AAAI 2014, July 27-31, 2014, in Quebec City, Canada

CORECLUSTER: A Degeneracy Based Graph Clustering Framework

Christos Giatsidis
École Polytechnique
giatsidis@lix.polytechnique.fr

Fragkiskos D. Malliaros
École Polytechnique
fmalliaros@lix.polytechnique.fr

Dimitrios M. Thilikos
CNRS, LIRMM, and UoA
sedthilk@thilikos.info

Michalis Vazirgiannis
École Polytechnique and AUEB
mvazirg@lix.polytechnique.fr

Abstract

Graph clustering or community detection constitutes an important task for investigating the internal structure of graphs, with a plethora of applications in several domains. Traditional tools for graph clustering, such as spectral methods, typically suffer from high time and space complexity. In this article, we present CORECLUSTER, an efficient graph clustering framework based on the concept of graph degeneracy, that can be used along with any known graph clustering algorithm. Our approach capitalizes on processing the graph in a hierarchical manner provided by its core expansion sequence, an ordered partition of the graph into different levels according to the k-core decomposition. Such a partition provides a way to process the graph in an incremental manner that preserves its clustering structure, while making the execution of the chosen clustering algorithm much faster due to the smaller size of the graph's partitions onto which the algorithm operates.

Introduction

Detecting clusters or communities in graphs constitutes a cornerstone problem with many applications in several disciplines. Characteristic application domains include social and information network analysis, biological networks, recommender systems and image segmentation. Due to its importance and multidisciplinary nature, the problem of graph clustering has received great attention from the research community and numerous algorithms have been proposed (see (Fortunato 2010) for a survey in the area).

Spectral clustering methods (e.g., (Ng, Jordan, and Weiss 2001)) impose a high cost of computing resources both in time and space regardless of the data on which it is going to be applied (Fortunato 2010). Other well-known approaches for community detection are the ones based on modularity optimization (Newman and Girvan 2004; Clauset, Newman, and Moore 2004), stochastic flow simulation (Satuluri and Parthasarathy 2009) and local partitioning methods (Fortunato 2010). In any case, scalability is still a major challenge in the graph clustering task, especially nowadays with the significant increase of the graphs' size.

Typically, there are two main methodologies for scaling up a graph clustering method: (i) algorithm-oriented and (ii) data-oriented. The first one considers the algorithm of interest and appropriately optimizes – whenever is possible – the "parts" of the algorithm responsible for scalability issues. Prominent examples here are the fast modularity optimization method (Clauset, Newman, and Moore 2004) and the scalable flow-based Markov clustering algorithm (Satuluri and Parthasarathy 2009). The second and widely used methodology is to rely on sampling/sparsification techniques. In this case, the size of the graph onto which the algorithm will operate is reduced, by disregarding nodes/edges. However, in this approach possibly useful structural information of the graph (i.e., nodes/edges) is ignored.

In this paper, we propose CORECLUSTER, a graph clustering framework that capitalizes on the notion of graph degeneracy – also known as k-core decomposition (Seidman 1983). The main idea behind our approach is to combine any known graph clustering algorithm with an *easy-to-compute*, *clustering-preserving* hierarchical representation of the graph – as produced by the k-core decomposition – towards a scalable graph clustering tool. The k-core of a graph is a maximal size subgraph where each node has at least k neighbors in the subgraph (we say that k is the *rank* of such a core). The maximum k for which a graph contains a k-core is known as its *degeneracy*. We refer to this core as "the densest core". Intuitively, the k-core of such a graph is located in its "densest territories". Based on this idea, we show that the densest cores of a graph are roughly maintaining its clustering structure and thus constitute *good starting points* (seed subgraphs) for computing it. Given the fact that the size of the densest core of a graph is orders of magnitude smaller than that of the original graph, we apply a clustering algorithm starting from its densest core and then, on the resulting structure, we incrementally cluster the rest of the nodes in the lower rank cores in decreasing order – following the hierarchy produced by the k-core decomposition.

The main contributions of this paper are the following:

- *Clustering Framework*: We introduce CORECLUSTER, a scalable degeneracy-based graph clustering framework, that can be used along with any known graph clustering algorithm. We show how CORECLUSTER utilizes the k-core decomposition of a graph in order to (i) select seed subgraphs for starting the clustering process and (ii) expand the already formed clusters or create new ones.

- *Scalability and Accuracy Analysis*: We discuss analytically the ability of CORECLUSTER to scale-up, describing its expected running time. We also justify why the k-core structure captures the clustering properties of a graph, thus being able to indicate good seed subgraphs for a clustering algorithm.
- *Experiments*: We perform an extensive experimental evaluation regarding the efficiency and accuracy of CORECLUSTER, both on synthetic and real-world graphs. The experimental results show that the time complexity is improved by 3-4 orders of magnitude (compared to a baseline algorithm), especially for large graphs. Moreover, for graphs with inherent community structure, the quality of the results is maintained or even is improved.

Related Work

Graph clustering. The problem of community detection and graph clustering has been extensively studied from several points of view. Some well-known approaches include spectral clustering (e.g., (Ng, Jordan, and Weiss 2001; Shi and Malik 2000; White and Smyth 2005)), modularity optimization (e.g., (Newman 2004; Clauset, Newman, and Moore 2004)), multilevel graph partitioning (e.g., Metis (Karypis and Kumar 1998)), flow-based methods (Satuluri and Parthasarathy 2009), hierarchical methods (Newman and Girvan 2004) and many more. A very informative and comprehensive review over the different approaches can be found in (Fortunato 2010). Also, the authors of (Lancichinetti, Fortunato, and Radicchi 2008) have conducted a comparative analysis on the performance of some of the most recent algorithms, in artificial data produced by their parametrized generator of benchmark graphs. In our work, we use the same graph generator as in (Lancichinetti, Fortunato, and Radicchi 2008) to evaluate our framework.

Scaling-up graph clustering. The efficiency of graph clustering can be improved in various ways. Two well-known approaches are the ones of *sampling* and *sparsification*. In the case of spectral clustering, sampling-based approaches include the Nyström method (Kumar, Mohri, and Talwalkar 2009) and randomized SVD algorithms (Drineas et al. 2004). Concerning graph sampling, the goal is to produce a graph of smaller size (nodes and edges), preserving a set of desired graph properties (e.g., degree distribution, clustering coefficient) (Leskovec and Faloutsos 2006). The work by Maiya and Berger-Wolf (Maiya and Berger-Wolf 2010), presents a method – based on the notion of expansion properties – to sample a subgraph that preserves the community structure, i.e., contains representative nodes of the communities. Then, the community membership of the nodes that do not belong to the sample can be expressed as an inference problem. Unlike the aforementioned methods that sample both nodes and edges, the graph sparsification algorithm presented in (Satuluri, Parthasarathy, and Ruan 2011) reduces only the number of edges (focusing on inter-community edges) in order improve the running time of a clustering algorithm. In contrast to the above methods, our approach keeps the structure of the graph intact, without excluding any structural information from the clustering process.

k-core decomposition. Seidman (Seidman 1983) first applied the k-core decomposition to study the cohesion of social networks. Since then, it has been applied in several graph-related tasks, such as graph visualization (Alvarez-Hamelin et al. 2005; Zhang and Parthasarathy 2012) , as an edge ordering criterion for graph coarsening (Abou-Rjeili and Karypis 2006), and in the decomposition of massive graphs (Cheng et al. 2011).

Preliminaries

Given a graph G, we denote by $V(G)$ and $E(G)$ the sets of its vertices and edges respectively. Given a set $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G that is obtained if we remove from it all vertices that do not belong in S. We also use n and m for the number of the vertices and edges of G, i.e., $n = |V(G)|$ and $m = |E(G)|$. The neighborhood of a vertex $v \in V(G)$ in G is denoted by $N_G(v)$ and contains all vertices of G that are adjacent to v. The degree $\mathbf{deg}_G(v)$ of a vertex v in G is equal to $|N_G(v)|$. The *minimum degree* of G, denoted by $\delta(G)$, is the minimum degree of the vertices in G, i.e., $\delta(G) = \min\{\mathbf{deg}_G(v) \mid v \in V(G)\}$. Given a non-negative integer k, we define its *k-core*, denoted by $\mathbf{core}_k(G)$ as the maximum size subgraph of G with minimum degree k and we say that k is the *rank* of this core. The *degeneracy*, denoted by $\delta^*(G)$ of a graph G is the maximum k for which G contains a non-empty core. In other words, $\delta^*(G) = \max\{\delta(H) \mid H \subseteq G\}$.

Intuitively, the dense cores of a graph, i.e., those whose ranks are close to $\delta^*(G)$, may serve as a good seeds of starting any clustering algorithm as we expect them to respect the clustering structure of the original graph. Later at this paper, we make this statement more precise, providing the necessary theoretical justification.

For graph G and $\delta^*(G) = k$, we define its *core expansion sequence* as the sequence of vertex sets $\{V_k, V_{k-1}, ..., V_0\}$ that is recursively defined as follows: $V_k = V(\mathbf{core}_k(G))$ and $V_i = V(\mathbf{core}_i(G)) \setminus V_{i+1}, i = k-1, \ldots, 0$. We refer to the sets of a core expansion sequence as *layers*, with set V_i being its i-th layer (see Fig. 1).

Detecting the i-core of a graph G is easy (Batagelj and Zaversnik 2003): just remove vertices of degree less than i until this is not possible any more. It can be easily seen that the computation of k-core decomposition can be done in $O(k \cdot n)$ steps. As real world graphs have typically small degeneracy, this makes the computation of the *core expansion sequence* an easy computational task.

Proposed Method

CORECLUSTER capitalizes on the concept of degeneracy to improve the efficiency of graph clustering. The main idea behind our approach is that the k-core decomposition preserves the clustering structure of a graph and therefore the "best" k-core subgraph can be used as good starting point for a clustering method. Furthermore, the decomposition provides an hierarchical organization of the nodes in the graph, that can "guide" the clustering process.

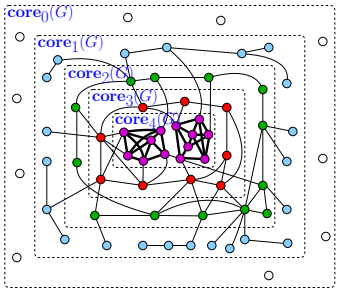


Figure 1: A graph G of degeneracy 4 and its cores. The different colors express the partition of the vertices of the graph to layers V_4, V_3, V_2, V_1, and V_0. Fat-edges indicate parts of a clustering of the graph.

The CORECLUSTER Framework

Suppose we have an algorithm that takes as input a graph G and outputs a partition of $V(G)$ into a number of sets that form a clustering of G. As in this section we are mainly focused on the general aspects of our method, we do not further specify the attributes of such an algorithm and we, abstractly, name it **Cluster**. We also assume that it runs in $O(n^3)$ steps as, in our experiments, the **Cluster** algorithm is the spectral algorithm of (Ng, Jordan, and Weiss 2001). Our aim is to define a procedure – called CORECLUSTER – that uses **Cluster** and accelerates the algorithm without any significant loss in its accuracy.

In simple terms, the CORECLUSTER framework applies the **Cluster** algorithm at the highest k-core of the graph and then it iterates from the highest to the lowest core trying to apply the following logic: assign with a simple criterion all the nodes that can be assigned to the existing clusters and apply the **Cluster** algorithm to the remaining nodes (in order to create new clusters).

Procedure CORECLUSTER(G).
Input: A graph G.
Output: A partition of $V(G)$ into clusters.
1. $k := \delta^*(G)$.
2. $q := 0$.
3. Let $V_k, \ldots, V_0$ be the core expansion sequence of G.
4. For $i = 0, \ldots, k$, let G_i be the i-core of G,
5. Let $S_k = V_k$.
6. Let $\mathcal{A}_k = \{C_1^k, \ldots, C_{\rho_k}^k\} = \mathbf{Cluster}(G[S_k])$.
7. **for** $i = k-1$ **to** 0 **do**
8. $\quad S_i =$*Select*$(G_i, \mathcal{A}_k \cup \ldots \cup \mathcal{A}_{i+1}, V_i)$,
9. $\quad$ let $\mathcal{A}_i = (C_1^i, \ldots, C_{\rho_i}^i) = \mathbf{Cluster}(G[S_i])$.
10. **Return** $\mathcal{A}^k \cup \cdots \cup \mathcal{A}^0$.

Initially, CORECLUSTER performs k-core decomposition to obtain the core expansion sequence of the graph. Then, algorithm **Cluster** is applied to the k-core subgraph, creating the first clusters. The procedure *Select* takes as input the, so far, created clusters, i.e., the sets in $\mathcal{F}_{i+1} = \mathcal{A}_k \cup \cdots \cup \mathcal{A}_{i+1}$ and the i-layer V_i and tries to assign each of the vertices of V_i in some cluster in $\mathcal{A}_k \cup \cdots \cup \mathcal{A}_{i+1}$. After this update, the procedure *Select* returns the unassigned vertices. The choice of the selection procedure considers the way the vertices of V_i are adjacent with the vertices of the clusters in $\mathcal{A}_k \cup \cdots \cup \mathcal{A}_{i+1}$. This selection can be done with several heuristic approaches, and next we describe such a procedure.

We stress that CORECLUSTER can be essentially seen as a "meta-algorithmic procedure" in the sense that it can be applied to any clustering algorithm. The discussion that follows in the next parts of the paper argues that this indeed can improve the clustering argument in time without any significant expected loss in its performance.

Selection procedure

In this section we describe the selection procedure (*Select*()) in Line 8 of CORECLUSTER. The procedure takes as input the so far created clustering $\mathcal{F}_{i+1} = \mathcal{A}_k \cup \cdots \cup \mathcal{A}_{i+1}$ and the vertex set V_i. We describe below how this procedure assigns some of the vertices of V_i to the clusters in $\mathcal{F}_{i+1}$ and outputs the remaining ones.

First of all we call a pair $(G, \mathcal{F}, V)$ a *candidate triple*, if G is a graph, and $\mathcal{F} \cup \{V\}$ is a partition of $V(G)$. Given a candidate triple $(G, \mathcal{F}, V)$, we define the following property on the vertices of V:

$$\mathbf{P}^{\alpha,\beta}(v) \quad = \quad \exists C \in \mathcal{F} : \frac{|N_G(v) \cap V(C)|}{|N_G(v)|} \geq \alpha \quad \text{and} \quad |N_G(v)| \geq \beta,$$

where $\alpha > 0.5$ and β is a positive integer. Notice that, as $\alpha > 0.5$, the truth of $\mathbf{P}^{\alpha,\beta}(v)$ can be certified by a unique set C in $\mathcal{F}$. We call such a set the *certificate* of v.

Procedure *Select*$(G, \mathcal{F}, V)$.
Input: A candidate triple $(G, \mathcal{F}, V)$
Output: A subset S of V and a partition $\mathcal{F}'$ of $V(\mathcal{F}) \cup (V \setminus S)$.
1. while $\mathbf{P}^{\alpha,\beta}(v)$ is true for some $v \in V$,
2. $\quad$ set $\mathcal{F} \leftarrow (\mathcal{F} \setminus \{C\}) \cup \{C \cup \{v\}\}$ where
3. $\quad\quad C$ is the certificate of v
4. $\quad$ and set $V \leftarrow V \setminus \{v\}$.
5. set $V^1 = N_G(V(\mathcal{F}))$ and $V^2 = V \setminus (V^1 \cup \mathcal{F})$.
6. if V^2 is either empty or an independent set of G,
7. $\quad$ then
8. $\quad\quad \mathcal{F} \leftarrow$ *assign*$(G, \mathcal{F}, V, V^1)$
9. $\quad\quad \mathcal{F} \leftarrow$*assign*$(G, \mathcal{F}, V, V^2)$
10. $\quad\quad$ return $\emptyset$
11. $\quad$ else return $V^1 \cup V^2$.

Before we present the *assign* routine we need some definitions. Given a candidate triple $(G, \mathcal{F}, V)$ and a vertex $v \in V$ we define $\mathbf{span}(v) = \max\{|N_G(v) \cap V(C)| \mid C \in \mathcal{F}\}$. We also define $\mathbf{argspan}(v)$ as a minimum size $C \in \mathcal{F}$ with the property that $|N_G(v) \cap V(C)| = \mathbf{span}(v)$.

Procedure *assign*$(G, \mathcal{F}, V, S)$.
Input: A candidate triple $(G, \mathcal{F}, V)$ and a subset S of V
Output: A partition $\mathcal{F}$
1. while $S \neq \emptyset$,
2. $\quad$ let $l = \max\{\mathbf{span}(v) \mid v \in S\}$
3. $\quad$ let $L = \{v \in S \mid \mathbf{span}(v) = l\}$
4. $\quad$ for every $v \in L$,
5. $\quad\quad$ set $C' = C \cup \{v\}$ where $C = \mathbf{argspan}(v)$
6. $\quad\quad$ set $S \leftarrow S \setminus \{v\}$
7. $\quad\quad$ set $\mathcal{F} \leftarrow (\mathcal{F} \setminus \{C\}) \cup \{C'\}$
8. return $\mathcal{F}$.

The selection procedure first (lines 1–4) tries to assign vertices of V to clusters of $\mathcal{F}$ using the criterion of the property $\mathbf{P}^{\alpha,\beta}$ that assigns a vertex to a cluster only if the vast majority of its neighbors belong in this cluster. The quantification of this "vast majority" criterion is done by the constants α and β. The vertices that cannot be assigned are partitioned into two groups: V^1 contains those that have neighbors in vertices that are already classified in the clusters of $\mathcal{F}$ and V^2 contains the rest. As the vertices in V^2 have no neighbors in the clusters, they have at least k neighbors out of them, it is most likely that they may not enter to any existing cluster in the future, unless, possibly, they are completely disjoint. If this is not the case, a further (milder) classification is attempted by the *assign* procedure that first classifies the vertices in V^1 in the existing clusters and then we do the same for the vertices in V^2. We stress that this last selection has been useful in our experiments in cores of low rank (where many independent vertices may appear). The procedure *assign* is a heuristic that classifies each vertex to the cluster that has the majority of its neighbors.

Expected execution time

The speed up of the algorithm is based on the fact that CORECLUSTER(G) now runs in $k+1$ disjoint subgraphs of G instead from G itself. As the i-th selection phase requires $O(|V(G_i)|^3)$ steps, we conclude that the running time of CORECLUSTER(G) is bounded by

$$\sum_{i=k,\ldots,0} O(|S_i|^3) \leq \sum_{i=k,\ldots,0} O(|V_i|^3) \leq O(k \cdot n_{\max}^3), \qquad (1)$$

where $n_{\max} = \max\{|V_k|,\ldots,|V_0|\}$. In the above bound, the first equality holds only in the extremal case where no selection occurs during the selection phases. Clearly, the general bound in Eq. (1) is the best possible when $|V_k|,\ldots,|V_0|$ tend to be equally distributed (which would accelerate the running time by a factor of k^2). According to the first inequality of Eq. (1) the running time of the algorithm is proportional to $(k+1)\cdot n_{\max}^3$, where $n_{\max} = \max\{|S_k|,\ldots,|S_0|\}$. Let $\rho_G = \max\{\frac{|V(G|)}{|S_i|} \mid i = 0,\ldots,k\}$ and $\mu_G = \max\{\frac{|V(G)}{|V_i|} \mid i = 0,\ldots,k\}$ and observe that $\rho_G \geq \mu_G$. Notice that the discrepancy between ρ and μ is a measure of the acceleration of the algorithm because of the selection phases.

Concluding, the acceleration of CORECLUSTER is upper bounded by

$$\sum_{i=k,\ldots,0} O(|S_i|^3) = O\left(\frac{k}{\rho^3}\cdot n^3\right).$$

This estimation is purely theoretical and its purpose is to expose the general complexity contribution of our algorithmic machinery. In practice, the acceleration can be *much better* and this also depends on the heuristics that we apply for the selection phase.

Quality of the CORECLUSTER framework

The intuition behind our framework is that the core expansion sequence $V_k, V_{k-1},\ldots,V_0$ gives a good sense of direction on how to perform clustering in an incremental way. After that, the procedure considers V_{k-1} as the remaining vertices of the $(k-1)$-core G_{k-1}, and tries to assign them one by one to the already existing clusters $C_1^k,\ldots,C_{\rho_k}^k$. The vertices for which such an assignment is not possible, form the set S_{k-1} and the **Cluster** is now applied on $G[S_{k-1}]$. As the algorithm continues, the existing clusters grow up and the vertices for which this is not possible, are grouped to new clusters. The fact that this procedure approximates satisfactorily the result of the application of **Cluster** to the whole graph is justified by the observation that the early i-cores (i.e., i-cores where i is close to k) are already dense, and therefore *sufficiently coherent*, to provide a good starting clustering that will expand well because of the selection criterion. In fact, the subgraphs obtained by the k-core decomposition, provide an $(1/2)$-approximation algorithm for the DENSEST-SUBGRAPH problem (Andersen and Chellapilla 2009).

Theoretical justification. We claim that the decomposition identifies subgraphs that progressively correspond to the most central regions and connected parts of the graph. Here we show that nodes with high clustering coefficient (Watts and Strogatz 1998) in G, are more likely to survive at the highest k-core subgraph by the pruning (k-core decomposition) procedure. Our claim is based on the following theorem:

Theorem 1 ((Gleich and Seshadhri 2012)) *Let G be a graph with heavy-tailed degree distribution, and let C_G be the (global) clustering coefficient of G. Then, there exists a k-core in G for $k \geq C_G \frac{d_{\max}^{\varepsilon}}{2}$, where $\varepsilon < 1$ is a constant such that most edges are incident to a node with degree at least $d_{\max}^{\varepsilon}$ (typically $\varepsilon = 2/3$), where $d_{\max}$ is the maximum degree of the nodes.*

The above theorem implies that graphs with heavy-tailed degree distribution and high global clustering coefficient C_G, have large degeneracy. Next we present our claim for the relationship between the local clustering coefficient C_v, $\forall v \in V(G)$ and the k-core subgraph justifying the selection of the k-core as good seed subgraph in the clustering procedure.

Claim 1 *Let G be a graph with heavy-tailed degree distribution. The contribution of each node $v \in V(G)$ to the k-core decomposition of the graph is proportional to the local clustering coefficient C_v.*

Proof Sketch. The global clustering coefficient C_G of the entire graph is given by the average of the local clustering coefficients C_v, $\forall v \in V(G)$, i.e., $C_G = \frac{1}{n}\sum_v C_v$, where $n = V(G)$. Then, from Theorem 1 we have that:

$$k \geq C_G \frac{d_{\max}^{\varepsilon}}{2} = \left(\frac{1}{n}\sum_v C_v\right)\frac{d_{\max}^{\varepsilon}}{2} = \underbrace{\left(\frac{1}{n}\frac{d_{\max}^{\varepsilon}}{2}\right)}_{\gamma}\sum_v C_v$$

$$\Rightarrow \quad k \geq \gamma \sum_v C_v,$$

where parameter γ captures global characteristics of the graph (that depend on the total number of nodes and the

maximum degree). Therefore, nodes with high clustering coefficient (in the original graph) are more likely to be found in the best k-core ($k = \delta^*(G)$) subgraph, since they tend to be more robust to the degeneracy process. Thus, the k-core subgraph can be used as good starting point (seed subgraph) for the clustering task. Additionally, we have experimentally validated the above claim (see (Giatsidis et al. 2014)).

Experimental Evaluation

Here we present the experimental results of our framework on both the amelioration of the execution time and the quality of the clustering results. Based on parameter space exploration, the values for the parameters of the *Select* procedure are chosen to be $a = 0.8$ and $\beta = 5$ (this choice appears to work optimally in our experiments).

Spectral algorithm As the baseline and the basis for the CORECLUSTER framework (algorithm **Cluster**), we use the Ng-Jordan-Weiss spectral clustering algorithm as it is described in (Ng, Jordan, and Weiss 2001). The basic idea of this algorithm is to keep the top k eigenvectors of the normalized adjacency matrix and perform k-means clustering on the rows of the matrix composed from these eigenvectors. In our variation we are using k-means++ (Arthur and Vassilvitskii 2007) for its advantage of performing better seeding during the initialization process and, since we desire to have an automatic choice for k, we define k by the "eigengap" as it is suggested in (Polito and Perona 2001).

Datasets description

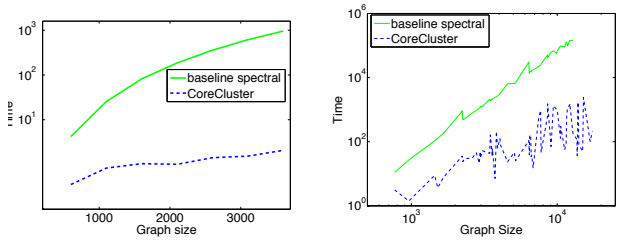
While real networks are the objective, actual datasets lack ground truth which leaves only evaluation metrics of the quality of clustering as an option and not direct comparison. On the other hand, artificial networks offer ground truth and a large variety of properties that can be parameterized to produce different "types" of networks.The evaluation of our framework is conducted on both real and artificial networks in order to have complete and decisive results.

Artificial Networks. We exploit the graph generator proposed in (Lancichinetti, Fortunato, and Radicchi 2008) to produce graphs with ground truth clustering structure. This graph generator provides a wide range of input parameters. We used the parameters in Table 1 and tuned them for various combinations in order to get a wide range of graphs with different features. Thus, the testing of our approach is credible as it is evaluated in essentially hundreds of graphs with different properties and quality of clustering structure. The parameters used are: N is the size of the graph, max_d is the maximum node degree, min_d is the minimum node degree (this is the most important parameter as it is the one differentiating the overall density of the graph) and μ is the mixing parameter representing the overlapping between clusters, i.e., each node shares a fraction $1 - \mu$ of its links with the other nodes of its community and a fraction μ with the other nodes of the network.

Real Networks. We also perform evaluations to a subset of the *Facebook* dataset (Traud, Mucha, and Porter 2011). This is a collection of friendship networks of Facebook from 2005, for 100 US Universities (i.e., 100 individual networks). The evaluations were not performed to the full extend of this dataset as hardware limitation did not allow us to evaluate, with spectral clustering, networks with more than $13K$ nodes (the CORECLUSTER framework could handle much larger networks). About half of the networks from this dataset were used for the final evaluation.

	D1	D2	D3
max_d (node max degree)	10%, 30%, 50%	10%, 30%, 50%	200 edges
min_d(node min degree)	∼5 (the absolutely minimum)	7	20
μ (mixing parameter)	1% – 43% (in 7 steps)	3% – 43% (in 6 steps)	3% – 43% (in 6 steps)
N (graph size in nodes)	600−3600	3500−5500	3500−5500

Table 1: Parameters' values for the artificial graphs.



(a) Graph D1 (b) Facebook

Figure 2: Execution time of baseline spectral graph clustering and of our framework for various graph sizes for (a) the graph dataset D1 and (b) Facebook respectively.

Time performance

Figures 2 (a) and 2 (b) depict the specific execution times for the artificial graph D1 and Facebook respectively (similar results can be observed for the artificial datasets D2 and D3; see Supplemental Material (Giatsidis et al. 2014)). It is evident that the gain in execution time using the proposed framework is very significant (i.e., at least three orders of magnitude for graph sizes above 3000 nodes) and increases polynomially with the graph size. Additionally, as we can see in Fig. 3, the execution time of our framework increases rather linearly with the graph size – thus showing good scaling features.

Clustering quality evaluation and setup

For each of the graphs at hand we run **i**. as baseline approach the Ng-Jordan-Weiss (Ng, Jordan, and Weiss 2001) spectral graph clustering algorithm and **ii**. our CORECLUSTER framework on the datasets presented earlier. Following, we describe the methods and metrics for evaluating the results on artificial and real networks.

Artificial networks: We measure the quality of the clustering results in terms of the widely used Normalized Mutual Infor-

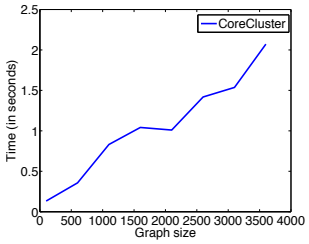
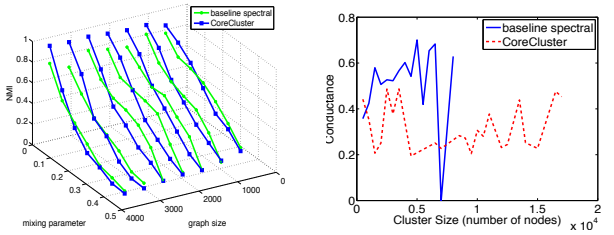


Figure 3: Execution time of the CORECLUSTER framework for various graph sizes.



(a) Graph D1: NMI (b) Facebook: Conductance

Figure 4: Clustering quality comparison (a) in terms of NMI (higher values are better) values for the artificial graph D1 and (b) in terms of conductance (lower values are better) for the Facebook dataset.

mation (Manning, Raghavan, and Schütze 2008) (NMI). In Fig. 4 (a), we give the comparative performance (in terms of NMI values) of the plain spectral algorithm as compared to the performance of our framework (CORECLUSTER) for different sizes and mixing parameter values of the D1 artificial graph (see Supplemental Material (Giatsidis et al. 2014) for graphs D2 and D3). Each point represents the average NMI value for all the graphs produced for each different combinations of parameter values, whose ranges appear in Table 1. For promoting statistical significance of the results for each of the aforementioned combinations, we run the generator ten times and we compute the corresponding graphs. We notice that our approach performs almost perfectly (with NMI>0.92) and generally outperforms the quality of the spectral algorithm applied directly on the graphs in most cases, especially as the graph size grows (this happens for mixing parameters values generally smaller than 0.2). For larger values of the mixing parameter, plain spectral clustering performs better (even though the absolute quality is low). Of course the counterargument here is that for larger values of the mixing parameter the overlap of the clusters is such that it basically prevents the definition of a clustering structure – and therefore perhaps it is meaningless to search for clusters in these cases.

We have to stress that in all cases, the execution time of our algorithm, especially for large graphs, is 3-4 orders of magnitude smaller that those of the plain clustering algorithms – achieving essentially the same or even better quality for reasonable values of the mixing parameters.

Facebook: The networks of this dataset lack ground truth, and for this reason we choose to evaluate the results with the evaluation criterion of *conductance*. Given a graph G and a cut $(S, \overline{S})$, conductance is defined as $\phi(S) = \frac{\sum_{i \in S, j \notin S} A_{ij}}{\min(a(S), a(\overline{S}))}$, where A_{ij} are the entries in the adjacency matrix $\mathbf{A}$ of G and $a(S) = \sum_{i \in S} \sum_{j \in G} A_{ij}$. Informally, conductance measures (for a cluster) the ratio of internal to external connectivity. It has been used widely to examine clustering quality (e.g., (Leskovec, Lang, and Mahoney 2010)) and has a simple and intuitive definition. In Fig. 4 (b), we can see the comparison of conductance values versus different sizes of detected communities by the two methods. Conductance has values in the range $(0, 1)$ with lower values indicating better clustering quality.

For better presentation (in Fig. 4 (b)) we have aggregated the detected cluster sizes (in terms of number of nodes) into bins of 500 (e.g., $0-500$, $501-1000$, etc.) and have provided the average conductance for each bin. This plot essentially provides the comparison of average clustering quality between the baseline and CORECLUSTER for different cluster sizes. Before commenting on the comparison, it is important to note that – for both methods – clusters with less than 10 nodes were excluded as they were trivia with regards to the clustering criteria for large scale graphs. Moreover, we have evaluated CORECLUSTER to a larger subset of *Facebook* including networks that we could not evaluate with the baseline spectral, due to limitations of hardware memory. Consequently, we have results of clusters up to $8K$ nodes (from networks of up to $13K$ of nodes) for the baseline and results of clusters up to $16K$ (from networks of up to $23K$ of nodes) for CORECLUSTER.

Moving on to the comparison, we can see that CORECLUSTER displays better clustering quality than the baseline, with the exception of the first bin. The difference is negligible and only slightly surpassed by the baseline's conductance value. For the last two bins of the baseline, we should note that there was only one cluster found for each, with the one having zero conductance consisting of the entire network (i.e., the whole graph was found as one cluster). In fairness, we could consider an "in between" value but it would be still worse than the corresponding conductance of CORECLUSTER. Overall, we see that CORECLUSTER displays a quite low conductance regardless of cluster size, indicating better clustering results in much faster time.

Conclusions

In this paper, we articulate an effort for optimizing the efficiency of graph clustering, capitalizing on the intuition that the extreme k-core of a graph preserves the clustering structure of the original graph, while it is much faster to execute clustering on this degenerate graph due to its much smaller size. Our main contribution is the CORECLUSTER framework, that initiates clustering on the highest rank core of the graph and then incrementally clusters the graph's nodes in the subsequent lower rank cores. Moreover we described analytically why this framework scales-up the clustering process and we showed experimentally on a multitude of graph data that the framework decreases the execution time while maintaining (or even improving) the clustering quality.

Acknowledgments

The authors would like to thank the anonymous reviewers for the constructive comments. Christos Giatsidis and Michalis Vazirgiannis were partially supported by the DIGITEO Chair grant LEVETONE in France. Fragkiskos D. Malliaros is a recipient of the Google Europe Fellowship in Graph Mining, and this research is supported in part by this Google Fellowship. Dimitrios M. Thilikos was co-financed by the E.U. (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: "Thales. Investing in knowledge society through the European Social Fund".

References

Abou-Rjeili, A., and Karypis, G. 2006. Multilevel algorithms for partitioning power-law graphs. In *IPDPS*, 124–124.

Alvarez-Hamelin, J. I.; Dall'asta, L.; Barrat, A.; and Vespignani, A. 2005. k-core decomposition: a tool for the visualization of large scale networks. *arXiv*.

Andersen, R., and Chellapilla, K. 2009. Finding dense subgraphs with size bounds. In *WAW*, 25–37.

Arthur, D., and Vassilvitskii, S. 2007. k-means++: the advantages of careful seeding. In *SODA*, 1027–1035.

Batagelj, V., and Zaversnik, M. 2003. An o(m) algorithm for cores decomposition of networks. *CoRR*.

Cheng, J.; Ke, Y.; Chu, S.; and Ozsu, M. 2011. Efficient core decomposition in massive networks. In *ICDE*, 51–62.

Clauset, A.; Newman, M. E. J.; and Moore, C. 2004. Finding community structure in very large networks. *Phys. Rev. E* 70(6):066111.

Drineas, P.; Frieze, A.; Kannan, R.; Vempala, S.; and Vinay, V. 2004. Clustering large graphs via the singular value decomposition. *Mach. Learn.* 56(1-3).

Fortunato, S. 2010. Community detection in graphs. *Physics Reports* 486(3-5).

Giatsidis, C.; Malliaros, F. D.; Thilikos, D. M.; and Vazirgiannis, M. 2014. Supplemental material: CoreCluster: A degeneracy based graph clustering framework. `http://www.lix.polytechnique.fr/dascim/wp-content/uploads/papers/corecluster14_supplemental.pdf`.

Gleich, D. F., and Seshadhri, C. 2012. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD*, 597–605.

Karypis, G., and Kumar, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20(1):359–392.

Kumar, S.; Mohri, M.; and Talwalkar, A. 2009. On sampling-based approximate spectral decomposition. In *ICML'09*, 553–560.

Lancichinetti, A.; Fortunato, S.; and Radicchi, F. 2008. Benchmark graphs for testing community detection algorithms. *Physical Review E* 78.

Leskovec, J., and Faloutsos, C. 2006. Sampling from large graphs. In *KDD*, 631–636.

Leskovec, J.; Lang, K. J.; and Mahoney, M. 2010. Empirical comparison of algorithms for network community detection. In *WWW*, 631–640.

Maiya, A. S., and Berger-Wolf, T. Y. 2010. Sampling community structure. In *WWW*, 701–710.

Manning, C. D.; Raghavan, P.; and Schütze, H. 2008. *Introduction to information retrieval*. Cambridge University Press.

Newman, M. E. J., and Girvan, M. 2004. Finding and evaluating community structure in networks. *Physical Review E* 69.

Newman, M. E. J. 2004. Fast algorithm for detecting community structure in networks. *Phys. Rev. E* 69(6):066133.

Ng, A. Y.; Jordan, M. I.; and Weiss, Y. 2001. On spectral clustering: Analysis and an algorithm. In *NIPS*, 849–856.

Polito, M., and Perona, P. 2001. Grouping and dimensionality reduction by locally linear embedding. In *NIPS*, 1255–1262.

Satuluri, V., and Parthasarathy, S. 2009. Scalable graph clustering using stochastic flows: applications to community discovery. In *KDD*, 737–746.

Satuluri, V.; Parthasarathy, S.; and Ruan, Y. 2011. Local graph sparsification for scalable clustering. In *SIGMOD*, 721–732.

Seidman, S. B. 1983. Network structure and minimum degree. *Social Networks* 5(3):269–287.

Shi, J., and Malik, J. 2000. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 22(8):888–905.

Traud, A. L.; Mucha, P. J.; and Porter, M. A. 2011. Social structure of facebook networks. *CoRR*.

Watts, D. J., and Strogatz, S. H. 1998. Collective dynamics of'small-world'networks. *Nature* 393(6684).

White, S., and Smyth, P. 2005. A spectral clustering approach to finding communities in graph. In *SDM*.

Zhang, Y., and Parthasarathy, S. 2012. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE*, 1049–1060.