An Ensemble Learning Strategy for Graph Clustering

Michael Ovelgönne and Andreas Geyer-Schulz

Abstract. This paper is on a graph clustering scheme inspired by ensemble learning. In short, the idea of ensemble learning is to learn several weak classifiers and use these weak classifiers to form a strong classifier. In this contribution, we use the generic procedure of ensemble learning and determine several weak graph clusterings (with respect to the objective function). From the partition given by the maximal overlap of these clusterings (the cluster cores), we continue the search for a strong clustering. We demonstrate the performance of this scheme by using it to maximize the modularity of a graph clustering. We show, that the quality of the initial weak clusterings is of minor importance for the quality of the final result of the scheme if we iterate the process of restarting from maximal overlaps. In addition to the empirical evaluation of the clustering scheme, we will link its search behavior to global analysis. With help of Morse theory and a discussion of the path space of the search heuristics we explain the superior search performance of this clustering scheme.

1. Introduction

Graph clustering, i.e. the identification of cohesive submodules or ‘natural’ groups in graphs, is an important technique in several domains. The identification of functional groups in metabolic networks [GA05] and the identification of social groups in friendship networks are two popular application areas of graph clustering.

Here we define graph clustering as the task of simultaneously detecting the number of submodules in a graph and detecting the submodules themselves. In contrast, we use the term graph partitioning for the problem of identifying a parametrized number of partitions where usually additional restrictions apply (usually, that all submodules are of roughly equal size). Two recent review articles on graph clustering by Schaeffer [Sch07] and Fortunato [For10] provide a good overview on graph clustering techniques as well as on related topics like evaluating and benchmarking clustering methods.

Graph clustering by optimizing an explicit objective function became popular with the introduction of the modularity measure [NG04]. Subsequently, a number of variations of modularity [MRC05, LZW+08] have been proposed to address shortcomings of modularity such as its resolution limit [FB07]. The identification

2000 Mathematics Subject Classification. Primary 05C85; Secondary 05C70, 68R10, 90C27.

©2000 (copyright holder)
of a graph clustering by finding a graph partition with maximal modularity is NP-hard [BDG+08]. Therefore, finding clusterings of a problem instance with more than a few hundred vertices has to be based on good heuristics. A large number of modularity optimization heuristics has been proposed in recent years, but most of them have a poor optimization quality.

The objective of this contribution is to present a new graph clustering scheme, called the Core Groups Graph Clustering (CGGC) scheme, which is able to find high quality clustering by using an ensemble learning approach. In [OGS10] we presented an algorithm called RG+ for maximizing the modularity of a graph partition via an intermediate step of first identifying core groups of vertices. The RG+ algorithm was able to outperform all previously published heuristics in terms of optimization quality. This paper deals with a generalization of this optimization approach.

The paper has been organized in the following way. First, we briefly discuss ensemble learning in Section 2. Then, we introduce the CGGC scheme in Section 3 and modularity maximization algorithms in Section 4. In Section 5, we evaluate the performance of the CGGC scheme using modularity maximization algorithms within the scheme. We discuss the scheme from the viewpoint of global analysis in Section 6. Finally, a short conclusion follows in Section 7.

2. Ensemble Learning

Ensemble based systems have been used in decision making for quite some time. Ensemble learning is a paradigm in machine learning, where several intermediate classifiers (called weak or base classifiers) are generated and combined to finally get a single classifier. The algorithms used to compute the weak classifiers are called weak learners. An important notion is, that even if a weak learner has only a slightly better accuracy than random choice, by combining several classifiers created by this weak learner, a strong classifier can be created [Sch90]. For a good introduction to this topic, see the review article by Polikar [Pol06].

Two examples of ensemble learning strategies are bagging and boosting. A bagging algorithm for supervised classification trains several classifiers from bootstraps of the training data. The combined classifier is computed by simple majority voting of the ensemble of base classifiers, i.e. a data item gets the label the majority of base classifiers assigns to that data item. A simple boosting algorithm (following [Pol06]) works with classifiers trained from three subsets of the training data. The first dataset is a random subset of the training data of arbitrary size. The second dataset is created so that the classifier trained with the first dataset classifies half of the data items correctly and the other half wrong. The third dataset consists of the data items the classifiers trained by the first and the second dataset disagree on. The strong classifier is the majority vote of the three classifiers.

Another ensemble learning strategy called Stacked Generalization has been proposed by Wolpert [Wol92]. This strategy is based on the assumption that some data points are more likely to be misclassified than others, because they are near to the boundary that separates different classes of data points. First, an ensemble of classifiers is trained. Then, using the output of the classifiers a second level of classifiers is trained with the outputs of the ensemble of classifiers. In other words, the second level of classifiers learns for which input a first level classifier is correct or how to combine the “guesses” of the first level classifiers.
An ensemble learning strategy for clustering has been used by Fred and Jain \cite{FJ05}, first. They called this approach evidence accumulation. They worked on clustering data points in an Euclidean space. Initially, the data points are clustered several times based on their distance and by means of an algorithm like k-means. The ensemble of generated clusterings is used to create a new distance matrix called the co-association matrix. The new similarity between two data points is the fraction of partitions that assign both data points to the same cluster. Then, the data points are clustered on basis of the co-association matrix.

3. Core Groups Graph Clustering Scheme

Let us restrict our considerations to the problem of whether a pair of vertices should belong to the same cluster or to different clusters. Making this decision is complicated. Many algorithms get misled during their search so that sometimes bad decisions are made. But what if we have one or more algorithms that find several clusterings with fair quality but still a lot of non-optimal decisions on whether a pair of vertices belongs to the same cluster? If all clusterings agree on whether a pair of vertices belongs to the same cluster, we can be pretty sure that this decision is correct. However, if the clusterings disagree, we should have a second look at this pair.

Based on these considerations, we propose the CGGC scheme. We use the agreements of several clusterings with fair quality to decide whether a pair of vertices should belong to the same cluster. The groups of vertices which are assigned to the same cluster in every clustering (i.e. the maximal overlaps of the clusterings) are denoted as core groups. To abstract from any specific quality measure, we use the term \textit{good} partition for a partition that has a good quality according to an arbitrary quality measure. The CGGC scheme consists of the following steps:

1. Create a set $S$ of $k$ \textit{good} partitions of $G$ with base algorithm $A_{\text{viral}}$
2. Identify the partition $\hat{P}$ of the maximal overlaps in $S$
3. Create a graph $\hat{G}$ induced by the partition $\hat{P}$
4. Use base algorithm $A_{\text{final}}$ to search for a \textit{good} partition of $\hat{G}$
5. Project partition of $\hat{G}$ back to $G$

Initially, a set $S$ of $k$ partitions of $G$ is created. That means, one non-deterministic clustering algorithm is started $k$ times to create the graph partitions, $k$ deterministic but different algorithms are used or a combination of both is used. In terms of ensemble learning, the used algorithms are the base algorithms or weak learners and the computed clusterings are the weak classifiers.

Next, we combine the information of the weak classifiers: We calculate the maximal overlap of the clusterings in $S$. Let $c_P(v)$ denote the cluster that vertex $v$ belongs to in partition $P$. We create from a set $S$ of partitions $\{P_1, \ldots, P_k\}$ of $V$ a new partition $\hat{P}$ of $V$ so that

$$\forall v, w \in V : \left( \bigwedge_{i=1}^{k} c_{P_i}(v) = c_{P_i}(w) \right) \Leftrightarrow c_{\hat{P}}(v) = c_{\hat{P}}(w)$$

Extracting the maximum overlap of an ensemble of partitions creates an intermediate solution which is used as the starting point for the base algorithm $A_{\text{final}}$ to calculate the final clustering. The base algorithm used in this phase could be
an algorithm used in step 1 or any other algorithm appropriate to optimize the objective function. For example, algorithms that are not able to cluster the original network in reasonable time could be used to cluster the smaller graph $\hat{G} = (\hat{V}, \hat{E})$ induced by $\hat{P}$. To create the induced graph, all vertices in a cluster in $\hat{P}$ are merged to one vertex in $G$. Accordingly, $\hat{G}$ has as many vertices as there are clusters in $\hat{P}$. An edge $(v, w) \in E$ has the weight of the combined weights of all edges in $G$ that connect vertices in the clusters represented by $v$ and $w$. Then, the clustering of $\hat{G}$ would have to be projected back to $G$ to get a clustering of the original graph.

Agglomerative hierarchical optimization schemes often show the best scalability for clustering algorithms as they usually make local decisions. A partial explanation is that the number of partitions of $n$ nodes in $k$ classes grows as a Stirling number of the second kind $S(n, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (k-r)^n$ and that this implies that growth of the search space is smaller in the bottom-up direction than in the top-down direction [Boc74, p. 110]. For the example shown in Figure 1, we have 10 partitions (5 objects in 4 clusters) for the first bottom-up decision versus 15 partitions (5 objects in 2 clusters) for the first top-down decision.

While using only local information increases the scalability, it is a source of globally poor decisions, too. Extracting the overlap of an ensemble of clusterings provides a more global view. Figure 1 shows the complete merge lattice of an example graph of 5 vertices. An agglomerative hierarchical algorithm always starts with the partition into singletons (shown at the bottom) and merges in some way the clusters until only one cluster containing all vertices remains (shown at the top). Every merge decision means going one level up in the lattice. Restarting the search at the maximal overlap of several partitions in an ensemble means to go back to a point in the lattice from which all of the partitions in this ensemble can be reached. If we restart the search for a good partition from this point, we will most probably be able to reach other good partitions than those in the ensemble, too. In fact, reaching other good or even better partitions than those in the ensemble will be easier than starting from singletons as poor cluster assignments in the ensemble have been leveled out.

### 3.1. The Iterated Approach

Wolpert [Wol92] discussed the problem that some data points are harder to assign to the correct cluster than others. Data points at the natural border of two clusters are harder to assign than those inside. For the specific case of modularity maximization with agglomerative hierarchical algorithms, we discussed the influence of prior merge decision on all later merges in [OGS12]. Often, the order of the merge operations influences which side of the border a vertex is assigned to. Node 3 in Figure 1 is an example for this effect.

With the help of the maximal overlaps of the CGGC scheme we try to separate the cores of the cluster from the boundaries. The harder decisions on which clusters contain the vertices at the boundaries are made, when the knowledge of the cores provides additional information. This idea of separating cores and boundaries can be iterated in the following way (subsequently denoted as the CGGCi scheme):

1. Set $P^{best}$ to the partition into singletons and set $\hat{G}$ to $G$
2. Create a set $S$ of $k$ (fairly) good partitions of $\hat{G}$ with base algorithm $A_{initial}$
3. Identify the partition $\hat{P}$ of the maximal overlaps in $S$
4. If $\hat{P}$ is a better partition than $P^{best}$, set $P^{best} = \hat{P}$, create the graph $\hat{G}$ induced by $\hat{P}$ and go back to step 2
Figure 1. Graph $G$ with 5 vertices and its merge lattice. The edges indicate the possible merge paths of hierarchical clustering algorithms (not all edges drawn). The stroked edges indicate paths leading through the saddle point (1 2) 3 4 5 to the two local maxima. The dotted edges and italic partitions can be neglected as they correspond to merging clusters that are not adjacent. Merging non-adjacent clusters will always decrease the modularity.

(5) Use base algorithm $A_{\text{final}}$ to search for a good partition of $\hat{G}$

(6) Project partition of $\hat{G}$ back to $G$

In every new clustering $P^{\text{best}}$ some more vertices or groups of vertices have been merged or rearranged. So, every new clustering is likely to provide more accurate information on the structure of the graph for the succeeding iterations.

4. Modularity and its Optimization

Modularity is a popular objective function for graph clustering that measures the non-randomness of a graph partition. Let $G = (V, E)$ be an undirected, unweighted graph, $n := |V|$ the number of vertices, $m := |E|$ the number of edges and $P = \{C_1, \ldots, C_k\}$ a partition of $V$, i.e. $\bigcup_{i=1}^{k} C_i = V$ and $\forall i \neq j \in \{1, \ldots, k\} C_i \cap C_j = \emptyset$. The modularity $Q$ of the partition $P$ of graph $G$ is defined as

$$Q(G, P) = \frac{1}{2m} \sum_{v_x, v_y} \left( w_{xy} - \frac{s_x s_y}{2m} \right) \delta(c_P(v_x), c_P(v_y))$$

where $w_{xy}$ is an element in the adjacency matrix of $G$, $s_x$ is the degree of vertex $v_x$, $c_P(v_x)$ is the cluster of $v_x$ in partition $P$ and the Kronecker symbol $\delta(c(v_x), c(v_y)) = 1$ when $v_x$ and $v_y$ belong to the same cluster and $\delta(c(v_x), c(v_y)) = 0$ otherwise.

Research on modularity maximization algorithms has been very popular in the last years and a lot of heuristic algorithms have been proposed. In the following, we discuss a randomized greedy and a label propagation algorithm in detail, as
we will use them exemplarily to evaluate the CGGC scheme. We will give a brief
summary of other algorithms which could be used as base algorithms for the CGGC
scheme as well. For an extensive overview on modularity maximization algorithms,
see [For10].

4.1. Randomized Greedy (RG). Newman [New04] proposed the first algorithm
to be used to identify clusterings by maximizing modularity. The hierarchical agglomerative algorithm starts with a partition into singletons and merges in each step one pair of clusters that causes the maximal increase in modularity. The result is the cut of the dendrogram with the maximal modularity. This algorithm is slow, as it considers to merge every pair of adjacent clusters in every step. The complete search over all adjacent pairs also leads to an unbalanced merge process. Some clusters grow faster than others and the size difference is a bias for later merge decisions. Large clusters are merged with many small clusters in their neighborhood, whether this is good from a global perspective or not [OGS12].

The randomized greedy algorithm [OGS10] is a fast agglomerative hierarchical algorithm that has a very similar structure to Newman’s algorithm but does not suffer from an unbalanced merge process. This algorithm selects in every step a small sample of $k$ vertices and determines the best merge involving one of the vertices in the sample (see Algorithm 1). Because of the sampling, the algorithm can be implemented quite efficiently and has a complexity of $O(m \ln n)$ (see [OGS10]).

<table>
<thead>
<tr>
<th>Algorithm 1: Randomized Greedy (RG) algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> undirected, connected graph $G = (V, E)$, sample size $k$</td>
</tr>
<tr>
<td><strong>Output:</strong> clustering</td>
</tr>
<tr>
<td>▼ <strong>Initialize</strong></td>
</tr>
<tr>
<td>for all the $v \in V$ do</td>
</tr>
<tr>
<td>for all the neighbors $n$ of $v$ do</td>
</tr>
<tr>
<td>$e[v, n] \leftarrow 1/(2*\text{edgecount})$;</td>
</tr>
<tr>
<td>$a[v] \leftarrow \text{rowsum}(e[v])$</td>
</tr>
<tr>
<td>▼ <strong>Build Dendrogram (Randomized Greedy)</strong></td>
</tr>
<tr>
<td>for $i = 1$ to rank($e$)-1 do</td>
</tr>
<tr>
<td>maxDeltaQ $\leftarrow -\infty$;</td>
</tr>
<tr>
<td>for $j = 1$ to $k$ do //search among $k$ communities for best join</td>
</tr>
<tr>
<td>$c1 \leftarrow$ random community;</td>
</tr>
<tr>
<td>for all communities $c2$ connected to $c1$ do</td>
</tr>
<tr>
<td>$deltaQ \leftarrow 2(e[c1, c2] - (a[c1] * a[c2]));$</td>
</tr>
<tr>
<td>if $deltaQ &gt; maxDeltaQ$ then</td>
</tr>
<tr>
<td>$maxDeltaQ \leftarrow deltaQ;$</td>
</tr>
<tr>
<td>$nextjoin \leftarrow (c1, c2);$</td>
</tr>
<tr>
<td>join(nextjoin);</td>
</tr>
<tr>
<td>joinList $\leftarrow$ joinList + nextjoin;</td>
</tr>
<tr>
<td>clusters $\leftarrow \text{extractClustersFromJoins(joinList)}$</td>
</tr>
</tbody>
</table>

In [OGS10] we also introduced the RG+ (improved randomized greedy) algorithm, which we generalized to the CGGC scheme in this contribution. The RG+
Algorithm 2: Label Propagation (LP) algorithm

Input: undirected, connected graph \( G = (V, E) \) set of labels \( LP \)
Output: clustering

\( \triangledown \) Initialize

\( \forall \) all the \( v \in V \) do
\( \quad \) label\([v]\) \leftarrow \text{getUniqueID}(LP); \\
\( \triangledown \) Propagate Labels

\( \text{majorityLabelCount} \leftarrow 0; \)
\( \text{while} \) \( \text{majorityLabelCount} \neq |V| \) \( \text{do} \)
\( \quad \text{majorityLabelCount} \leftarrow 0; \)
\( \forall \) all the \( v \in V \) at random do
\( \quad \text{label}[v] \leftarrow \arg\max_{l \in LP} \sum_{n \in \text{neighbors}(v)} \delta(l, \text{label}[n]); \)
\( \quad \text{if} \sum_{n \in \text{neighbors}(v)} \delta(\text{label}[v], \text{label}[n]) \geq |V|/2 \text{ then} \)
\( \quad \text{majorityLabelCount} \leftarrow \text{majorityLabelCount} + 1; \)

algorithm uses the RG algorithm as its base clustering algorithm to create the weak classifiers and for the final clustering starting from the maximal overlap of these partitions. To obtain a standardized naming of all other CGGC scheme algorithms in this article we will denote this algorithms as \( CGGC_{RG} \) in the following.

4.2. Label Propagation (LP). Raghavan et al. [RAK07] proposed a label propagation algorithm for graph clustering. This algorithm initializes every vertex of a graph with a unique label. Then, in iterative sweeps over the set of vertices the vertex labels are updated. A vertex gets the label that the maximum number of its neighbors have. Ties are broken arbitrarily. The procedure is stopped when every vertex has the label that at least half of its neighbors have. The pseudocode of the LP algorithm is shown in Algorithm 2.

This procedure does not explicitly or implicitly maximize modularity. It is especially interesting, because it has a near linear time complexity. Every sweep has a complexity of \( O(m) \) and Raghavan et al. report that 95% of the vertices have a label the majority of its neighbors have in only about 5 iterations.

As we will show in Section 5, the CGGC scheme is able to find good final clusterings from weak results of intermediate runs of base algorithms. It does not matter if the algorithm is stopped prior to its originally defined stopping criterion.

4.3. Other Modularity Maximization Algorithms. A very fast agglomerative hierarchical algorithm has been developed by Blondel et al. [BGLL08]. The algorithm starts with singleton clusters. Every step of the algorithm consists of two phases. At first, all vertices are sequentially and iteratively moved between their current and a neighboring cluster, if this increases the modularity. In the case that several moves have a positive influence on the modularity, the one with the highest modularity increase is chosen. To speed up this process, a threshold is introduced to determine, when to stop the first phase based on the relative increase in modularity. In the second phase of each step, the result of the first phase is used to create a new graph, where all vertices that have been assigned to the same
cluster in the first phase are represented by one vertex. The edge weights between the original vertices are summed up and give the new edge weights between the new vertices. Then, the algorithm returns to the first phase and moves the new vertices between clusters.

Noack and Rotta [NR09] experimentally investigated a framework of hierarchical agglomerative modularity optimization algorithms. While most algorithms only use the modularity increase as the priority criterion, they analyzed several other priority criteria that weight modularity increase in some way. Furthermore, they considered merging more than one pair of vertices in every step and locally refining the intermediate partitions regularly during the merging process (multi-level refinement). With the best configuration of their framework Noack and Rotta achieve significantly better results than Blondel et al. [BGLL08] at the price of a much higher runtime.

Another well performing algorithm is the MOME algorithm by Zhu et al. [ZWM+08]. In a first phase, the coarsening phase, the algorithm recursively creates a set of graphs. Starting with the input graph, each vertex of the graph will be merged with the neighbor that yields the maximal increase in modularity. If the modularity delta is negative for all neighbors, the vertex will be left as it is. The resulting graph will be recursively processed until the graph can not be contracted any more. Subsequently, in the uncoarsening phase, the set of successively collapsed graphs will be expanded while the clustering gets refined by moving vertices between neighboring clusters.

Many other algorithms have been proposed. For practical usage and to be used within the CGGC scheme most of them are of no interest due to their inferior performance in terms of modularity maximization or runtime efficiency. Among these algorithms are several spectral algorithms ([WS05], [New06], [RZ07], [RZ08]) and algorithms based on generic meta heuristics like iterated tabu search [MLR06], simulated annealing [MAnD05], or mean field annealing [LH07]. Formulations of modularity maximization as an integer linear program (e.g. [AK08], [BDG+07]) allow finding an optimal solution without enumerating all possible partitions. However, processing networks with as few as 100 vertices is already a major problem for current computers.

4.3.1. Refinement. The results of most modularity maximization algorithms can be improved by a local vertex mover strategy. Noack and Rotta [NR09] surveyed the performance of several strategies inspired by the famous Kernighan-Lin algorithm [KL70]. We employ the fast greedy vertex movement strategy to the results of all evaluated algorithms, because all other strategies scale much worse without providing significant improvements in quality. The fast greedy vertex mover strategy sweeps iteratively over the set of vertices as long as moving a vertex to one of its neighboring clusters improves modularity.

5. Evaluation

The clustering scheme is evaluated by means of real-world and artificial networks from the testbed of the 10th DIMACS implementation challenge on graph partitioning and graph clustering. Memory complexity is a bigger issue than time complexity for our algorithms and we had to omit the two largest datasets from the category Clustering Instances because of insufficient main memory. We also
omitted the small networks with less than 400 vertices where many algorithms are able to find the optimal partitions [OGS10].

Before we conducted the evaluation, we first determined the best choice for the number of partitions in the ensembles. The results of our tests (see Figure 3) show that the ensemble size should be roughly $\ln n$ for all algorithms but $CGGC_{LP}$. When using LP as the base algorithm, the quality improves with increasing ensemble size for the iterated scheme but heavily decreases for the non-iterated scheme (see Figure 2). This seems to be a result of the weak learning quality of LP. A larger ensemble size results in more and smaller core groups in the maximal overlap partition. LP is not able to find a good clustering from finer decompositions when not iteratively applied as in the CGGCi scheme.

The results in Table 2 show the average optimization quality and therefore the quality we can expect when using the algorithm in a practical context. In Table 1 we show the boundary of the scheme, i.e. the best optimization quality we were able to achieve using the scheme given much time.

While the iterated CGGCi scheme does not provide much improvement compared to the non-iterated scheme when used with the RG algorithm ($CGGC_{iRG}$ vs. $CGGC_{RG}$), its improvement for the LP algorithm is significant ($CGGC_{iLP}$ vs. $CGGC_{LP}$). There is still a difference between the $CGGC_{iRG}$ and $CGGC_{iLP}$. But for most networks, $CGGC_{iLP}$ achieves better results than the standalone $RG$ algorithm which showed to be a quite competitive algorithm [OGS10] among non-CGGC scheme algorithms.

A notable result is that the LP algorithm performs extremely bad on the preferentialAttachment network (pref.Attach.). This network is the result of a random network generation process where iteratively edges are added to the network and the probability that an edge is attached to one vertex depends on the current degree of the vertex. The average modularity for the standalone LP on the preferentialAttachment network is extremely low as the algorithm identified only in 1 of 100 test
Figure 3. Average modularity of 30 test runs of the CGGC/CGGCi-scheme algorithms subject to the ensemble size $k$ for the two datasets PGPgiantcompo and caidaRouterLevel. The dotted vertical line shows the value of $\ln n$ (where $n$ is the number of vertices)
Table 1. Best modularity of a clustering computed for networks from the DIMACS testbed categories Clustering Instances and Coauthors. All partitions have been identified with help of the CGGC scheme and the denoted base algorithm.

<table>
<thead>
<tr>
<th>Network</th>
<th>Max Modularity</th>
<th>Alg.</th>
<th>Network</th>
<th>Max Modularity</th>
<th>Alg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-celegans_metabolic</td>
<td>0.4526664838</td>
<td>RG</td>
<td>eu-2005</td>
<td>0.9415630621</td>
<td>RG</td>
</tr>
<tr>
<td>Email</td>
<td>0.5828085954</td>
<td>RG</td>
<td>in-2004</td>
<td>0.9806076266</td>
<td>RG</td>
</tr>
<tr>
<td>PGPGiantcompo</td>
<td>0.8865501696</td>
<td>RG</td>
<td>road_central</td>
<td>0.9976280448</td>
<td>RG</td>
</tr>
<tr>
<td>as-22july06</td>
<td>0.6783599573</td>
<td>RG</td>
<td>road_usa</td>
<td>0.9982186002</td>
<td>RG</td>
</tr>
<tr>
<td>astro-ph</td>
<td>0.7444262906</td>
<td>RG</td>
<td>caidaRouterLevel</td>
<td>0.872095371</td>
<td>RG</td>
</tr>
<tr>
<td>cond-mat</td>
<td>0.8530972563</td>
<td>RG</td>
<td>pref.Attach.</td>
<td>0.3048516381</td>
<td>RG</td>
</tr>
<tr>
<td>cond-mat-2003</td>
<td>0.7786823470</td>
<td>RG</td>
<td>smallworld</td>
<td>0.7930994465</td>
<td>LP</td>
</tr>
<tr>
<td>cond-mat-2005</td>
<td>0.7464446826</td>
<td>RG</td>
<td>G_n,pin,pout</td>
<td>0.5002934104</td>
<td>LP</td>
</tr>
<tr>
<td>hep-th</td>
<td>0.8565536355</td>
<td>RG</td>
<td>citationCiteseer</td>
<td>0.8241257861</td>
<td>RG</td>
</tr>
<tr>
<td>netscience</td>
<td>0.9598999889</td>
<td>RG</td>
<td>coAuthorsCiteseer</td>
<td>0.9053559700</td>
<td>RG</td>
</tr>
<tr>
<td>polblogs</td>
<td>0.4270879141</td>
<td>RG</td>
<td>coAuthorsDBLP</td>
<td>0.8415177919</td>
<td>RG</td>
</tr>
<tr>
<td>power</td>
<td>0.9404810777</td>
<td>RG</td>
<td>coPapersCiteseer</td>
<td>0.9226201646</td>
<td>RG</td>
</tr>
<tr>
<td>cnr-2000</td>
<td>0.9131075546</td>
<td>RG</td>
<td>coPapersDBLP</td>
<td>0.8667751453</td>
<td>RG</td>
</tr>
</tbody>
</table>

6. A Global Analysis View on the CGGC Scheme

We already gave an intuitive explanation of the way the CGGC scheme works in Section 3. Now, we want to provide a link to global analysis and Morse theory. The merge lattice shown in Figure 1 shows the space of all paths an agglomerative hierarchical algorithm can follow. The level number $k$ corresponds to the
<table>
<thead>
<tr>
<th>Network</th>
<th>RG</th>
<th>CGGC&lt;sub&gt;RG&lt;/sub&gt;</th>
<th>CGGC&lt;sub&gt;iRG&lt;/sub&gt;</th>
<th>LP</th>
<th>CGGC&lt;sub&gt;LP&lt;/sub&gt;</th>
<th>CGGC&lt;sub&gt;iLP&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>celegens_metabolic</td>
<td>0.43674</td>
<td>0.45021</td>
<td>0.45019</td>
<td>0.37572</td>
<td>0.43856</td>
<td>0.44343</td>
</tr>
<tr>
<td>Email</td>
<td>0.57116</td>
<td>0.57986</td>
<td>0.58012</td>
<td>0.41595</td>
<td>0.55750</td>
<td>0.55668</td>
</tr>
<tr>
<td>PGP_giantcompo</td>
<td>0.86436</td>
<td>0.88616</td>
<td>0.88617</td>
<td>0.76512</td>
<td>0.85431</td>
<td>0.88565</td>
</tr>
<tr>
<td>as-22_july06</td>
<td>0.66676</td>
<td>0.67742</td>
<td>0.67747</td>
<td>0.54930</td>
<td>0.61205</td>
<td>0.67316</td>
</tr>
<tr>
<td>astro-ph</td>
<td>0.69699</td>
<td>0.74275</td>
<td>0.74277</td>
<td>0.67511</td>
<td>0.70272</td>
<td>0.74143</td>
</tr>
<tr>
<td>cond-mat</td>
<td>0.82975</td>
<td>0.85240</td>
<td>0.85242</td>
<td>0.75661</td>
<td>0.79379</td>
<td>0.85116</td>
</tr>
<tr>
<td>cond-mat-2003</td>
<td>0.75715</td>
<td>0.77754</td>
<td>0.77755</td>
<td>0.67852</td>
<td>0.70551</td>
<td>0.77524</td>
</tr>
<tr>
<td>cond-mat-2005</td>
<td>0.72203</td>
<td>0.74543</td>
<td>0.74550</td>
<td>0.64184</td>
<td>0.67453</td>
<td>0.74199</td>
</tr>
<tr>
<td>hep-th</td>
<td>0.83403</td>
<td>0.85577</td>
<td>0.85573</td>
<td>0.76102</td>
<td>0.80614</td>
<td>0.85463</td>
</tr>
<tr>
<td>netscience</td>
<td>0.94037</td>
<td>0.95975</td>
<td>0.95973</td>
<td>0.92577</td>
<td>0.95375</td>
<td>0.95933</td>
</tr>
<tr>
<td>polblogs</td>
<td>0.42585</td>
<td>0.42678</td>
<td>0.42680</td>
<td>0.42610</td>
<td>0.42635</td>
<td>0.42633</td>
</tr>
<tr>
<td>power</td>
<td>0.92818</td>
<td>0.93962</td>
<td>0.93966</td>
<td>0.72124</td>
<td>0.79601</td>
<td>0.93794</td>
</tr>
<tr>
<td>cnr-2000</td>
<td>0.91266</td>
<td>0.91302</td>
<td>0.91309</td>
<td>0.86887</td>
<td>0.90603</td>
<td>0.91284</td>
</tr>
<tr>
<td>eu-2005</td>
<td>0.93903</td>
<td>0.94114</td>
<td>0.94115</td>
<td>0.85291</td>
<td>0.90610</td>
<td>0.93982</td>
</tr>
<tr>
<td>in-2004</td>
<td>0.97763</td>
<td>0.97832</td>
<td>0.98057</td>
<td>0.92236</td>
<td>0.97086</td>
<td>0.97791</td>
</tr>
<tr>
<td>road_central*</td>
<td>0.99716</td>
<td>0.99761</td>
<td>0.99767</td>
<td>0.70863</td>
<td>0.94351</td>
<td>0.99749</td>
</tr>
<tr>
<td>road_usa*</td>
<td>0.99786</td>
<td>0.99821</td>
<td>0.99825</td>
<td>0.72234</td>
<td>0.94682</td>
<td>0.99812</td>
</tr>
<tr>
<td>caidaRouterLevel</td>
<td>0.86136</td>
<td>0.86762</td>
<td>0.87172</td>
<td>0.76353</td>
<td>0.81487</td>
<td>0.87081</td>
</tr>
<tr>
<td>prefAttach</td>
<td>0.27984</td>
<td>0.29389</td>
<td>0.30099</td>
<td>0.00202</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>smallworld</td>
<td>0.78334</td>
<td>0.79289</td>
<td>0.79300</td>
<td>0.66687</td>
<td>0.69181</td>
<td>0.79307</td>
</tr>
<tr>
<td>G_n_pin_pout</td>
<td>0.47779</td>
<td>0.49991</td>
<td>0.50006</td>
<td>0.30609</td>
<td>0.34639</td>
<td>0.50023</td>
</tr>
<tr>
<td>citationCiteeseer</td>
<td>0.80863</td>
<td>0.82333</td>
<td>0.82336</td>
<td>0.66184</td>
<td>0.72526</td>
<td>0.82064</td>
</tr>
<tr>
<td>coAuthorsCiteeseer</td>
<td>0.89506</td>
<td>0.90507</td>
<td>0.90509</td>
<td>0.79549</td>
<td>0.83862</td>
<td>0.90360</td>
</tr>
<tr>
<td>coAuthorsDBLP</td>
<td>0.82081</td>
<td>0.83728</td>
<td>0.84055</td>
<td>0.71502</td>
<td>0.75108</td>
<td>0.83661</td>
</tr>
<tr>
<td>coPapersCiteeseer</td>
<td>0.91626</td>
<td>0.92168</td>
<td>0.92221</td>
<td>0.85653</td>
<td>0.89921</td>
<td>0.92162</td>
</tr>
<tr>
<td>coPapersDBLP</td>
<td>0.85383</td>
<td>0.86471</td>
<td>0.86655</td>
<td>0.77918</td>
<td>0.82674</td>
<td>0.86540</td>
</tr>
</tbody>
</table>

Table 2. Average modularity of the results of 100 test runs (10 test runs for very large networks marked with *) on networks from the DIMACS testbed categories Clustering Instances and Coauthors. CGGC<sub>X</sub> and CGGC<sub>iX</sub> denote the usage of an base algorithm X within the CGGC and the iterated CGGC scheme, respectively.
Figure 4. The clustering process of the iterated CGGCi scheme on the cond-mat-2005 dataset for the base algorithms RG (top) and LP (bottom). All points but the last one are core groups, i.e., maximal overlaps of the $k$ partitions in the ensemble. The last points are the results for the final clustering run and after applying the refinement procedure.
number of clusters of the partition(s) at level \( k \) in the merge lattice: from the singleton partition (the inf of the lattice) at level \( k = n \) to the partition with a single cluster (the sup of the lattice) at level \( k = 1 \). For a partition with \( k \) clusters we have \( \binom{k-1}{2} \) merge choices.

In each iteration of the CGGCi scheme, the search starts at some partition \( P \) at level \( k \) and goes up in the lattice to identify several new local maxima (the partitions in the ensemble \( S \)). For example, the algorithm starts twice at the partition 1 2 3 4 5 at level 5 in Figure 1 and reaches the two local maxima (1 2 3)(4 5) and (1 2)(3 4 5) at level 2. Then the algorithm goes down in the lattice to the maximal overlap partition \( \hat{P} \) at a level \( k_\hat{P} \leq k_P \). In the example, this is the partition (1 2) 3 (4 5) at level 3. In the worst case, when the ensemble of partitions \( S \) created starting at \( P \) does not agree on any vertex, the maximal overlap is again \( P \) and the core groups search stops. Otherwise, when the ensemble agrees on how to merge at least one vertex, a new core groups partition is identified at a level \( k_\hat{P} < k_P \).

If a local optimum \( P \) has been reached by a hill-climbing method, all partitions that have been visited on the way through the merge lattice to \( P \) have a lower objective function value than the local optima. As can be seen from the merge lattice given in Figure 1, there are usually many paths to get from the bottom partition on level \( n \) to any other partition.

A path in the merge lattice can be identified by an ordered set of partitions. Let \( F_P \) denote the set of all paths that connect the singleton partition to the partition \( P \), let \( \Omega \) denote all partitions of a set of vertices \( V \), and \( S \) be a set of partitions. Then, \( \mathcal{P}(S) = \{ P \in \Omega \mid \forall P_i \in S \exists D \in F_P : P \in D \} \) is the set of all partitions that are included in at least one path to each partition in \( S \). In other words, \( \mathcal{P}(S) \) is the set of all branch points from which all partitions in \( S \) can be reached. \( \mathcal{P}(S) \) always contains at least the singleton partition which all paths share as the starting point. The maximal overlap \( \hat{P} \) of the ensemble of partitions in \( S \) is the partition in \( \mathcal{P}(S) \) with the minimal number of clusters. That means, \( \hat{P} \) is the latest point from where a hierarchical agglomerative algorithm can reach all partitions in the ensemble. We see that the core groups partition of the maximal overlap is a special partition as it is a branching point in the merge path of the ensemble \( S \).

For a moment, we put the merge path discussion aside and discuss Morse theory which originates from the work of Morse on the topology of manifolds \([\text{Mor34}]\). Although the theory originally has been developed for continuous function spaces, and we are dealing with discrete optimization, Morse theory provides a suitable means to understand the topology of high-dimensional non-linear functions. In the following, we assume that the discrete points (the partitions of a graph) are

\begin{table}[h]
\centering
\begin{tabular}{lcccc}
\hline
 & \#Vertices & \#Edges & \text{RG} & \text{CGGC}_{RG} & \text{CGGCi}_{RG} \\
\hline
polblogs & 1490 & 16715 & 0.02 & 0.11 & 0.16 \\
power & 4941 & 6594 & 0.02 & 0.12 & 0.49 \\
cond-mat-2005 & 31163 & 120029 & 0.53 & 6.18 & 24.0 \\
caidaRouterLevel & 192244 & 609066 & 2.23 & 30.1 & 83.0 \\
eu-2005 & 862664 & 16138468 & 32.1 & 466 & 505 \\
\hline
\end{tabular}
\caption{Average runtime results (in sec.) for selected networks.}
\end{table}
embedded in a continuous space in such a way that the critical points (maxima, minima, saddle-points) of the discrete modularity maximization problem are also critical points in the continuous version of the problem.

Following the discussion in [JJT00], let us assume we have a continuous function \( f : \mathbb{R}^2 \to \mathbb{R} \) as in Figure 5a. The inverse of \( f \), \( f^{-1} \), then gives the level line of all points having the same value of \( f \). We denote \( f^{-1}(y) \) as the level line at level \( y \). While the level line is continuously deformed while going along \( y \), whenever a level passes a stationary or Karush-Kuhn-Tucker point (local minimum, local maximum or saddle point), its topology changes. Figure 5b shows the level lines at critical levels of the function in Figure 5a. At level 5 the level line connected at lower levels separates into the lines A and B, i.e. at level 5 the two lines are glued together by the saddle point and above level 5 they are unconnected.

This analysis of the level lines is important for optimization, as a greedy algorithm starting at level 6 can potentially reach any point, while for a hill-climbing algorithm starting from a point at level 5 the starting point determines the reachable parts of the search space. The separated level lines (A and B) create basins of attraction for the respective local optima (\( A^* \) and \( B^* \)). At level 5, the only point with a gradient path to both local maxima is the saddle point. Let us assume we have a deterministic, hill-climbing algorithm. Then, the result of the algorithm is determined by the starting point. Each local optimum has a basin of attraction, i.e. a non-empty set of points from which the algorithm goes to the respective local optimum.

In Figure 6 we show the basins of attraction for two functions in \( \mathbb{R}^1 \) and \( \mathbb{R}^2 \). Consider the bounded non-linear function in \( \mathbb{R}^1 \) shown in Figure 6a. Maxima and minima alternate when going from one end of the interval to the other. The minima are critical points for gradient algorithms, because they separate the basins of attraction (labeled A-D). Starting at a minimum, a gradient algorithm can reach either the maximum to its left or the one to its right. In addition, the intermediate value theorem tells us that in between two maxima, there must be at least one minimum.

In Figure 6b a similar situation in \( \mathbb{R}^2 \) is shown. In contrast to the situation in \( \mathbb{R}^1 \), for the higher dimensional space the borders of the basins of attraction are glued together at saddle points. Again, these saddle points are important starting
Figure 6. Basins of attraction for a non-linear function \( f \) in \( \mathbb{R}^1 \) (a) and for a non-linear function \( f \) in \( \mathbb{R}^2 \) (b). In (b) the broken arrows indicate the trajectories to the saddle points, the full arrows the trajectories to the local maxima \( A^*, B^*, C^*, \) and \( D^* \) (rough sketch). They separate the (open) basins of attraction \( A, B, C, \) and \( D \). We call this graph a Morse Graph. The dotted arrow in \( D \) goes from the local minimum to a saddle point. (Subfigure (b) is a variation of [JJT00, Fig. 1.4.2])

### Table 4. Properties of strict critical points

<table>
<thead>
<tr>
<th>Local maximum</th>
<th>( Df(x_c) = 0 )</th>
<th>Merge lattice of partitions</th>
<th>( \forall x \in N(x_c) : f(x_c) &gt; f(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D^2f(x_c) ) negative definite</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local minimum</td>
<td>( Df(x_c) = 0 )</td>
<td>( \forall x \in N(x_c) : f(x_c) &lt; f(x) )</td>
<td></td>
</tr>
<tr>
<td>Saddle point</td>
<td>( Df(x_c) = 0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

points for randomized gradient algorithms, because when starting from these points different local optima can be reached depending on the direction the randomized gradient algorithm follows. In contrast, gradient algorithms starting at points in the interior of basins of attraction lead to one local maximum - even if they are randomized.

Table 4 compares the properties of strict critical points for at least 2-times continuously differentiable spaces with the properties of critical points in the merge
Figure 7. Rugged mountain saddle and its Morse graph

lattice of agglomerative hierarchical modularity clustering algorithms. Note, that saddle points are characterized as split points of algorithm paths to critical points. In Figure 1 such a path split occurs at the partition \((1\ 2)\ 3\ (4\ 5)\) with two paths leading to the two local maxima \((1\ 2\ 3)(4\ 5)\) and \((1\ 2)(3\ 4\ 5)\).

Thus, the core groups partitions correspond to saddle points as in the path space of a graph the core groups are branch points where the join-paths to local maxima separate. As the core groups partitions correspond to saddle points, they are good start points for randomized greedy algorithms. For other classifiers, e.g. the label propagation algorithm, core groups partitions work well as long as the classifiers reach points in different basins of attraction which is a weaker condition than the requirement of reaching a local maximum. Obviously, in order to be a good restart point in the CGGC scheme, other local optima need to be reachable from a core group than those used to create the core groups, too. The rugged mountain saddle shown in Figure 7 is a familiar example for such a branch point in \(\mathbb{R}^3\). By iteratively identifying core groups of increasing modularity, we identify saddle points that lead to higher and higher local maxima.

In summary, through the theoretical considerations of this section (and supported by the evaluation in Section 5) our explanation for the high optimization quality of the CGGC scheme is:

- The operation of forming core groups partitions from sets of locally (almost) maximal partitions identifies (some) critical points on the merge lattice of partitions.
- Core group partitions are good points for restarting randomized greedy algorithms, because a core groups partition is a branch point (saddle point) in the search space where different basins of attraction meet.

7. Conclusion

In this paper we have shown that learning several weak classifiers has a number of advantages for graph clustering. The maximal overlap of several weak classifiers is a good restart point for further search. Depending on the viewpoint, this approach can be regarded as a way to make first the 'easy' decisions on which pairs of vertices belong together and make 'harder' decisions not before the unambiguous ones have been made. When looking at the search space, maximal overlaps seem to be capable of identifying those critical points from which especially randomized gradient algorithms can find good local maxima.

As it turned out, when using the CGGC scheme, the choice of base algorithm has no major impact on the clustering quality. This is an important notion. Using the core groups scheme, the base algorithm(s) can be selected because of other
considerations. For example, for most so far developed algorithms for modularity maximization an efficient implementation for distributed computer environments (e.g. a Hadoop cluster) would be very hard. However, the label propagation algorithm seems to be very suitable for this kind of environment. Propagating labels requires only to pass the label information between the nodes of a computer cluster. Thus, this algorithm can be used in the CGGCi scheme and in a distributed computing environment to find high quality clusterings of billion-edge networks.

For greedy base algorithms, we showed that the CGGC scheme explores the Morse graph of critical points. That explains why the scheme is able to achieve high optimization performance even in huge graphs with modest effort.

However, an open question is the theoretical justification of the size of the ensemble which is used for the determination of the maximal overlap partition.

References


MICHAEL OVELGÖNNE, UMIACS, UNIVERSITY OF MARYLAND, COLLEGE PARK, MD, USA
E-mail address: mov@umiacs.umd.edu

ANDREAS GEYER-SCHULZ, INSTITUTE OF INFORMATION SYSTEMS AND MANAGEMENT, KARLSRUHE INSTITUTE OF TECHNOLOGY, KARLSRUHE, GERMANY
E-mail address: andreas.geyer-schulz@kit.edu