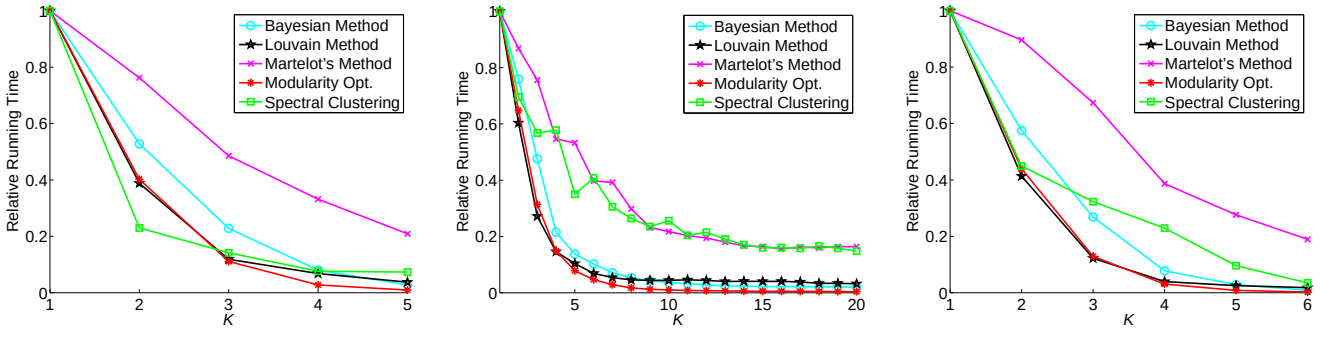
Table 3 lists the ratio of remaining nodes in K-core. With Fig. 6, it demonstrates empirically that if the remaining ratio in K-core is above 50%, the modularity obtained by our framework is almost invariant, and if the remaining ratio is above 20%, the modularity decreases slightly. In all those cases, the running time reduces significantly as the number of remaining nodes decreases.

Table 3: Remaining Nodes in K-core in Multiple Graphs

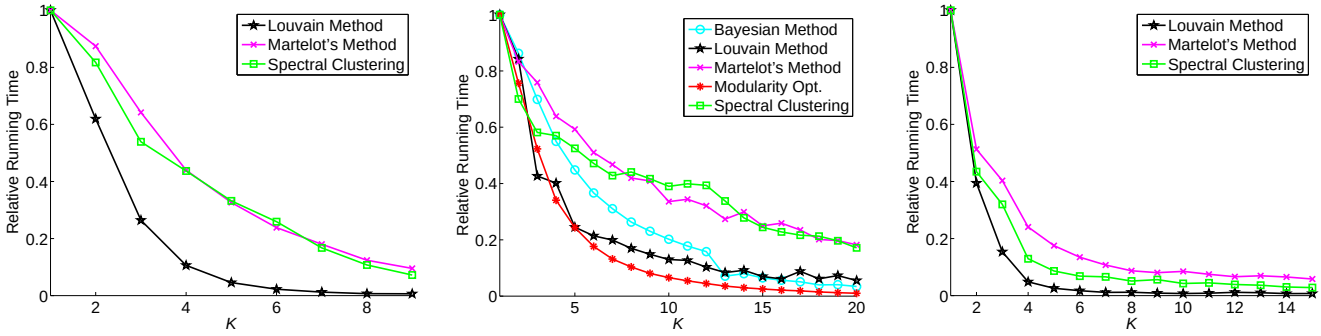
Data Set Name	K (Remaining Ratio)
Email-Enron	3 (0.58), 6 (0.25), 9 (0.14)
ca-AstroPh	5 (0.66), 10 (0.44), 20 (0.23)
ca-CondMat	4 (0.58), 6 (0.35), 8 (0.19)
ca-GrQc	3 (0.50), 4 (0.30), 5 (0.17)
ca-HepPh	5 (0.45), 10 (0.23), 20 (0.16)
ca-HepTh	3 (0.52), 4 (0.32), 5 (0.21)
oregon1_010331	2 (0.64), 3 (0.20), 4 (0.08)
oregon1_010421	2 (0.65), 3 (0.21), 4 (0.09)
oregon1_010428	2 (0.65), 3 (0.20), 4 (0.08)

4 Theoretical Analysis

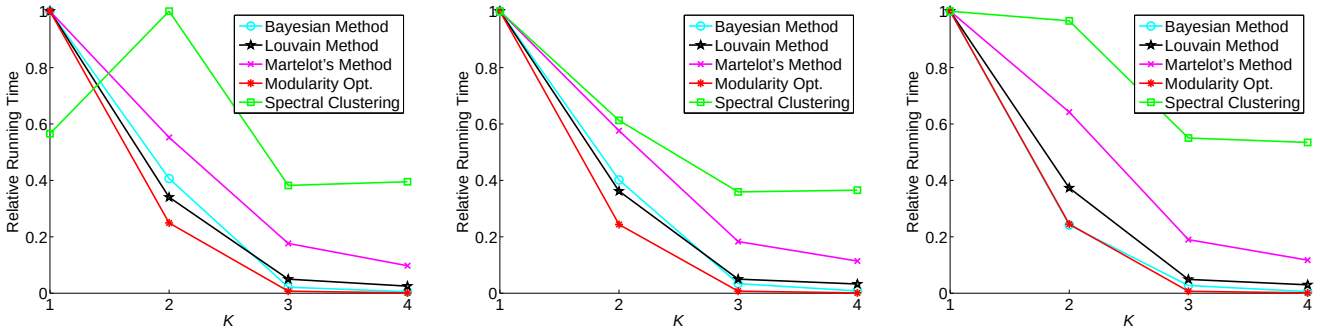
In this section we will discuss the theoretical basis of our proposed method. The essence of our approach is that the dense communities are preserved after a K-core reduction, while the size of the graph reduces drastically. Subsequently, we can identify the same communities with much less effort. The reduction is graph size after K-core reductions is widely studied and empirically documented in our experimental results.



(a) ca-GrQc (b) ca-HepPh (c) ca-HepTh

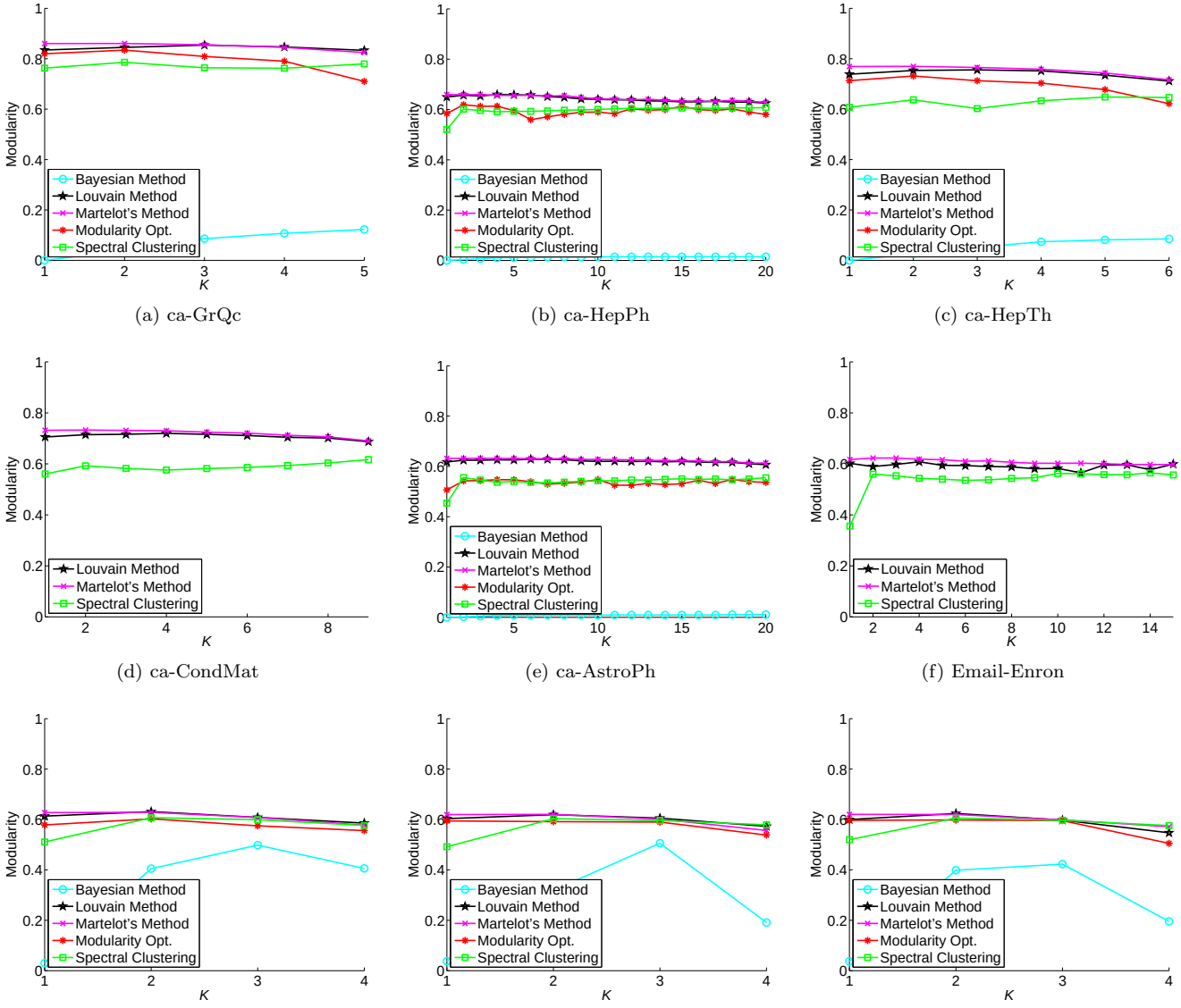


(d) ca-CondMat (e) ca-AstroPh (f) Email-Enron



(g) oregon1_010331 (h) oregon1_010421 (i) oregon1_010428

Figure 5: Relative run time over different K's

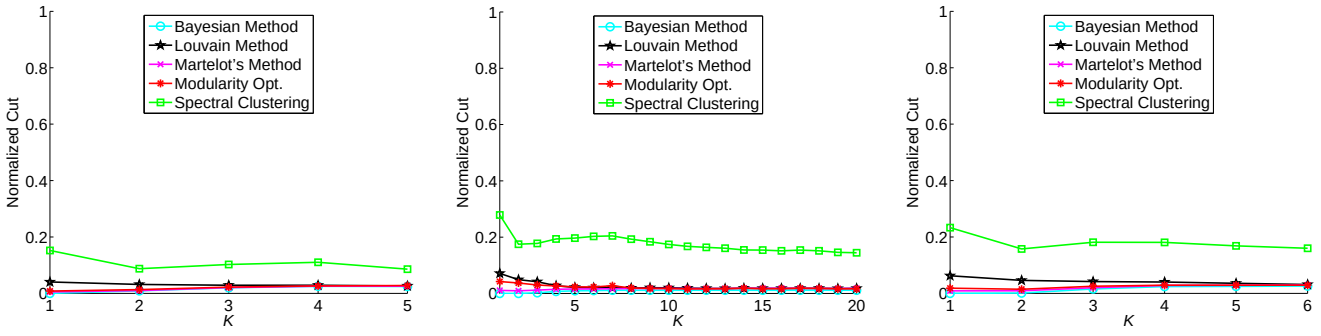


(a) ca-GrQc (b) ca-HepPh (c) ca-HepTh

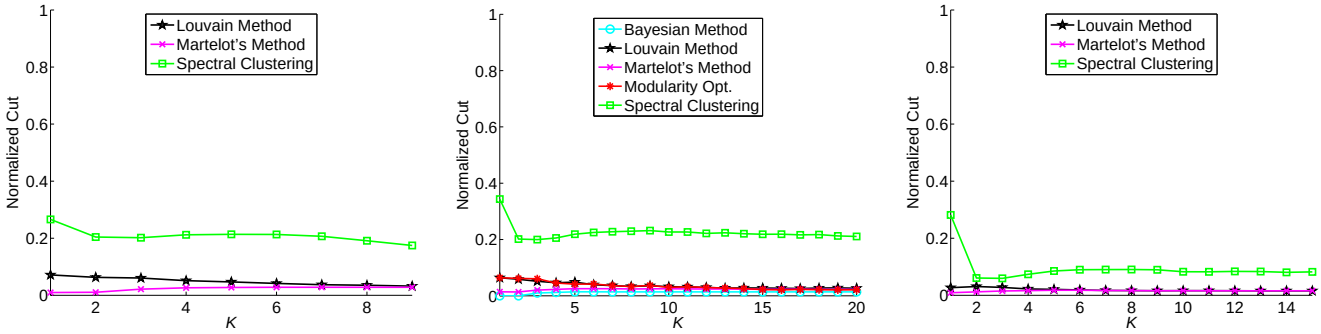
(d) ca-CondMat (e) ca-AstroPh (f) Email-Enron

(g) oregon1_010331 (h) oregon1_010421 (i) oregon1_010428

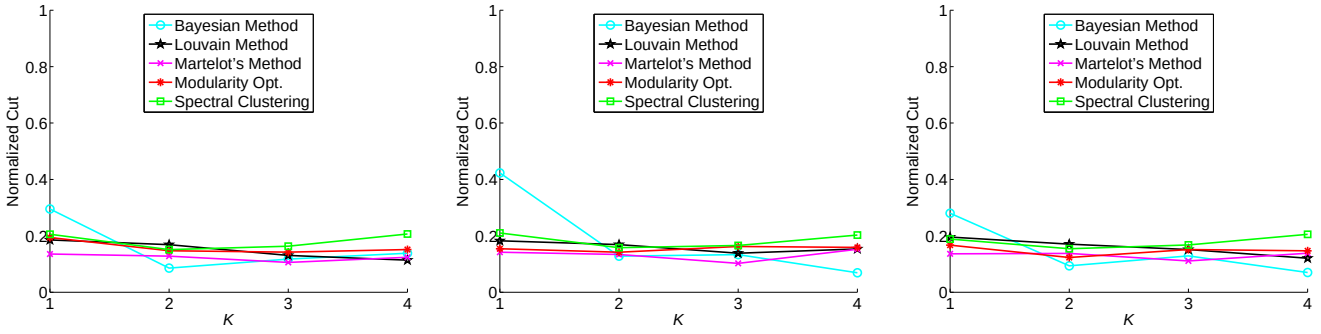
Figure 6: Modularity over different K's



(a) ca-GrQc (b) ca-HepPh (c) ca-HepTh



(d) ca-CondMat (e) ca-AstroPh (f) Email-Enron



(g) oregon1_010331 (h) oregon1_010421 (i) oregon1_010428

Figure 7: Normalized cut over different K's

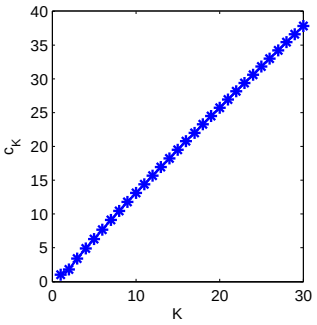


Figure 8: Average degree (c_K) that ensures a K-core exists in an ER subgraph with high probability.

But what happens to the communities?

For the analysis, we assume each community contains a subgraph with an Erdős-Rényi (ER) structure, i.e., all vertices within the ER subgraph are equally likely to be connected. Note that full communities may not have a uniformly random structure because they may contain many nodes of smaller degrees, but for our analysis it is sufficient for a community to just contain a dense ER subgraph of adequate size. Our analysis is based on a beautiful result by Pittel, Spencer, and Wormald [23]. Here we state the key result:

> For the Erdős-Rényi random graph $G(n, m)$ on n vertices with m edges, and for $k \geq 3$, with high probability, a giant K-core appears suddenly when m reaches $c_K n/2$; here $c_K = \min_{\lambda>0} \lambda/\pi_K(\lambda)$ and $\pi_K(\lambda) = P\{\text{Poisson}(\lambda) \geq K-1\}$.

Note that c_K in this result is equal to the average degree of the ER graph. This result shows that an ER subgraph will preserve most of its vertices as long as its average degree *within the ER subgraph* is high enough. Figure 8 shows the c_K values that ensure that a K-core exists with high probability. As seen in this figure, c_K values are quite realistic, and even for $K = 25$ any community with an average degree of around 30 will have a high probability of containing a K-core and hence be preserved.

For our full approach, consider a graph as combination of a random graph (where two edges are connected with a probability directly proportional to the product of the vertex degrees) and a collection of denser ER subgraphs. Note that the first component, the random graph, is what the modularity metric uses as a null hypothesis, and thus maximizing modularity should be equivalent to identifying the dense ER graphs of the second part. Consider applying a K-core reduction on this graph. By the results of Pittel, Spencer, and Wormald [23], we know that dense communities will be preserved with high probability. However, we also know that many vertices and edges of the random graph (specifically vertices that are not part of the communities) will be eliminated. This will result in improved efficiency due to reduced graph size. Moreover, the dense structures will be more pronounced due to eliminations of the noise.

4.1 Selecting K This theoretical understanding leads to motivation for choosing K. As we can see from our experimental results, tightly knit communities like Facebook and the coauthorship networks can have relatively high values of K, like $K = 20$. According to our analysis, an ER subgraph with an average degree of 25 (within the community) has a high probability of containing a 20-core, so it is not surprising that some social networks demonstrate this phenomenon. For the coauthorship networks, coauthors of the same paper form a clique, so fields that have papers with high numbers of authors can have higher values of K. On the other hand, values like $K = 3$ only require ER subgraphs with average degree (within the subgraph) of 3.35. Hence, if communities are of approximately size 10, then number of edges should be 67 for a high probability of a 3-core. When nodes in each community are degree assortative [3], high degree nodes tend to remain in K-core, and we can use a larger K.

In practice, we can choose K simply according to the size of the subgraph. For example, we may choose the largest possible K with the K-core subgraph containing at least 20% of the total nodes. When nodes in each community are degree assortative [3], high degree nodes tend to remain in K-core, and we can use a larger K.

4.2 Node Remaining Ratios in K-core In this subsection, we define the remaining ratio R as the ratio of the nodes in the K-core to all the nodes in the graph, and analyze the relationship between R and K .

First, we define several variables: N, the total number of nodes; K is the parameter for the K-core subgraph; $\mathcal{V}$ is the set of nodes in a community, and $\mathcal{V}^{(t)}$ is the set of remaining nodes after the tth removal by the while loop in Algorithm 2; and $y^{(t)}(k)$ is the probability of being degree k for remaining nodes after the tth removal. Hence, $t = 0$ indicates the status of the original graph, without any removal. $k_{\max}$ is the maximum possible degree of any node in $\mathcal{V}$. $R^{(t)}$ is the remaining ratio of nodes after tth removal, and $R^{(0)} = 1$.

As defined, $y^{(0)}(k)$ should be the degree distribution of the original graph. In each removal, we define the remaining degree D_{rn} introduced by edges between the remaining nodes after this round removal, and the removed degree D_{rm} which is due to the edges between

the remaining nodes and the removed nodes by current removal.

In the $t+1$th removal ($t \geq 0$), the total number of remaining degrees is

(4.4)
$$D_{rn}^{(t+1)} = \sum_{k=K}^{k_{\max}} [k \times y^{(t)}(k)] \times N \times \frac{\sum_{k=K}^{k_{\max}} [k \times y^{(t)}(k)]}{\sum_{k=0}^{k_{\max}} [k \times y^{(t)}(k)]}$$

where $\sum_{k=K}^{k_{\max}} y^{(t)}(k)$ represents the expected ratio of neighboring nodes inside $\mathcal{V}^{(t+1)}$.

The total number of removed degrees is

(4.5)
$$D_{rm}^{(t+1)} = \sum_{k=K}^{k_{\max}} [k \times y^{(t)}(k)] \times N \times \frac{\sum_{k=0}^{K-1} [k \times y^{(t)}(k)]}{\sum_{k=0}^{k_{\max}} [k \times y^{(t)}(k)]}$$

Then, for nodes in $\mathcal{V}^{(t+1)}$, the ratio of remaining degree on average after the $t+1$th removal is

$$r^{(t+1)} = \frac{D_{rn}^{(t+1)}}{D_{rn}^{(t+1)} + D_{rm}^{(t+1)}} = \frac{\sum_{k=K}^{k_{\max}} [k \times y^{(t)}(k)]}{\sum_{k=0}^{k_{\max}} [k \times y^{(t)}(k)]} \quad (4.6)$$

For a node of degree k_c, if edges are randomly kept at probability r, then the probability of having k remaining edges is

$$z^{(t+1)}(k, k_c) = f(k, k_c, r^{(t+1)}) \quad (4.7)$$

where $f(k, k_c, r^{(t+1)})$ is the probability of having k edges successfully kept within a total of k_c edges. Function f is essentially a probability density function of a binomial distribution, and will be used several times in the remaining part of this section.

Therefore, the expected degree distribution after one removal is

$$y^{(t+1)}(k) = \sum_{k_c=K}^{k_{\max}} z^{(t+1)}(k, k_c) y^{(t)}(k_c) \quad (4.8)$$

Finally, we have the expected remaining ratio after $t+1$th removal.

$$R^{(t+1)} = R^{(t)} \times \sum_{k=K}^{k_{\max}} y^{(t)}(k); \quad (4.9)$$

This procedure can run iteratively as t increases and converge asymptotically as r reduces. It is noticeable that $r^{(t+1)}$ in Eq. (4.6) is only determined by the initial degree distribution $y^{(0)}(k)$, so is $R^{(t+1)}$.

4.3 Maximizing the Modularity in K-core In this subsection, we develop our theorem based on a variation of the planted l-partition model [16]. In our model, we assume the nodes in different ground-truth clusters have the same degree distribution and correlation coefficient. For a randomly selected edge, it could be an intra-cluster connection or an inter-cluster connection. If μ is defined as the proportion of the first type edges, we assume μ to be identical for all the ground-truth clusters.

PROPOSITION 4.1. *Under the assumptions above, if the modularity for G_1 is maximized by taking the ground-truth clusters represented by g_1, then the modularity for G_K is also maximized by taking the corresponding subset g_K from g_1.*

Let ρ be the remaining ratio of edges in G_K. Because each cluster has the same degree distribution and correlation coefficient, when K is given, ρ is invariant in different clusters.

We define $E_r = 2me_r$ as total degrees introduced by the edges in cluster r and $A_r = 2ma_r$ as the total degrees introduced by all the edges connecting to nodes in cluster r. Thus, the modularity can be rewritten as

$$\text{mod}\ (G_1, g_1) = \sum_r [\frac{E_r}{2m} - (\frac{A_r}{2m})^2],$$
$$\text{mod}\ (G_K, g_K) = \sum_r [\frac{\rho E_r}{2\rho m} - (\frac{\rho A_r}{2\rho m})^2] = \quad \text{mod}\ (G, g_0).$$

Thus, using the same configuration, the modularity in K-core is the same as in 1-core. As g_1 maximizes the modularity, any other configurations g' can be considered as a series of relabeling certain nodes from one cluster to the another based on g_1, which will reduce the modularity. Without loss of generality, we demonstrate that relabeling n_δ nodes degrees from their ground-truth cluster 'Cluster 1' to a new cluster 'Cluster 2' will reduce the modularity. A_δ is the corresponding number of degrees in K-core associated with those nodes.

After relabeling, the change of intra-cluster degrees is

$$-\frac{\mu A_\delta \frac{\rho A_1 - A_\delta}{\rho A_1}}{2\rho m} + \frac{(1-\mu) A_\delta \frac{\rho A_2}{\sum_r \rho A_r}}{2\rho m}, \quad (4.10)$$

where μA_δ is the intra-cluster degree introduced by edges connecting to the relabeling nodes in ground-truth Cluster 1, and the first term means the loss is proportional to the remaining degrees in Cluster 1. The second term means the degree increase is proportional to the degrees in Cluster 2. On the other hand, the

change caused by the null model is

$$\begin{aligned} & -[(\tfrac{\rho A_1 - A_\delta}{2\rho m})^2 + (\tfrac{\rho A_2 + A_\delta}{2\rho m})^2] \\ & +[(\tfrac{\rho A_1}{2\rho m})^2 + (\tfrac{\rho A_2}{2\rho m})^2] \\ = & \tfrac{A_\delta}{2\rho m}[\tfrac{\rho A_1 - \rho A_2 - A_\delta}{\rho m}]. \end{aligned} \tag{4.11}$$

Therefore, the total change is the summation of Eq. (4.10) and (4.11)

$$\begin{aligned} & \text{mod}\,(G_K, g'_K) - \quad \text{mod}\,(G_K, g_K) \\ = & \frac{A_\delta}{2\rho m}[\mu \frac{\rho A_1 - A_\delta}{\rho A_1} + (1-\mu)\frac{A_2}{\sum_r A_r} \\ & + \frac{\rho A_1 - \rho A_2 - A_\delta}{\rho m}] \\ = & \frac{\frac{A_\delta}{\rho}}{2\rho m}[\mu \frac{A_1 - \frac{A_\delta}{\rho}}{A_1} + (1-\mu)\frac{A_2}{\sum_r A_r} \\ & + \frac{A_1 - A_2 - \frac{A_\delta}{\rho}}{m}] \\ = & \text{mod}\,(G_1, g'_1) - \quad \text{mod}\,(G_1, g_1) \\ \leq & 0 \end{aligned} \tag{4.12}$$

where g'_1 corresponding to the configuration of changing labels of node with $\frac{A_\delta}{\rho}$ edges from Cluster 1 to Cluster 2. $\rho = 1$ corresponds to the graph G_1.

Thus, the proposition is proved. Therefore, if the Step 2 algorithm can maximize the modularity, the computed community for K-core is the same as the ground-truth. As our Step 3 algorithm also maximizes the modularity for the remaining nodes, the ground-truth community structure of the whole graph can be found by our framework.

5 Conclusions

In this work, we introduce a K-core based framework that can accelerate community detection algorithms significantly. It includes three steps: first, find the K-core of the original graph; second, run a community detection algorithm on the K-core; third, find labels for all the nodes outside the K-core and then optimize for the whole graph. Experiments demonstrate efficiency and accuracy on different real graphs, under several quality measurements. Theoretical analysis supports our approach through K-core. The MATLAB code will be made freely available for download. For future work, we consider the case of overlapping communities.

References

[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre, *Fast unfolding of communities in large networks*, Journal of Statistical Mechanics: Theory and Experiment, 2008 (2008), p. P10008.

[2] Zhan Bu, Chengcui Zhang, Zhengyou Xia, and Jiandong Wang, *A fast parallel modularity optimization algorithm (fpmqa) for community detection in online social network*, Knowledge-Based Systems, 50 (2013), pp. 246–259.

[3] Marek Ciglan, Michal Laclavík, and Kjetil Nørvåg, *On community detection in real-world networks and the importance of degree assortativity*, in Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2013, pp. 1007–1015.

[4] Aaron Clauset, Mark EJ Newman, and Cristopher Moore, *Finding community structure in very large networks*, Physical Review E, 70 (2004), p. 066111.

[5] Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas, *Comparing community structure identification*, Journal of Statistical Mechanics: Theory and Experiment, 2005 (2005), p. P09008.

[6] Sergey N Dorogovtsev, Alexander V Goltsev, and Jose Ferreira F Mendes, *K-core organization of complex networks*, Physical Rreview Letters, 96 (2006), p. 040601.

[7] Santo Fortunato, *Community detection in graphs*, Physics Reports, 486 (2010), pp. 75–174.

[8] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han, *On community outliers and their efficient detection in information networks*, in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2010, pp. 813–822.

[9] Ullas Gargi, Wenjun Lu, Vahab S Mirrokni, and Sangho Yoon, *Large-scale community detection on youtube for topic discovery and exploration.*, in Proceedings of the 5th International AAAI Conference on Weblogs and Social Media, 2011.

[10] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis, *Evaluating cooperation in communities with the k-core structure*, in the IEEE International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 2011, pp. 87–93.

[11] Michelle Girvan and Mark EJ Newman, *Community structure in social and biological networks*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 7821–7826.

[12] Roger Guimera, S Mossa, A Turtschi, and LA Nunes Amaral, *The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles*, Proceedings of the National Academy of Sciences, 102 (2005), pp. 7794–7799.

[13] Jake M Hofman and Chris H Wiggins, *Bayesian approach to network modularity*, Physical Review Letters, 100 (2008), p. 258701.

[14] Jianbin Huang, Heli Sun, Yaguang Liu, Qinbao Song, and Tim Weninger, *Towards online multiresolution community detection in large-scale networks*,

PloS one, 6 (2011), p. e23829.

[15] Svante Janson and Malwina J Luczak, *A simple solution to the k-core problem*, Random Structures & Algorithms, 30 (2007), pp. 50–62.

[16] Andrea Lancichinetti and Santo Fortunato, *Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities*, Physical Review E, 80 (2009), p. 016118.

[17] Andrea Lancichinetti, Santo Fortunato, and János Kertész, *Detecting the overlapping and hierarchical community structure in complex networks*, New Journal of Physics, 11 (2009), p. 033015.

[18] Erwan Le Martelot and Chris Hankin, *Fast multi-scale detection of relevant communities in large-scale networks*, The Computer Journal, (2013).

[19] Jure Leskovec, Kevin J Lang, and Michael Mahoney, *Empirical comparison of algorithms for network community detection*, in Proceedings of the 19th International Conference on World Wide Web, ACM, 2010, pp. 631–640.

[20] MEJ Newman, *Spectral methods for network community detection and graph partitioning*, arXiv:1307.7729, (2013).

[21] Mark EJ Newman, *Fast algorithm for detecting community structure in networks*, Physical Review Eeview E, 69 (2004), p. 066133.

[22] Mark EJ Newman and Michelle Girvan, *Finding and evaluating community structure in networks*, Physical Review E, 69 (2004), p. 026113.

[23] Boris Pittel, Joel Spencer, and Nicholas Wormald, *Sudden emergence of a giant k-core in a random graph*, J. Comb. Theory Ser. B, 67 (1996), pp. 111–151.

[24] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara, *Near linear time algorithm to detect community structures in large-scale networks*, Physical Review E, 76 (2007), p. 036106.

[25] Jason Riedy, David A Bader, and Henning Meyerhenke, *Scalable multi-threaded community detection in social networks*, in the 26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), IEEE, 2012, pp. 1619–1628.

[26] Oliver Riordan, *The k-core and branching processes*, Combinatorics, Probability & Computing, 17 (2008), p. 111.

[27] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy, *Efficient community detection in large networks using content and links*, in Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 1089–1098.

[28] Marcel Salathé and James H Jones, *Dynamics and control of diseases in networks with community structure*, PLoS Computational Biology, 6 (2010), p. e1000736.

[29] Ahmet Erdem Sariyüce, Bugra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Umit V Catalyürek, *Streaming algorithms for k-core decomposition*, Proceedings of the VLDB Endowment, 6 (2013).

[30] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan, *Local graph sparsification for scalable clustering*, in Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, ACM, 2011, pp. 721–732.

[31] M Ángeles Serrano and Marián Boguñá, *Clustering in complex networks. ii. percolation properties*, Physical Review E, 74 (2006), p. 056115.

[32] Karsten Steinhaeuser and Nitesh V Chawla, *Community detection in a large real-world social network*, in Social computing, behavioral modeling, and prediction, Springer, 2008, pp. 168–175.

[33] Amanda L Traud, Peter J Mucha, and Mason A Porter, *Social structure of facebook networks*, Physica A: Statistical Mechanics and its Applications, 391 (2012), pp. 4165–4180.

[34] Ulrike Von Luxburg, *A tutorial on spectral clustering*, Statistics and Computing, 17 (2007), pp. 395–416.

[35] Jaewon Yang, Julian McAuley, and Jure Leskovec, *Community detection in networks with node attributes*, in Data Mining (ICDM), 2013 IEEE 13th International Conference on, IEEE, 2013, pp. 1151–1156.

[36] Zhemin Zhu, Chen Wang, Li Ma, Yue Pan, and Zhiming Ding, *Scalable community discovery of large networks*, in the 9th International Conference on Web-Age Information Management, 2008 (WAIM'08), 2008, pp. 381–388.

Place Holder
for
Art Only