An Ensemble Learning Strategy for Graph Clustering

Michael Ovelgönne and Andreas Geyer-Schulz

Institute of Information Systems and Management
Karlsruhe Institute of Technology
Karlsruhe, Germany
{michael.ovelgoenne, andreas.geyer-schulz}@kit.edu

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 determine 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.

Keywords: graph clustering, ensemble learning

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 [8] 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 [25] and Fortunato [5] 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 [17]. Subsequently, a number of variations of modularity have been proposed [14, 11] to address shortcomings of modularity such as its resolution limit [6]. The identification of a graph clustering

by finding a graph partition with maximal modularity is NP-hard [4]. 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 Cluster (CGGC) scheme, which is able to find high quality clustering by using an ensemble learning approach. In [19] 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. Finally, a short conclusion follows in Section 6.

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 [26]. For a good introduction to this topic, see the review article by Polikar [21].

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 [21]) 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 [28]. 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 [7], 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 clustering. 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 this 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 *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 *good* partitions of G with base algorithm $A_{initial}$
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_{final} to search for a *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 c 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

$$\forall i \in [1,k], v,w \in V : c_{P_i}(v) = c_{P_i}(w) \Rightarrow c_{\hat{P}}(v) = c_{\hat{P}}(w) \quad (1)$$

$$\exists i \in [1,k], v,w \in V : c_{P_i}(v) \neq c_{P_i}(w) \Rightarrow c_{\hat{P}}(v) \neq c_{\hat{P}}(w) \quad (2)$$

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_{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 $\hat{G}$. Accordingly, $\hat{G}$ has as many vertices as there are clusters in $\hat{P}$. An edge $(v,w) \in \hat{E}$ 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. 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 a complete graph of 4 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 [28] 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 in [20] the influence of prior merge decision on all later merges. The order of the merge operation influences which side of the border a vertex is assigned to.

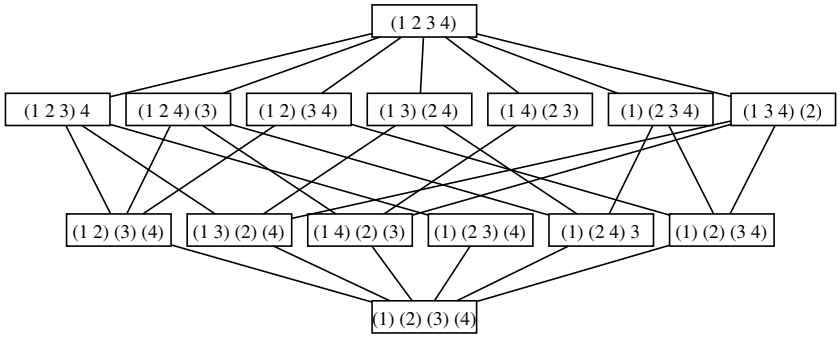

Fig. 1. Merge lattice of a complete graph with four vertices. The edges show the possible merge paths of hierarchical clustering algorithms.

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 decision on 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
5. Use base algorithm A_{final} to search for a *good* partition of $\hat{G}$
6. Project partition of $\hat{G}$ back to G

In every new clustering P^{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. $\cup_{i=1}^{k} C_i = V$ 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

$$Q(G, P) = \frac{1}{2m} \sum_{v_x, v_y} \left(w_{xy} - \frac{s_x s_y}{2m} \right) \delta(c_P(v_x), c_P(v_y)) \tag{3}$$

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 [5].

4.1 Randomized Greedy (RG)

Newman [15] 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. But this 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 [20].

The randomized greedy algorithm [19] 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 z 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 roughly $O(m \ln n)$.

In [19] we also introduced the RG+ (improved randomized greedy) algorithm, which we generalized to the CGGC scheme in this contribution. The RG+ 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. [22] 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. 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 implicitely maximize modularity. It is especially interesting, because it has a near linear time complexity. Every sweep

Algorithm 1 Randomized Greedy (RG) algorithm

Input: undirected, connected graph g, constant k
Output: clustering

```
▼ Initialize
    forall v ∈ V do
        forall neighbors n of v do
            e[v, n] ← 1/(2 ∗ edgecount)
        a[v] ← rowsum(e[v])

▼ Build Dendrogram (Randomized Greedy)
    for i = 1 to rank(e)-1 do
        maxDeltaQ ← −∞
        for j = 1 to k do //search among k communities for best join
            c1 ← random community
            for all communities c2 connected to c1 do
                deltaQ ← 2(e[c1, c2] − (a[c1] ∗ a[c2]))
                if deltaQ > maxDeltaQ then
                    maxDeltaQ ← deltaQ
                    nextjoin ← (c1, c2)
        join(nextjoin)
        joinList ← joinList + nextjoin
    clusters ← extractClustersFromJoins(joinList)
```

Algorithm 2 Label Propagation (LP) algorithm

Input: undirected, connected graph g, constant k
Output: clustering

```
▼ Initialize
    forall v ∈ V do
        label[v] ← getUniqueID()

▼ Propagate Labels
    majorityLabelCount ← 0
    while majorityLabelCount ≠ |V| do
        majorityLabelCount ← 0
        forall v ∈ V at random do
            label[v] ← argmax_l ∑_{n∈neighbors(v)} δ(l, label[n])
            if ∑_{n∈N(v)} δ(l, label[n]) ≥ |V|/2 then
                majorityLabelCount ← majorityLabelCount + 1
```

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. [2]. 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.

Another well performing algorithm is the MOME algorithm by Zhu et al. [29]. 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 [27] [16] [23] [24] and algorithms based on generic meta heuristics like iterated tabu search [13], simulated annealing [12] or mean field annealing [10]. Formulations of modularity maximization as an integer linear programs (e.g. [1], [3]) 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.

Refinement The results of most modularity maximization algorithms can be improved by a local vertex mover strategy. Noack and Rotta [18] surveyed the performance of several strategies inspired by the famous Kernighan-Lin algorithm [9]. We employ to the results of all evaluated algorithms the fast greedy

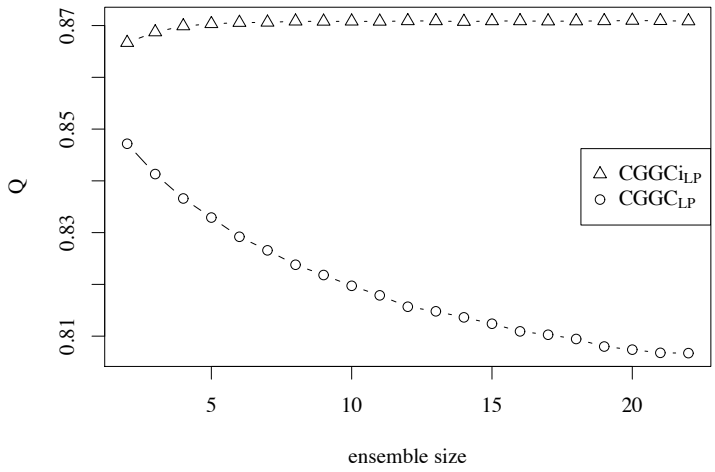

Fig. 2. Average modularity of 30 test runs of the CGGC- and CGGCi-scheme using LP as the base algorithm subject to the ensemble size k for the dataset *caidaRouterLevel*.

vertex movement strategy, 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 [19].

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.

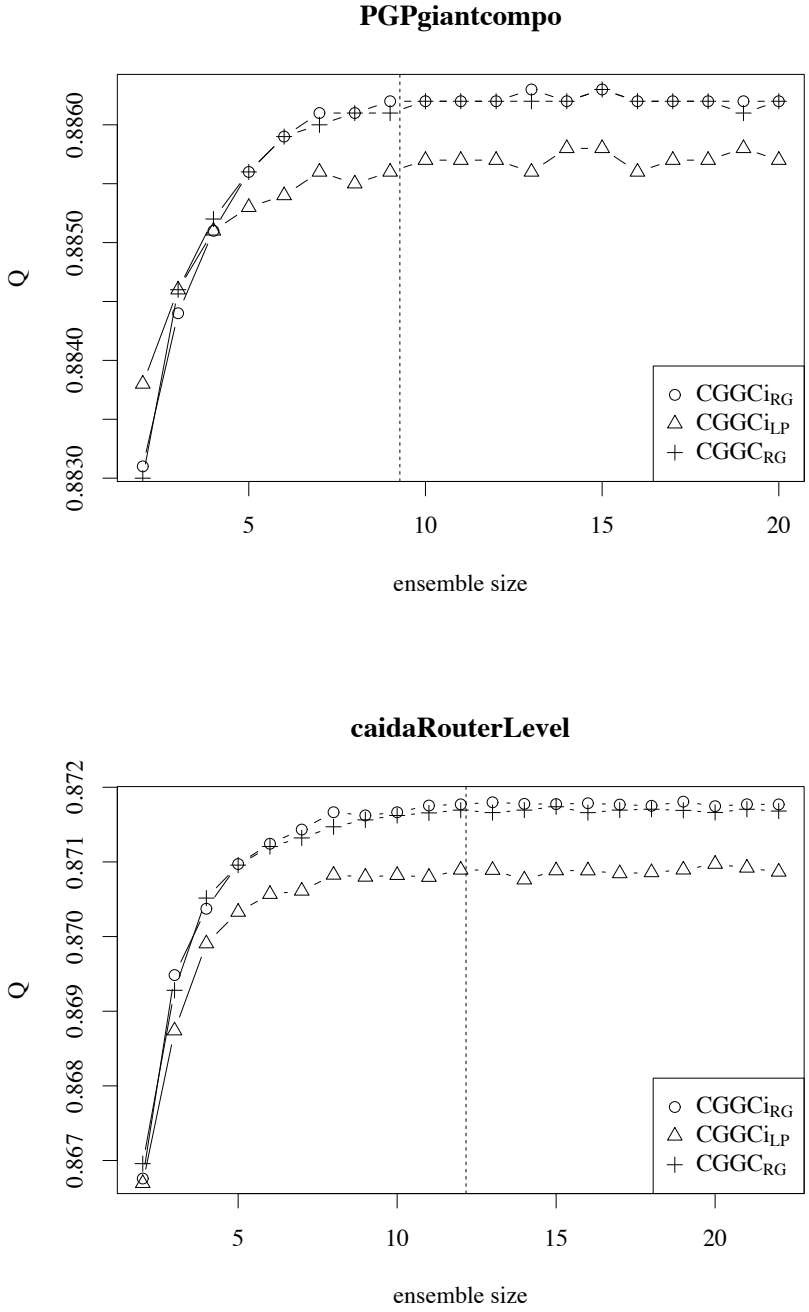

Fig. 3. Average modularity of 30 test runs of the CGGC/CGGCi-scheme algorithms subject to the ensemble size k for the two datasets *PGPgiantcompo* and *caidaRouterLevel*. The dotted vertical line shows the value of ln n (where n is the number of vertices)

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 CGGCi scheme and the denoted base algorithm.

Network	Max Modularity	Alg.	Network	Max Modularity	Alg.
celegans_metabolic	0.4526664838	RG	eu-2005	0.9415630621	RG
Email	0.5828085954	RG	in-2004	0.9806076266	RG
PGPgiantcompo	0.8865501696	RG	road_central	0.9976280448	RG
as-22july06	0.6783599573	RG	road_usa	0.9982186002	RG
astro-ph	0.7444262906	RG	caidaRouterLevel	0.8720295371	RG
cond-mat	0.8530972563	RG	pref.Attach.	0.3048516381	RG
cond-mat-2003	0.7786823470	RG	smallworld	0.7930994465	LP
cond-mat-2005	0.7464446826	RG	G_n_pin_pout	0.5002934104	LP
hep-th	0.8565536355	RG	citationCiteseer	0.8241257861	RG
netscience	0.9598999889	RG	coAuthorsCiteseer	0.9053559700	RG
polblogs	0.4270879141	RG	coAuthorsDBLP	0.8415177919	RG
power	0.9404810777	RG	coPapersCiteseer	0.9226201646	RG
cnr-2000	0.9131075546	RG	coPapersDBLP	0.8667751453	RG

While the iterated CGGCi scheme does not provide much improvement compared to the non-iterated scheme when used with the RG algorithm ($CGGCi_{RG}$ vs. $CGGC_{RG}$), its improvement for the LP algorithm is significant ($CGGCi_{LP}$ vs. $CGGC_{LP}$). There is still a difference between the $CGGCi_{RG}$ and $CGGCi_{LP}$. But for most networks, $CGGCi_{LP}$ achieves better results than the standalone *RG* algorithm which showed to be a quite competitive algorithm [19] 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 runs a community structure. In all other cases the identified clusterings were partitions into singletons. Therefore, using LP within the CGGC scheme failed as well. However, we can argue that trying to find a significant community structure in a random network should fail.

The clustering process of the iterated CGGC scheme is shown by example in Figure 4. The LP algorithm is a much weaker learner than the RG algorithm and initially finds clusterings with very low modularity. But after a few iterations the modularity of the core groups of both base algorithms are about the same. But although the quality of the final core groups for both base algorithms is similar, the core groups are different. The final core groups identified from the ensemble generated with the LP algorithm are a weaker restart point than those identified with RG. If we use RG as the base algorithm for the final clustering (A_{final}) to start from the LP core groups, the identified partitions have about the same

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*. $CGGC_X$ and $CGGCi_X$ denote the usage of an base algorithm X within the CGGC and the iterated CGGC scheme, respectively.

	RG	$CGGC_{RG}$	$CGGCi_{RG}$	*LP*	$CGGC_{LP}$	$CGGCi_{LP}$
celegans_metabolic	0.43674	0.45021	0.45019	0.37572	0.43856	0.44343
Email	0.57116	0.57986	0.58012	0.41595	0.55750	0.55668
PGPgiantcompo	0.86436	0.88616	0.88617	0.76512	0.85431	0.88565
as-22july06	0.66676	0.67742	0.67747	0.54930	0.61205	0.67316
astro-ph	0.69699	0.74275	0.74277	0.67511	0.70272	0.74143
cond-mat	0.82975	0.85240	0.85242	0.75661	0.79379	0.85116
cond-mat-2003	0.75715	0.77754	0.77755	0.67852	0.70551	0.77524
cond-mat-2005	0.72203	0.74543	0.74550	0.64184	0.67453	0.74199
hep-th	0.83403	0.85577	0.85575	0.76102	0.80614	0.85463
netscience	0.94037	0.95975	0.95974	0.92477	0.95375	0.95933
polblogs	0.42585	0.42678	0.42680	0.42610	0.42635	0.42633
power	0.92818	0.93962	0.93966	0.72124	0.79601	0.93794
cnr-2000	0.91266	0.91302	0.91309	0.86887	0.90603	0.91284
eu-2005	0.93903	0.94114	0.94115	0.85291	0.90610	0.93982
in-2004	0.97763	0.97832	0.98057	0.92236	0.97086	0.97791
road_central*	0.99716	0.99761	0.99767	0.70863	0.94351	0.99749
road_usa*	0.99786	0.99821	0.99825	0.72234	0.94682	0.99812
caidaRouterLevel	0.86136	0.86762	0.87172	0.76353	0.81487	0.87081
pref.Attach.	0.27984	0.29389	0.30099	0.00202	0.00000	0.00000
smallworld	0.78334	0.79289	0.79300	0.66687	0.69181	0.79307
G_n_pin_pout	0.47779	0.49991	0.50006	0.30609	0.34639	0.50023
citationCiteseer	0.80863	0.82333	0.82336	0.66184	0.72256	0.82064
coAuthorsCiteseer	0.89506	0.90507	0.90509	0.79549	0.83862	0.90360
coAuthorsDBLP	0.82081	0.83728	0.84055	0.71502	0.75108	0.83661
coPapersCiteseer	0.91626	0.92168	0.92221	0.85653	0.89921	0.92162
coPapersDBLP	0.85383	0.86471	0.86655	0.77918	0.82674	0.86540

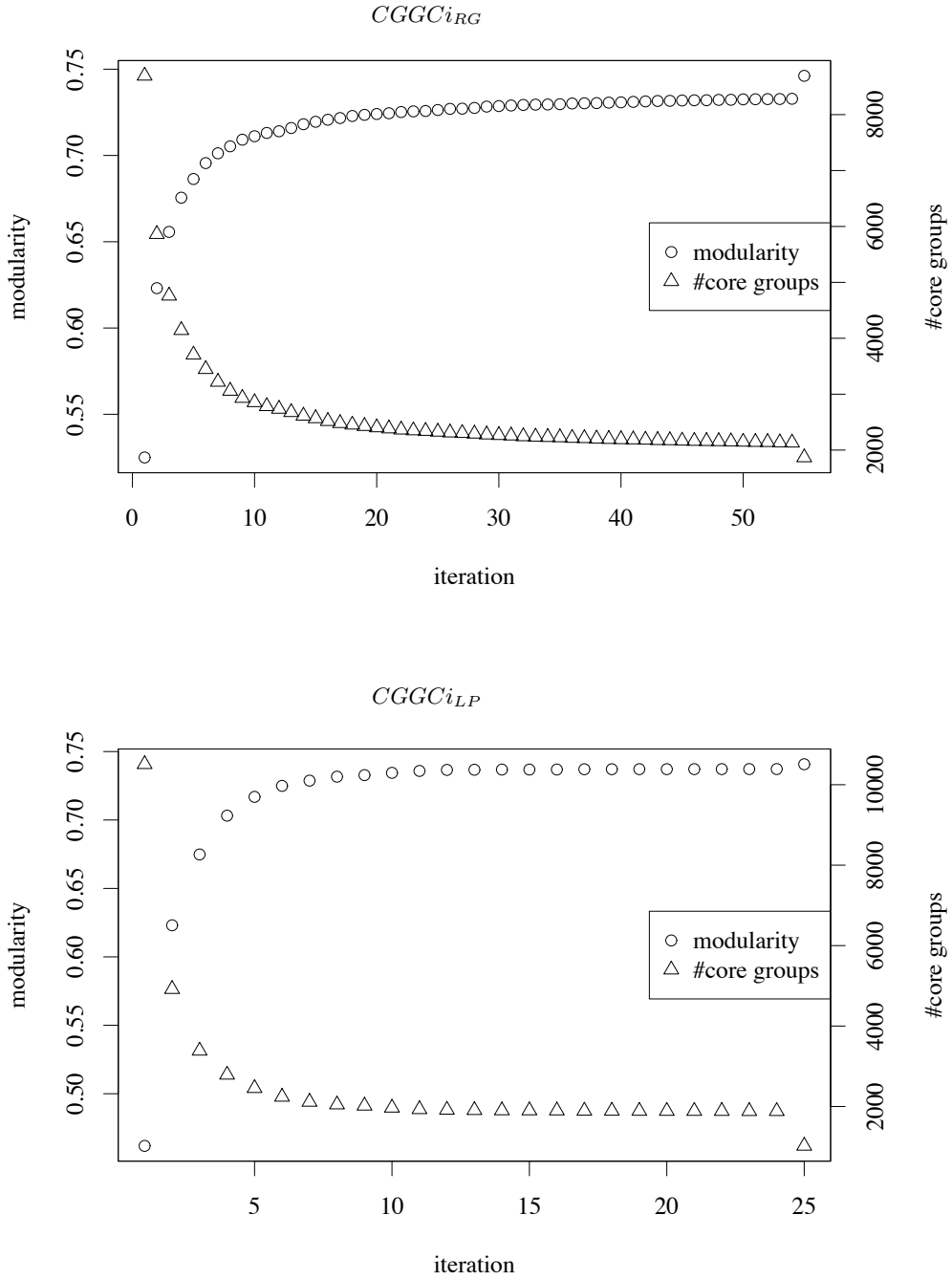

Fig. 4. The clustering process of the iterated CGGCi scheme on the *cond-mat-2005* dataset for the base algorithms RG (top) and LP (bottom). All points but the last one are core groups, i.e. maximal overlaps of the k partitions in the ensemble. The last points are the results for the final clustering run and after applying the refinement procedure.

modularity than those identified with LP. Because of page limitations we omit detailed results.

6 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 gradient algorithms can find good local maxima.

As it turned out, when using the CGGCi 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 mega scale networks.

References

1. Agarwal, G., Kempe, D.: Modularity-maximizing graph communities via mathematical programming. European Journal of Physics B 66, 409–418 (2008)
2. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10), P10008 (2008)
3. Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On finding graph clusterings with maximum modularity. In: Graph-Theoretic Concepts in Computer Science, pp. 121–132. Springer (2007)
4. Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Transactions on Knowledge and Data Engineering 20(2), 172–188 (2008)
5. Fortunato, S.: Community detection in graphs. Physics Reports 486(3-5), 75–174 (2010)
6. Fortunato, S., Barthélemy, M.: Resolution limit in community detection. Proceedings of the National Academy of Sciences of the United States of America 104(1), 36–41 (2007)
7. Fred, A.L.N., Jain, A.K.: Combining multiple clusterings using evidence accumulation. IEEE Trans. Pattern Anal. Mach. Intell. 27, 835–850 (2005)
8. Guimera, R., Amaral, L.: Functional cartography of complex metabolic networks. Nature 433, 895–900 (2005)

9. Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal 49(1), 291–307 (1970)
10. Lehmann, S., Hansen, L.: Deterministic modularity optimization. The European Physical Journal B - Condensed Matter and Complex Systems 60(1), 83–88 (2007)
11. Li, Z., Zhang, S., Wang, R.S., Zhang, X.S., Chen, L.: Quantitative function for community detection. Phys. Rev. E 77(3) (2008)
12. Medus, A., Acuña, G., Dorso, C.: Detection of community structures in networks via global optimization. Physica A: Statistical Mechanics and its Applications 358(2-4), 593–604 (2005)
13. Misevicius, A., Lenkevicius, A., Rubliauskas, D.: Iterated tabu search: an improvement to standard tabu search. Information Technology and Control 35, 187–197 (2006)
14. Muff, S., Rao, F., Caflisch, A.: Local modularity measure for network clusterizations. Physical Review E 72(5), 056107 (2005)
15. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Physical Review E 69(6), 066133 (2004)
16. Newman, M.E.J.: Modularity and community structure in networks. Proceedings of the National Academy of Sciences of the United States of America 103(23), 8577–8582 (2006)
17. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E 69(2), 026113 (2004)
18. Noack, A., Rotta, R.: Multi-level algorithms for modularity clustering. In: Vahrenhold, J. (ed.) Experimental Algorithms, Lecture Notes in Computer Science, vol. 5526, pp. 257–268. Springer Berlin / Heidelberg (2009)
19. Ovelgönne, M., Geyer-Schulz, A.: Cluster cores and modularity maximization. In: ICDMW '10. IEEE International Conference on Data Mining Workshops. pp. 1204–1213 (2010)
20. Ovelgönne, M., Geyer-Schulz, A.: A comparison of agglomerative hierarchical algorithms for modularity clustering. In: Advances in Data Analysis and Classification. Springer Berlin / Heidelberg (2011), to appear
21. Polikar, R.: Ensemble based systems in decision making. Circuits and Systems Magazine, IEEE 6(3), 21–45 (2006)
22. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Physical Review E 76(3), 036106 (2007)
23. Ruan, J., Zhang, W.: An efficient spectral algorithm for network community discovery and its applications to biological and social networks. In: ICDM 2007, Seventh IEEE International Conference on Data Mining. pp. 643–648 (2007)
24. Ruan, J., Zhang, W.: Identifying network communities with a high resolution. Physical Review E 77, 016104 (2008)
25. Schaeffer, S.E.: Graph clustering. Computer Science Review 1(1), 27–64 (2007)
26. Schapire, R.E.: The strength of weak learnability. Machine Learning 5, 197–227 (1990)
27. White, S., Smyth, P.: A spectral clustering approach to finding communities in graphs. In: Proceedings of the Fifth SIAM International Conference on Data Mining. pp. 274–285. SIAM (2005)
28. Wolpert, D.H.: Stacked generalization. Neural Networks 5(2), 241 – 259 (1992)
29. Zhu, Z., Wang, C., Ma, L., Pan, Y., Ding, Z.: Scalable community discovery of large networks. In: WAIM '08: Proceedings of the 2008 The Ninth International Conference on Web-Age Information Management. pp. 381–388 (2008)