J Heuristics (2016) 22:759–782
DOI 10.1007/s10732-016-9315-8

Partitioning (hierarchically clustered) complex networks via size-constrained graph clustering

Henning Meyerhenke[1] · **Peter Sanders**[1] · **Christian Schulz**[1]

Received: 11 April 2015 / Revised: 7 July 2016 / Accepted: 9 September 2016 /
Published online: 4 October 2016

Abstract The most commonly used method to tackle the graph partitioning problem in practice is the multilevel metaheuristic. In this paper we introduce *size-constrained label propagation* (SCLaP) and show how it can be used to instantiate both the coarsening phase and the refinement phase of multilevel graph partitioning. We mainly target networks with highly irregular and hierarchically clustered structure (but other network types can be partitioned as well). Additionally, we augment the basic algorithm with several extensions to further improve its speed and/or solution quality. Depending on the configuration of the resulting partitioner using SCLaP, we are able to compute high-quality partitions outperforming all competitors, or instead, to compute similarly good partitions as the best competitor in terms of quality, hMetis, while being an order of magnitude faster. Our fastest configuration partitions the largest real-world graph in our study (it has 3.3 billion edges) with sequential code in about ten minutes while cutting less than half of the edges than the fastest competitor, kMetis.

Keywords Multilevel graph partitioning · Size-constrained graph clustering · Heuristics · Label propagation · Aggressive graph coarsening

A preliminary shorter version of this paper appeared in *Proceedings of 13th International Symposium on Experimental Algorithms* (*SEA 2014*) (Meyerhenke et al. 2014).

✉ Henning Meyerhenke
meyerhenke@kit.edu

✉ Christian Schulz
christian.schulz@kit.edu

Peter Sanders
sanders@kit.edu

[1] Karlsruhe Institute of Technology (KIT), Institute of Theoretical Informatics, Am Fasanengarten 5, 76131 Karlsruhe, Germany

1 Introduction

Graph partitioning (GP) is important for processing very large graphs, e.g. networks stemming from finite element methods, route planning, social networks or web graphs. Often the node set of such graphs needs to be partitioned such that there are few edges between the blocks (node subsets, parts). In particular, when you process a graph in parallel on k processing elements (PEs), you often want to partition the graph into k blocks of (about) equal size. Then each PE owns a roughly equally sized part of the graph. In this paper we focus on a commonly used version of the problem that constrains the maximum block size to $(1 + \epsilon)$ times the average block size and tries to minimize the total cut size, i.e., the number of edges that run between blocks. Such edges are supposed to model the communication at block boundaries between the corresponding PEs. It is well-known that there are more realistic (but more complicated) objective functions involving also the block that is worst and the number of its neighboring nodes (Hendrickson and Kolda 2000), but the cut size has been the predominant optimization criterion. For a formal definition of the problem and related work see Sect. 2. The GP decision problem is NP-complete and there is no approximation algorithm with a constant factor for general graphs (Bui and Jones 1992). Thus heuristic algorithms are used in practice.

A successful heuristic for partitioning large graphs is the *multilevel graph partitioning* (MGP) approach depicted in Fig. 1a, where the graph is recursively *contracted* to obtain smaller graphs which should reflect the basic structure as the input graph. After applying an *initial partitioning* algorithm to the smallest graph, the contraction is undone and, at each level, a *local search* method is used to improve the partition induced by the coarser level.

Complex networks, such as social networks or web graphs, often feature a set of clusters organized in a hierarchical fashion (Lancichinetti et al. 2009). Such graphs have become a focus of investigation (Costa et al. 2011) and target for GP. While partitioning meshes is a mature field, the structure of complex networks poses new challenges to graph partitioning methods. Complex networks are often *scale-free* (many low-degree nodes, few high-degree nodes) and have the small-world property. Small world means that the network has a small diameter, so that the whole graph is

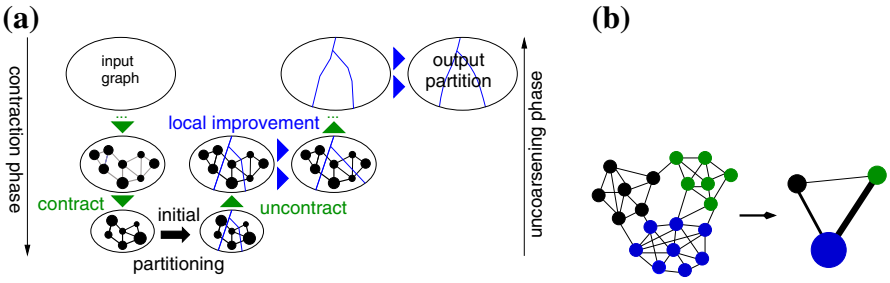

Fig. 1 **a** Sketch of multilevel graph partitioning, the predominant GP heuristic framework in practice. **b** Contraction of a clustering. The clustering of the graph is indicated by the colors. Each cluster of the graph *left* of the *arrow* corresponds to a node (=supervertex) in the graph *right* of the *arrow*

discovered within a few hops from any source node. These two properties distinguish complex networks from traditional meshes and make finding small cuts difficult with established tools. Yet, to cope with massive network data sets in reasonable time, there is a need for parallel algorithms. Their efficient execution requires the partitioning of such networks onto different processing elements (PEs) with high quality.

In this paper we present a new multilevel algorithm designed for partitioning complex networks. Its rationale is to use aggressive cluster-based coarsening and simple, yet effective local search. As we will see, our new *S*ize-*C*onstrained *La*bel *P*ropagation (SCLaP) algorithm serves both purposes. During coarsening, we compute a graph clustering with size-constrained clusters and contract each cluster to a supervertex, also see Fig. 1b. This contraction yields a new level in the multilevel hierarchy. During uncoarsening the same algorithm can be used as a fast local search algorithm (see Sect. 3). Furthermore, we augment the basic SCLaP algorithm by several algorithmic components that are supposed to improve speed and/or quality of the partitioning process (Sect. 4). These extensions include different orders of node traversals, combining several clusterings into one, iterations of the multilevel approach, larger imbalance on coarser levels, and the omission of nodes in the clustering process that are certain not to change their cluster. Our experiments on various complex networks indicate that the presented algorithm, more precisely its different configurations integrated into the partitioning tool KaHIP, is clearly able to improve on the state of the art. Excellent partitioning quality is obtained in a short amount of time (Sect. 5). For example, a web graph with 3.3 billion edges can be partitioned with sequential code in about ten minutes while cutting less than half of the edges than the partitions computed by the established competitor kMetis.

2 Preliminaries

2.1 Basic concepts

Consider an undirected graph $G = (V = \{0, \ldots, n-1\}, E, c, \omega)$ with edge weights $\omega : E \to \mathbb{R}_{>0}$, node weights $c : V \to \mathbb{R}_{>0}, n = |V|$, and $m = |E|$. We extend c and ω to sets, i.e. $c(V') := \sum_{v \in V'} c(v)$ and $\omega(E') := \sum_{e \in E'} \omega(e)$. $N(v) := \{u : \{v, u\} \in E\}$ denotes the *neighbors* of v. We are looking for *blocks* of nodes $V_1, \ldots, V_k$ that *partition* V, i.e., $V_1 \cup \cdots \cup V_k = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. The *balancing constraint* demands that $\forall i \in \{1, \ldots, k\} : c(V_i) \leq L_{\max} := (1+\epsilon)c(V)/k + \max_{v \in V} c(v)$ for some parameter ϵ. The last term in this equation arises because each node is atomic and therefore a deviation of the heaviest node has to be allowed. Note that for unweighted graphs the balance constraint becomes $\forall i \in \{1, \ldots, k\} : |V_i| \leq (1+\epsilon)\lceil \frac{|V|}{k} \rceil$. The objective is to minimize the total *cut* $\sum_{i<j} w(E_{ij})$ where $E_{ij} := \{\{u, v\} \in E : u \in V_i, v \in V_j\}$. We say that a block V_i is *underloaded* if $|V_i| < L_{\max}$ and *overloaded* if $|V_i| > L_{\max}$.

A *clustering* is also a partition; however, when asking for the computation of a clustering, k may not be given in advance and the balance constraint is removed. A *size-constrained clustering* constrains the size of the blocks of a clustering by a given upper bound U such that $c(V_i) \leq U$. Note that by adjusting the upper bound, one can control the number of blocks of a feasible clustering to some extent. For example,

when using $U = 1$, the only feasible size-constrained clustering in an unweighted graph is the clustering where each node forms a block of its own.

A node $v \in V_i$ that has a neighbor $w \in V_j, i \neq j$, is a boundary node. A *matching* $M \subseteq E$ is a set of edges that do not share any common nodes, i.e. the graph (V, M) has maximum degree one. By default, our initial inputs will have unit edge and node weights. However, even those will be translated into weighted problems in the course of the multilevel algorithm. In order to avoid tedious notation, we will denote by G the current graph, before and after an (un)contraction in the multilevel scheme.

2.2 Related work

There has been a *huge* amount of research on graph partitioning so that we refer the reader to Schloegel et al. (2003), Bichot and Siarry (2011) and Buluc et al. (2014) for comprehensive overviews. Here, we focus on issues closely related to our main contributions.

All general-purpose methods that are able to obtain good partitions for large real-world graphs are based on the multilevel principle. The basic idea can be traced back to multigrid solvers for solving systems of linear equations (Southwell 1935), but more recent practical methods are mostly based on graph theoretic aspects, in particular edge contraction and local search. There are many ways to create graph hierarchies by recursive graph coarsening such as matching-based schemes (Walshaw and Cross 2007; Karypis and Kumar 1998; Pellegrini 2012a) or variations thereof (Abou-Rjeili and Karypis 2006) and techniques similar to algebraic multigrid, e.g. (Meyerhenke et al. 2006; Chevalier and Safro 2009). We refer the interested reader to the respective papers for more details. Well-known software packages based on this approach include (Jostle Walshaw and Cross 2007; Metis Karypis and Kumar 1998; Scotch Pellegrini 2012b).

Multilevel graph clustering algorithms and tools are available, too (Dhillon et al. 2007; LaSalle and Karypis 2014; Rotta and Noack 2011; Staudt and Meyerhenke 2016). Since they solve a different problem (and thus proceed somewhat differently even if the techniques are similar), we forego their detailed description. Graph clustering with the label propagation algorithm (LPA), a local cut optimizer, was originally proposed by Raghavan et al. (2007); earlier variants exist as well under different names [e.g. Gilbert et al. (2006)]. LPA has previously been used to partition networks by Ugander and Backstrom (2013). The authors do not use a multilevel scheme, though, and rely on a given or random partition which is improved by combining the unconstrained label propagation approach with linear programming. Their results indicate that this single-level approach does not yield high quality partitions. A distributed ensemble graph clustering algorithm with LPA techniques needs a few hours on a 50 nodes Hadoop system to cluster a web graph called uk-2007 Ovelgonne (2013), whereas our sequential partitioning code processes this graph in about ten minutes.

Recently, Slota et al. (2014) have used LPA for partitioning complex networks as well. Their algorithm, termed PuLP by the authors, differs in one very important aspect: It does not use the multilevel approach. Thus, as we will see in our experimental comparison, the partitioning quality suffers. Also, it may be difficult to adhere to a tight

node balance constraint such as the typical 3 % with PuLP. On the other hand, PuLP can balance according to edges as well, a property our algorithm does not have. Both PuLP (shared memory, Slota et al. 2014) and SCLaP (distributed memory, Meyerhenke et al. 2015) have been implemented in parallel for further acceleration.

Recent work by Kirmani et al. (2013) solves a relaxed version of the GP problem where no strict balance constraint is enforced. The blocks only have to have approximately the same size so that the results are incomparable. Note that the problem is easier than fulfilling a strict balance constraint. Their approach attempts to obtain information on the graph structure by computing an embedding into the coordinate space with multilevel graph drawing. Afterwards partitions are computed using a geometric scheme.

2.3 KaHIP

The software tool KaHIP—Karlsruhe High Quality Partitioning—is a family of GP programs that tackle the balanced GP problem (Sanders and Schulz 2016; Sanders and Schulz 2013). We integrate the techniques described in this paper into KaHIP and use the existing framework to simplify the evaluation of our new algorithms. KaHIP already includes various GP algorithms. Among them is Karlsruhe Fast Flow Partitioner (KaFFPa), which is a matching-based multilevel graph partitioning framework that uses for example flow-based methods and more localized local searches to compute high quality partitions.

3 New complex network partitioning algorithm

Recall that complex networks often feature a set of clusters organized in a hierarchical fashion (Lancichinetti et al. 2009). While these clusters can be of drastically different size, this property can still be exploited twofold for multilevel partitioning. First of all, the cluster hierarchy can be exploited within the multilevel approach. Secondly, since only relatively few edges leave clusters, cluster boundaries may make good boundaries for the blocks of a partition as well. In this section we outline how we use variations of the same algorithm for both aspects.

3.1 Our multilevel partitioning framework

To create a new level of a graph hierarchy, our rationale is to compute a clustering with clusters that are bounded in size and then to *contract* each cluster into a supervertex. This coarsening procedure is repeated recursively until the coarsest graph is small enough. Contracting the clustering works by replacing each cluster with a single node. The weight of this new node (or supervertex) is set to the sum of the weight of all nodes in the original cluster. There is an edge between two nodes u and v in the contracted graph if the two corresponding clusters in the clustering are adjacent to each other in G, i.e. if cluster u and cluster v are connected by at least one edge. The weight of an edge (A, B) is set to the sum of the weight of edges that run between block A

and block B of the clustering. The hierarchy created in this recursive manner is then used by our partitioner. Due to the way the contraction is defined, it is ensured that a partition of the coarse graph corresponds to a partition of the finer graph with the same cut and balance. An example is shown in Fig. 1b.

Note that cluster contraction is an aggressive coarsening strategy. In contrast to previous approaches, it enables us to drastically shrink the size of networks with irregular structure and obtain hierarchies with relatively few levels. Matching-based coarsening, in turn, is not well-suited for complex networks. The latter often contain star-like structures made up of high-degree vertices (so-called *hubs*) connected to many low-degree vertices. In such cases, only few matching edges can be found, so that coarsening is not effective and many levels are created. Even pseudo-matchings, where 2-hop neighbors without a direct connection can mate, seem to alleviate this problem only to some extent. Moreover, as indicated above, the intuition behind our clustering-based technique is that a clustering of the graph (one hopes) contains many edges running inside the clusters and only a few edges running between clusters, which is favorable for the edge cut objective. Regarding complexity, our experiments in Sect. 5 indicate that the number edges per node of the contracted graphs is smaller than the number of edges per node of the input network, and the clustering algorithm we use is fast.

After coarsening, we use initial partitioning routines already existing in KaHIP. Afterwards, when performing local improvement on each level of the multilevel hierarchy, we use two options: (i) Local search algorithms already existing in KaHIP and (ii) our new clustering-based approach. For the latter we need to constrain the cluster sizes to the admissible size of blocks. Now that the partitioning rationale within the multilevel framework is clear, we introduce the size-constrained label propagation algorithm (SCLaP), which is used to compute the aforementioned clusterings in the graph—both for coarsening and for local improvement.

3.2 Label propagation with size constraints

To perform graph clustering with size constraints on the clusters, we are looking for an algorithm that is simple, fast, and is potentially easy to parallelize. The LPA, originally proposed by Raghavan et al. (2007), fulfills these requirements, with the exception that it does not constrain cluster sizes. It is a fast, near-linear time algorithm that locally optimizes the number of edges cut. We outline the algorithm briefly. Initially, each node is in its own cluster, i.e. the initial cluster ID of a node is set to its node ID. The algorithm then works in rounds. In each round, the nodes of the graph are traversed in random order. When a node v is visited, it is *moved* to the block that has the strongest connection to v, i.e. it is moved to the cluster V_i that maximizes $\omega(\{(v,u) \mid u \in N(v) \cap V_i\})$. Ties are broken randomly. The process is repeated until the process has converged.

LPA is known to oscillate for some instances, though. Since we use SCLaP as an auxiliary tool (and do not necessarily need the best possible clustering), we avoid this issue by performing at most ℓ iterations of the algorithm, where ℓ is a tuning parameter. Moreover, we stop the algorithm if less then five percent of the nodes

changed its cluster during one round. Thus, oscillations appearing in the original LPA are no issue for us in practice.

Clearly, LPA can be categorized as a locally greedy edge cut minimizer since each local update tries to reduce the cut as much as possible. Its local optima are often reasonably good clusterings since a cluster ID is likely to propagate through and cover a dense cluster, but unlikely to spread beyond bottlenecks. One LPA round can be implemented to run in $\mathcal{O}(n + m)$ time and the algorithm has been empirically shown to reach a stable solution in only a few iterations, though not mathematically proven to do so for general graphs. Theoretical analysis in clustered random graphs has been done by Kothapalli et al. (2013), who show that LPA is able to detect the clusters in this restricted graph model in just two rounds.

In contrast to the original LPA described above, we have to ensure that each cluster fulfills a size constraint. There are two reason for this. First, consider a clustering of the graph in which the weight of a cluster would exceed $(1+\epsilon)\lceil\frac{|V|}{k}\rceil$, the maximum weight of a block in a valid partition. After contracting this clustering, it would be impossible to find a partition of the contracted graph that fulfills the balance constraint. Secondly, it has been shown that using more balanced graph hierarchies is beneficial when computing high quality graph partitions (Holtgrewe et al. 2010). Thus, to ensure that the clusters do not become too large, we introduce an upper bound $U := \max(\max_v c(v), W)$ for the cluster sizes and thus obtain the SCLaP algorithm for solving a size-constrained clustering problem. For coarsening we set W to $L_{\max}/f$, where f is a tuning parameter. When the algorithm starts to compute a graph clustering on the input graph, the constraint is fulfilled since each of the clusters contains exactly one node. A neighboring cluster V_ℓ of a node v is called *eligible* if V_ℓ will not become overloaded once v is moved to V_ℓ. Now when we visit a node v, we deviate for SCLaP from original LPA by moving v to the cluster with the strongest connection to v that is also eligible. Hence, after moving a node, the size of each cluster is still smaller than or equal to U. Moreover, after contracting the clustering, the weight of each node is smaller or equal to U. One round of the modified version of the algorithm can still run in linear time by using an array of size $|V|$ to store the cluster sizes.

To build the multilevel hierarchy, we repeat the process of computing and contracting a size-constrained clustering recursively. We stop the recursion when the number of remaining nodes is smaller than $\max(60k, n/(60k))$, values determined from experimental experience. The coarsest graph is initially partitioned by the initial partitioning algorithms provided in KaHIP. That means each node of the coarsest graph is assigned to a block. KaHIP uses a multilevel recursive bisection algorithm to create an initial partitioning (Schulz 2013). Afterwards, the solution is transferred to the next finer level by assigning each node of the finer graph to the block of its coarse representative. As is customary, we then try to improve the solution on each level with suitable local improvement methods.

Note that, with a small change, SCLaP can be used as a local improvement method as well. By using a different size-constraint—the constraint $W := L_{\max}$ of the original partitioning problem—SCLaP becomes a simple and fast local search algorithm for improving the edge cut of a solution on the level under consideration. However, one has to make additional small modifications to handle overloaded blocks. We modify the cluster/block selection rule when we use the algorithm as local search algorithm

in case that the current node v under consideration is from an overloaded block V_ℓ. In this case it is *moved* to the eligible block that has the strongest connection to v without considering the block V_ℓ it is contained in, i.e. it is moved to the block V_i that maximizes $\omega(\{(v,u) \mid u \in N(v) \cap V_i, i \neq \ell\})$. This way it is ensured that the move improves the balance of the partition (at the cost of the number of edges cut). Experiments in Sect. 5 show that the algorithm is a fast alternative to the local search algorithms provided by KaFFPa. Moreover, let us emphasize that the algorithm has a large potential to be efficiently parallelized.

4 Algorithmic extensions

Clearly, the combination of the multilevel framework and SCLaP leaves several degrees of freedom when implementing single parts of these generic algorithmic modules. Hence, in this section we present numerous algorithmic extensions to the approach presented above. They include using different node traversal orderings for SCLaP, combining multiple clusterings into one clustering, iterated multilevel cycles, allowing additional amounts of imbalance on coarse levels of the multilevel hierarchy, and a method to improve the speed of the algorithm. In our experimental study in Sect. 5, we will explore their effect on partitioning quality and running time.

4.1 Node ordering for SCLaP

The original LPA traverses the nodes in a random order and moves a node to a cluster with the strongest connection in its neighborhood. As an alternative option to the random order, we also use the ordering induced by the node degree (in increasing order) for SCLaP. That means that in the first round of SCLaP, nodes with small node degree can change their cluster before nodes with a large node degree. Intuitively, this ensures that there is already a meaningful cluster structure when SCLaP chooses the cluster of a high degree node. We also tried other node orderings such as weighted node degree. The overall solution quality and running time are comparable to the degree ordering so that we omit the other orderings here.

4.2 Ensemble clusterings

In machine learning, ensemble methods combine multiple weak classification algorithms to obtain a strong classifier. This main principle is also valid in our clustering scenario, where previous work combined several base clusterings from different LPA runs. These base clusterings are used to decide whether pairs of nodes should belong to the same cluster (Ovelgonne and Geyer-Schulz 2013; Staudt and Meyerhenke 2016). We follow the idea to get better clusterings for the coarsening phase of our multilevel algorithm.

Given a number of clusterings, the *overlay clustering* is a clustering in which two nodes belong to the same cluster if and only if they belong to the same cluster in each of the input clusterings, for an example see Fig. 2. Intuitively, if all of the input clusters

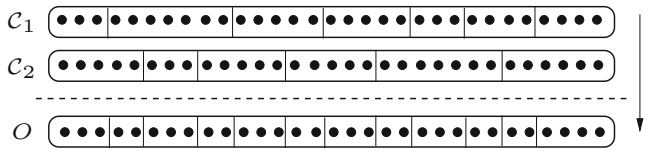

Fig. 2 Two clusterings (*top*) of a graph are combined to one overlay clustering (*bottom*). Two nodes belong to the same cluster in the overlay clustering if and only if they belong to the same cluster in each of input clusterings

agree that two nodes belong to the same cluster, the two nodes belong together with high confidence. In our ensemble approach we use the clusterings obtained by SCLaP as input to compute the overlay clustering. It is easy to see that the number of clusters in the overlay clustering cannot decrease compared to the number of clusters in each of the input clusterings. Moreover, the overlay clustering is feasible w.r.t. to the size constraint if each of the input clusterings is feasible.

Given ℓ clusterings $\{\mathcal{C}_1, \ldots, \mathcal{C}_\ell\}$, we use the following approach to compute the overlay clustering iteratively. Initially, the overlay clustering $\mathcal{O}$ is set to the clustering $\mathcal{C}_1$. We then iterate through the remaining clusterings and incrementally update the current solution $\mathcal{O}$. To this end, we use pairs of cluster IDs (i, j) as a key in a hash map $\mathcal{H}$, where i is a cluster ID of $\mathcal{O}$ and j is a cluster ID of the current clustering $\mathcal{C}$. We then iterate through the nodes and initialize a counter c to zero. Let v be the current node. If the pair $(\mathcal{O}[v], \mathcal{C}[v])$ is not contained in $\mathcal{H}$, we set $\mathcal{H}(\mathcal{O}[v], \mathcal{C}[v])$ to c and increment c by one. Afterwards, we update the cluster ID of v in $\mathcal{O}$ to $\mathcal{H}(\mathcal{O}[v], \mathcal{C}[v])$. At the end of the algorithm, c is equal to the number of clusters contained in the overlay clustering. Note that it is possible to compute the overlay clustering directly by hashing ℓ-tuples (Staudt and Meyerhenke 2016). However, we choose the simpler approach here since it is not a bottleneck in terms of running time—the computation of a clustering itself already takes near-linear time.

4.3 Iterated multilevel algorithms

A common approach to obtain high quality solutions with a multilevel algorithm using local search is to use random restarts with different seeds and use the best partition that has been found. However, one can usually do better in the context of multilevel algorithms by transferring the solution of the previous multilevel iteration down the hierarchy—one could say following the idea of iterated local search. Actually, the notion of *V-cycles* stems from multigrid methods for linear systems. A V-cycle simply refers to one multilevel iteration (due to its shape of a large graph at the top and successively smaller graphs on the lower hierarchy levels). In the GP context it has been introduced by Walshaw (2004). Later it has been augmented to more complex cycles (Sanders and Schulz 2011). Performing several V-cycles means to iterate the multilevel approach, each time starting with the solution produced by the previous cycle.

To adopt the iterated multilevel technique for our new coarsening scheme, we have to ensure that cut edges are not contracted after the first multilevel iteration. (The

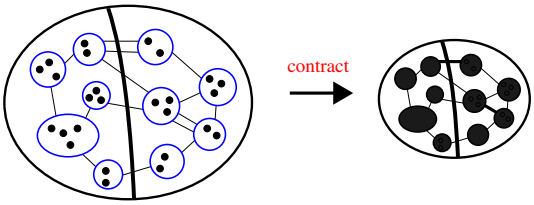

Fig. 3 Schematic drawing of a graph that is partitioned into two blocks (indicated by the thick *black line*). Each of the clusters (indicated by the *blue circles* on the *left hand side*) is a subset of one block of the input partition. Hence, cut edges of the input partition are not contracted, when the clustering is contracted. Moreover, the partition of the graph on the *left hand side* can be transformed into a partition of the contracted graph that has the same objective and balance

above mentioned previous work did this for matching-based coarsening, too.) We do this by modifying SCLaP such that each cluster of the computed clustering is a subset of a block of the input partition. In other words, each cluster only contains nodes of one unique block of the input partition. Hence, when contracting the clustering, every cut edge of the input partition will remain. Recall that LPA initially puts each node in its own cluster so that in the beginning of the algorithm each cluster is a subset of one unique block of the input partition. To keep this property during the course of SCLaP, we restrict the movements performed by the algorithm, i.e. a node is moved to an eligible cluster with the strongest connection in its neighborhood that is in the same block of the input partition as the node itself. More precisely, let $V = U_1 \cup \ldots \cup U_k$ be the partition of the graph in the current level of the multilevel hierarchy. When a node $v \in U_\ell$ is visited, it is moved to an eligible cluster $V_i \subseteq U_\ell$ that maximizes $\omega(\{(v,u) \mid u \in N(v) \cap V_i\})$. Figure 3 shows a clustering of the graph that respects a given partition.

Note that our approach ensures non-decreasing partition quality if the local search algorithm guarantees that solution quality never decreases. Moreover, due to the randomization of the coarsening, the hierarchies created in later cycles are usually different, which introduces further diversification for local search.

4.4 Allowing larger imbalances on coarse levels

It is well-known that temporarily allowing larger imbalance can be useful for creating better partitions (Walshaw and Cross 2000; Sanders and Schulz 2013). Allowing an additional amount of imbalance $\hat{\epsilon}$ means that the balance constraint is relaxed to $(1 + \epsilon + \hat{\epsilon})\lceil \frac{|V|}{k} \rceil$, where ϵ is the original imbalance parameter and $\hat{\epsilon}$ is a parameter that has to be set appropriately. Walshaw and Cross (2000) estimate the amount of additional allowed imbalance on each hierarchy level by considering the number of boundary nodes of perfectly partitioned 2D and 3D meshes. We adopt a simplified approach in this context and decrease the amount of additional allowed imbalance level-wise. No additional amount of imbalance is allowed on the finest level. To be more precise, let the levels of the hierarchy be numbered in increasing order $G_1, \ldots, G_q$ where G_1 is the input graph G and G_q is the coarsest graph. The amount of allowed imbalance on a coarse level $\ell > 1$ is set to $\hat{\epsilon}_\ell = \delta/(q - \ell + 1)$, where δ is a tuning

parameter. No additional amount of imbalance is allowed on the finest level. Moreover, we only allow a larger amount of imbalance during the first V-cycle. This approach allows the partitioning algorithm to explore larger regions of the search space, but does not jeopardize the adherence to the balance constraint in the final result.

4.5 Active nodes

SCLaP looks at every node in each round of the algorithm. Assume for now that we run LPA (i.e. SCLaP without a size constraint). After the first round of the algorithm, a node can only change its cluster if one or more of its neighbors changed its cluster in the previous round (for the sake of the argument we assume that ties are broken the same way). We follow (Staudt and Meyerhenke 2016) and adapt their LPA active nodes approach for use in SCLaP. It keeps track of nodes that cannot change their cluster due to an unchanged neighborhood. A node is called *active* if at least one of its neighbors changed its cluster in the *previous round*. In the first round all nodes are active. The original LPA is then modified so that only active nodes are considered for movement (Staudt and Meyerhenke 2016). Note that this modification does not change the result of LPA, but is likely to improve its running time.

Note that things are different when a size constraint has to be fulfilled. It is possible that the best movement of a node v cannot be performed due to the size constraint, i.e. v cannot be moved to the best cluster since it would overload the cluster. If later during that round a non-adjacent node is removed from the cluster, then the movement would be possible. However, it will only be considered by the modified algorithm if one of the neighbors of v changed its cluster. Experiments indicate that this slightly decreases the overall solution quality.

The data structures used in our active nodes implementation slightly differ from those in previous work (Staudt and Meyerhenke 2016). We additionally include two FIFO queues, one that stores the active nodes of the current round and one for the next round, as well as two bit vectors that store if a node is already contained in the respective queue. When a round is finished, all entries in the bit vector associated with the queue of the current round are false (and the queue is empty). Then, the queues change their roles, i.e. pointers to the queues and the associated bit vectors are swapped. A round of this modified algorithm can be implemented with running time linear in the amount of edges incident to the number of active nodes—whose number is usually significantly smaller than $|V|$ in later iterations.

5 Experiments

5.1 Methodology and settings

We have implemented the algorithm described above using C++ extending the KaFFPa multilevel partitioning framework, compiled to sequential code with g++ 4.8.2. The KaFFPa framework has different configurations. In this work, we consider the Strong and the Eco configuration of KaFFPa. The aim of KaFFPaEco is to be fairly fast and to compute partitions of high quality, whereas KaFFPaStrong targets very high solution

quality. To evaluate interactions and relative importance of our algorithmic improvements, we start with KaFFPaEco and enable the algorithmic components described in this paper step-by-step. When we start with KaFFPaStrong, we only look at the final algorithm including all algorithmic components to save running time and KaFFPaStrong itself. Unless otherwise mentioned, we perform ten repetitions for each configuration of the algorithm and report the arithmetic average of cut sizes and running times. When further averaging over multiple instances, we use the geometric mean in order to give every instance a comparable influence on the final score. For the number of partitions k, we choose the values used in Walshaw and Cross (2000): 2, 4, 8, 16, 32, 64.

We performed a large amount of experiments to tune our algorithm's basic parameters. Their tedious detailed description is omitted. The instances used for the parameter tuning are highlighted by italics in Table 1. For most of the parameters, except the cluster size-constraint factor f, we get the predictable effect that more work yields better solutions albeit at a decreasing return on investment. The interesting and influential choices are the algorithmic components which we evaluate in the following section. We use the following basic parameters of the algorithm which turned out to work well: the number of maximum label propagation iterations ℓ during coarsening and uncoarsening is set to 10, the factor f of the cluster size-constraint is set to 18, the number of V-cycles is set to three and the number of ensemble clusterings used to compute an ensemble clustering is set to 18 if k is smaller than 16, to 7 if k is 16 or 32 and to 3 if k is larger than 32.

Our new approach is not only compared to KaFFPa, but also to packages developed without our participation. Unless otherwise noted, our default value for the allowed imbalance is 3 % since this is a common setting in previous work [e.g. in Walshaw and Cross (2000)] and the default value in kMetis, a widely used graph partitioner. We mostly evaluate our algorithms on 25 graphs that have been collected from various public sources (Bader et al. 2014; Laboratory of Web Algorithms 2016; Leskovec 2016). They include a number of citation and social networks as well as web graphs. Table 1 summarizes the basic properties of these graphs. We use Test set I (medium-sized and large graphs) to evaluate the performance of different algorithms in Sect. 5.2 and compare the performance of the fastest algorithms on Test Set II (huge graphs) in Sect. 5.3. Additional comparisons to PuLP (Slota et al. 2014) are performed in Sect. 5.4. For this comparison we use the instances also employed in the PuLP paper.

We use two machines for our experiments, chosen for their memory size, not for their CPU performance (since our code is sequential): *Machine A* is used for our experimental evaluation in Sect. 5.2. It is equipped with two Intel Xeon E5-2670 Octa-Core processors (Sandy Bridge), which run at a clock speed of 2.6 GHz. The machine has 64 GB main memory, 20 MB L3-Cache and 8x256 KB L2-Cache. *Machine B* is used for the experiments on the huge networks in Sects. 5.3 and 5.4. It is equipped with four Intel Xeon E5-4640 Octa-Core processors (Sandy Bridge) running at a clock speed of 2.4 GHz. The machine has 1 TB main memory, 20 MB L3-Cache and 8x256 KB L2-Cache.

Table 1 Basic properties of the graphs test set

Graph	n	m	Ref.	Max deg.	Avg. deg	Type of network
Test set I (medium-sized and large graphs)						
p2p-Gnutella04	6405	29,215	Leskovec (2016)	103	9	Filesharing network
wordassoc-2011	10,617	63,788	Laboratory of Web Algorithms (2016)	332	12	Free word association network
PGPgiantcompo	10,680	24,316	Leskovec (2016)	205	4	Conn. comp. in PGPusers network
email-EuAll	16,805	60,260	Leskovec (2016)	3282	7	Network of connections via email
as-22july06	22,963	48,436	Bader et al. (2014)	2390	4	Autonomous systems in internet
soc-Slashdot0902	28,550	379,445	Leskovec (2016)	2272	26	News network
loc-brightkite	56,739	212,945	Leskovec (2016)	1134	7	location-based Friendship
enron	69,244	254,449	Laboratory of Web Algorithms (2016)	1634	7	network of Connections via email
loc-gowalla	196,591	950,327	Leskovec (2016)	14730	9	location-Based friendship
coAuthorsCiteseer	227,320	814,134	Bader et al. (2014)	1372	7	Citation network
wiki-Talk	232,314	≈1.5M	Leskovec (2016)	100,029	12	Network of user interactions
citationCiteseer	268,495	≈1.2M	Bader et al. (2014)	1318	8	Citation network
coAuthorsDBLP	299,067	977676	Bader et al. (2014)	336	6	Citation network
cnr-2000	325,557	≈2.7M	Bader et al. (2014)	18,236	16	Hyperlink network of webpages
web-Google	356,648	≈2.1M	Leskovec (2016)	5235	11	Hyperlink network of webpages
coPapersCiteseer	434,102	≈16.0M	Bader et al. (2014)	1188	73	Citation network
coPapersDBLP	540,486	≈15.2M	Bader et al. (2014)	3299	56	Citation network
as-skitter	554,930	≈5.8M	Leskovec (2016)	29,874	20	Internet service provider network
amazon-2008	735,323	≈3.5M	Leskovec (2016)	1077	9	Similarity among books
eu-2005	862,664	≈16.1M	Bader et al. (2014)	68,963	37	Hyperlink network of web pages
in-2004	≈1.3M	≈13.6M	Bader et al. (2014)	21,869	19	hyperlink network of web pages

Table 1 continued

Graph	n	m	Ref.	Max deg.	Avg. deg	Type of network
Test set II (huge web graphs)						
uk-2002	≈18.5M	≈262M	Leskovec (2016)	194955	28	Hyperlink network of web pages
arabic-2005	≈22.7M	≈553M	Leskovec (2016)	575628	48	Hyperlink network of web pages
sk-2005	≈50.6M	≈1.8G	Leskovec (2016)	8563816	71	Hyperlink network of web pages
uk-2007	≈106M	≈3.3G	Leskovec (2016)	975419	62	Hyperlink network of web pages

5.2 Main results

In this section we compare our algorithms against other frequently used publicly available tools. We compare the average and minimum edge cut values produced by all of these tools on the large graphs from Table 1, as well as their average running time on these graphs. Detailed experimental data for a representative subset of the graphs, number of algorithm configurations, and $k \in \{2, 16, 64\}$ are presented in the Appendix (Tables 4, 5). Experiments have been performed on machine A. For the comparison we used the k-way variant of hMetis 2.0 (p1) (Karypis et al. 1999), kMetis 5.1 (Karypis and Kumar 1998), and Scotch 6.0.0 (Pellegrini 2012a) employing the quality option. hMetis is actually a hypergraph partitioner, but it is recommended by its main author for high-quality GP as well. In contrast to our algorithm, hMetis and Scotch often produce imbalanced partitions. Hence, these tools have a slight advantage in the following comparisons because we do not disqualify imbalanced solutions. In case of hMetis the partitions are imbalanced in 105 out of 1260 cases (up to 12 % imbalance) and in case of Scotch the partitions are imbalanced in 218 out of 1260 cases (up to 226 % imbalance). Note that the latest version of kMetis (5.1) improved the balancing on complex networks by integrating a 2-hop matching algorithm that can match nodes if they share neighbors.

In addition to the existing default configurations KaFFPaEco and KaFFPaStrong, we have six new base configurations, PCBC-Eco, PCBC-Fast, and PCBC-Strong as well as CCBC-Eco, CCBC-Fast, and CCBC-Strong. All of these configurations use the new clustering-based coarsening scheme with the degree based node ordering. The configurations having an Eco in their name use the refinement techniques as used in KaFFPaEco, and the configurations having the word Fast in their name use SCLaP as local search algorithm instead. As initial partitioning algorithm we use KaFFPa, which employs multilevel recursive bipartitioning. The configurations starting with PCBC (partial cluster-based coarsening) use the matching-based approach during this initial partitioning phase, whereas the configurations starting with CCBC (complete cluster-based coarsening) use the clustering based coarsening scheme also during initial partitioning. We add additional letters to the base configuration name for each additional algorithmic component that is used. For example, PCBC-EcoV/B/E/A is

Table 2 Geometric mean of improvement ratios over Scotch (average cut, best cut) as well as geometric mean of running time results for diverse algorithms on Test set I

Algorithm	Geom. mean $\left(\frac{\text{avg. scotch cut}}{\text{avg. algorithm cut}}\right)$	Geom. mean $\left(\frac{\text{best scotch cut}}{\text{best algorithm cut}}\right)$	t [s]
PCBC-EcoR	1.461	1.444	10.2
PCBC-Eco	1.561	1.516	8.6
PCBC-EcoV	1.589	1.543	14.3
PCBC-EcoV/B	1.625	1.593	15.5
PCBC-EcoV/B/E	1.622	1.588	46.6
PCBC-EcoV/B/E/A	1.613	1.580	41.9
PCBC-FastR	1.410	1.397	4.7
PCBC-Fast	1.525	1.481	3.9
PCBC-FastV	1.553	1.508	5.7
PCBC-FastV/B	1.488	1.461	5.8
PCBC-FastV/B/E	1.522	1.489	28.4
PCBC-FastV/B/E/A	1.522	1.487	24.4
CCBC-Fast	1.517	1.480	1.5
CCBC-FastV	1.547	1.504	3.0
CCBC-EcoV/B	1.609	1.581	11.5
PCBC-Strong	1.744	1.670	422.1
CCBC-Strong	1.751	1.677	296.4
KaFFPaEco	1.222	1.211	36.2
KaFFPaStrong	1.662	1.610	640.8
Scotch	1.000	1.000	10.6
kMetis	1.458	1.426	0.4
hMetis	1.605	1.537	107.4

Geometric means are used to give each instance a comparable influence. Regarding solution quality, higher values are better. Configurations prefixed with PCBC perform cluster-based coarsening until the coarsest graph and matching-based coarsening for initial partitioning of this coarsest graph. CCBC configurations perform cluster-based coarsening for initial partitioning as well. Further configuration abbreviations: V-cycles (V), additional imbalance on coarse levels (B), ensemble clusterings (E), active nodes during coarsening (A), random node ordering (R)

based on PCBC-Eco and uses V-cycles (V), additional imbalance on coarse levels (B), ensemble clusterings (E), and the active node approach for SCLaP not only during local search but also during coarsening (A). Random node ordering (R) is included for completeness as well. PCBC-Strong uses additional imbalance on coarse levels and ensemble clusterings for coarsening. It uses the refinement techniques of KaFFPaStrong. CCBC-Strong is the same as PCBC-Strong but uses the cluster based partitioning approach for initial partitioning.

Table 2 summarizes the results of our experiments. First of all, and maybe most importantly, we observe large running time *and* quality improvements when switching from the matching-based coarsening scheme in KaFFPaEco to SCLaP for coarsening (KaFFPaEco vs. PCBC-EcoR). This step already improves running time by a factor of

3.5 and solution quality by roughly 20 %. Moreover, our experiments indicate that even the solutions on the coarsest graphs are much better than before. Additionally enabling the node ordering heuristics yields an extra 8 % improvement in solution quality and 20 % improvement in running time (PCBC-EcoR vs PCBC-Eco and PCBC-FastR vs. PCBC-Fast). This is due to the fact that the node ordering heuristic improves the results of SCLaP by computing clusterings that have fewer edges between the clusters. As a consequence, the contracted graphs have a smaller amount of total edge weight, which in turn yields better graph hierarchies for partitioning. Performing additional V-cycles and allowing additional imbalance on coarse levels improves solution quality but also increases running time (PCBC-Eco vs. PCBC-EcoV vs. PCBC-EcoV/B). However, allowing additional imbalance worsens the solution quality if SCLaP is used as a refinement algorithm (PCBC-FastV vs. PCBC-FastV/B). This is caused by the poor ability of label propagation to balance imbalanced solutions. Using ensemble clusterings can enhance solution quality (PCBC-FastV/B vs. PCBC-FastV/B/E), but does not have to (PCBC-EcoV/B vs. PCBC-EcoV/B/E).

Due to the size constraints, the clusterings computed by SCLaP with active nodes approach have more edges between the clusters than SCLaP without this acceleration technique. Hence, the active nodes approach improves running time but also degrades solution quality slightly. Additional speedups are obtained when the label propagation coarsening is also used during initial partitioning, but sometimes solution quality is worsened slightly. For example, we achieve a 2.7-fold speedup when switching from PCBC-Fast to CCBC-Fast, and switching from PCBC-Strong to CCBC-Strong improves solution quality slightly while improving running time by 42 %.

The comparison to other established partitioning packages is very often in favor of our new approach. On average, the configuration CCBC-EcoV/B yields comparable quality to hMetis while being an order of magnitude faster. When repeating this configuration ten times and taking the best cut, we get an algorithm that has comparable running time to hMetis and improves quality by 6 %. Our best configuration CCBC-Strong cuts 9 % less edges than hMetis and 20 % less edges than kMetis. In this case, hMetis is a factor 3 faster than CCBC-Strong. However, when taking the best cut out of ten repetitions of hMetis and comparing it against the average results of CCBC-Strong, we still obtain 6 % improvement. Overall, Scotch produces the worst partitioning quality among the competitors. This is hardly a surprise as Scotch has been designed for meshes and possesses no adaptations to complex networks. It cuts 75 % more edges than our best configuration CCBC-Strong. The fastest tool for Test set I is kMetis. It is about 3.5 times faster than our fastest configuration CCBC-Fast, but it also cuts more edges than CCBC-Fast.

5.3 Huge web graphs

In this section our experiments focus on the huge web graphs from Table 1 (which have up to 3.3 billion undirected edges). To use our shared experimental computing platforms economically, we fix the number of blocks to $k = 16$ and focus on the two fast configurations CCBC-Fast and CCBC-FastV. We further speed up CCBC-Fast and CCBC-FastV by only performing three label propagation iterations (instead of ten)

Table 3 Average edge cuts, best cuts and average running times on Test set II (huge networks) for $k = 16$

Graph	Arabic-2005			Uk-2002		
Algorithm	Avg. cut	Best cut	Avg. t [s]	Avg. cut	best cut	avg. t [s]
CCBC-Fast	1,914,988	1,865,702	111.2	1,467,201	1,432,420	71.7
CCBC-FastV	1,845,030	1,790,562	334.3	1,428,011	1,388,692	215.9
kMetis	3,580,993	3,500,691	99.6	2,459,260	2,412,660	63.7
	sk-2005			uk-2007		
CCBC-Fast	23,005,842	20,340,871	387.1	4,341,357	4,101,463	626.5
CCBC-FastV	19,818,520	18,178,517	1166.4	4,190,953	3,991,890	1756.4
kMetis	19,425,531	18,560,644	405.3	11,441,291	10,858,662	827.6

during coarsening. For comparison we also run kMetis and Scotch on these graphs, whereas we did not run hMetis due to its large running times on Test set I. Scotch crashes when trying to partition sk-2005 and uk-2007-05, and did not finish after 24 hours of computations on the other two graphs.

Table 3 summarizes the results. In three out of four cases our algorithms outperform kMetis in the number of edges cut by large factors while having a comparable running time. Moreover, in every instance the best cut produced by the algorithm CCBC-FastV is better than the best cut produced by kMetis. On average, we obtain 74 % improvement over kMetis. The largest improvements are obtained obtained on the web graph uk-2007. Here, CCBC-Fast cuts a factor 2.6 less edges than kMetis and is about 30 % faster. Such huge improvements are rarely seen for traditional inputs such as meshes and show the potential of the new approach. Intriguingly, after only one coarsening step the number of nodes is reduced by two orders of magnitude. This aggressive coarsening is certainly one important reason for the speed of CCBC-Fast. To put our results further into perspective, it is noteworthy that on all graphs (except sk-2005) already the initial partition is much better than the final result of kMetis. For example, on average the initial partition of uk-2007 cuts 4.8 million edges, which improves by a factor of 2.4 on kMetis.

5.4 Additional comparison to PuLP

PuLP (Slota et al. 2014) is a shared-memory parallel multi-constraint and multi-objective partitioning tool. Its code is not publicly available yet, but was kindly provided to us by its authors. Since PuLP is actually targeted at a somewhat different problem, our comparison is less extensive and separated from the traditional graph partitioners. To match our scenario, we restrict PuLP and use its single constraint variant with a single thread. As test instances we use the six graphs used in the PuLP paper except DBpedia, since DBpedia mostly contains self-loops. As in the PuLP paper, both algorithms receive a 10 % imbalance threshold for the partition. PuLP's solutions do not fulfill the balance constraint on one of the graphs (rmat), the blocks

are twice as large as the average block size. Single-threaded PuLP is on average 39 % faster than CCBC-Fast, which is to be expected due to the omission of a multilevel approach. The price is paid in terms of quality in our scenario. CCBC-Fast computes better cuts on all of the instances that PuLP can solve. On average, our algorithm cuts 21 % fewer edges on these instances and even the worst solutions of CCBC-Fast are always better than the best solutions of PuLP.

6 Conclusion and future work

Current state-of-the-art multilevel graph partitioners have difficulties when partitioning massive complex networks, at least partially due to ineffective coarsening. Thus, guided by techniques used in complex network clustering, we have devised a new scheme for coarsening based on the contraction of clusters. These clusters are derived from a label propagation scheme for which we have introduced a size constraint to control the aggressiveness of the coarsening. Additional algorithmic adaptations, also for local improvement, further improve running time and/or solution quality.

The different configurations of our algorithm give users a gradual choice between the best quality currently available and very high speed. The quality of the best competitor, hMetis, is already reached with an order of magnitude less running time. The strengths of our techniques particularly unfold for huge web graphs, where we significantly improve on kMetis in terms of solution quality (up to 2.6×) with a mostly comparable running time.

It seems plausible that cluster-based coarsening can be beneficial in other multilevel algorithms for complex networks as well, e.g. in graph drawing. Thus, we invite other researchers to benefit from our work. The algorithms presented here have been made available for download within the KaHIP graph partitioning framework. Moreover, partitioning is a key prerequisite for efficient large-scale parallel graph algorithms. As huge networks become abundant, there is a need for their parallel analysis. We have started to transfer some of the techniques presented in this paper to an MPI parallel version for compute clusters with distributed memory (Meyerhenke et al. 2015). As part of future work, we want to improve this parallel code further, in particular by integrating more techniques introduced here.

Acknowledgments This work was partially supported by Deutsche Forschungsgemeinschaft under Grants DFG SA 933/10-1 and DFG ME 3619/2-1

Appendix

See Tables 4 and 5.

Table 4 Average cuts for $k \in \{2, 16, 64\}$ and a representative graph set and a number of configurations

Graph	k	CCBC-EcoV/B	CCBC-strong	KaFFPaEco	KaFFPaStrong	Scotch	Metis	hMetis
amazon-2008	2	74439.2	70138.7	122529.2	72293.4	95758.1	75062.5	**67573.9**
amazon-2008	16	244206.3	233244.6	318764.7	232590.4	314113.3	263878.5	**231664.6**
amazon-2008	64	357917.9	**347762.4**	389604.9	348054.0	443156.8	400256.3	350539.5
coPapersCiteseer	2	291724.9	259546.1	488199.3	**252064.7**	422023.8	397619.3	274648.7
coPapersCiteseer	16	811676.1	**786877.2**	1111842.7	799219.9	1871086.1	1057051.4	795277.2
coPapersCiteseer	64	1067429.7	**1047782.5**	1245428.5	1118826.2	2237664.0	1321233.9	1111011.5
coPapersDBLP	2	522426.7	504117.2	848485.0	**473742.1**	759741.2	604937.1	487423.6
coPapersDBLP	16	1463265.4	1413583.5	2068121.1	**1409829.8**	2774358.7	1741097.2	1422171.5
coPapersDBLP	64	1908256.9	**1883190.7**	2141745.4	1927330.9	3315609.0	2246737.2	1961393.8
email-EuAll	2	757.2	789.2	978.7	760.6	727.0	966.4	**654.1**
email-EuAll	16	23015.5	**18864.4**	23466.3	20424.2	21741.1	22618.6	21095.9
email-EuAll	64	34719.5	**32203.2**	35312.9	32644.7	33871.0	33519.6	33176.4
enron	2	9176.1	10758.3	6084.6	**4970.7**	5881.8	7045.3	5409.4
enron	16	71469.2	66267.3	72302.4	**64006.6**	74402.8	75060.3	75723.6
enron	64	98570.7	**94133.2**	105893.0	94612.9	104728.0	99434.3	107004.0
eu-2005	2	24205.6	**19483.1**	119590.9	51731.4	654972.9	33318.5	34133.3
eu-2005	16	350500.0	**309087.2**	477165.6	345854.4	1037736.3	365451.0	335947.2
eu-2005	64	2025148.5	1865711.6	2116200.8	**1840963.1**	2513880.6	2126746.3	1930654.4
in-2004	2	3458.7	**3253.1**	12324.9	5173.7	113785.1	6922.2	8431.1
in-2004	16	19597.9	**17397.2**	54015.4	31632.9	229394.1	28652.8	29519.7

Table 4 continued

Graph	k	CCBC-EcoV/B	CCBC-strong	KaFFPaEco	KaFFPaStrong	Scotch	Metis	hMetis
in-2004	64	46904.4	**35159.6**	71121.9	48798.2	222074.8	66863.8	53363.2
loc-brightkite	2	20930.1	18781.4	29354.1	18064.6	20635.5	19426.1	**17706.9**
loc-brightkite	16	55278.1	53467.1	84016.9	53955.7	60070.6	56424.2	**53297.3**
loc-brightkite	64	74522.0	**70329.0**	87452.7	70809.0	82042.3	74585.9	73107.9
loc-gowalla	2	66050.5	53200.4	73755.6	**52290.9**	65397.4	59021.0	60701.6
loc-gowalla	16	242733.8	**215137.1**	348950.2	216882.1	235626.3	237362.7	232959.6
loc-gowalla	64	338191.8	**299586.7**	410318.2	302696.1	340536.9	335828.5	329020.4
PGPgiantcompo	2	**361.6**	365.9	627.9	392.5	690.7	411.9	364.2
PGPgiantcompo	16	1607.6	**1501.7**	1710.8	1555.7	2066.8	1815.3	1545.4
PGPgiantcompo	64	3040.7	2859.1	3176.9	2862.0	3298.7	3177.7	**2805.9**

Best results are highlighted by bold font

Table 5 Best cuts for $k \in \{2, 16, 64\}$ and a representative graph set and a number of configurations

Graph	k	CCBC-EcoV/B	CCBC-strong	KaFFPaEco	KaFFPaStrong	Scotch	Metis	hMetis
amazon-2008	2	72002	67939	99060	67446	91942	73921	**67372**
amazon-2008	16	238105	230246	305992	231130	307663	259179	**228853**
amazon-2008	64	354760	**344394**	382923	345316	427747	397619	348997
coPapersCiteseer	2	276644	**240496**	473920	243750	418265	382375	269770
coPapersCiteseer	16	801082	**776048**	1069176	785017	1783794	1038917	786245
coPapersCiteseer	64	1055809	**1041678**	1228614	1100649	2217913	1309505	1097703
coPapersDBLP	2	485385	499830	806719	**461819**	735569	588299	474290
coPapersDBLP	16	1448367	1396096	2023936	**1391497**	2681239	1714646	1411077
coPapersDBLP	64	1897951	**1872070**	2112868	1913078	3234255	2228405	1950931
email-EuAll	2	647	698	846	683	694	688	**627**
email-EuAll	16	19663	**18343**	23184	19854	19371	21996	20963
email-EuAll	64	34196	**31966**	34716	32383	33221	32952	32974
enron	2	6159	9106	4864	**4774**	5749	5916	5139
enron	16	68909	65578	68969	**62353**	72595	73393	74865
enron	64	97383	**92937**	104228	94072	103062	98204	106557
eu-2005	2	19691	**18615**	75347	40818	628914	27472	24190
eu-2005	16	311509	**279268**	417406	305367	930238	304620	313037
eu-2005	64	1997788	1843692	2050160	**1810136**	2353779	2049287	1917151
in-2004	2	3225	**3120**	9686	3981	111526	5939	6594
in-2004	16	18637	**16915**	50059	27450	172073	26872	26975
in-2004	64	36016	**33978**	68455	44816	171467	59478	49714
loc-brightkite	2	19871	18151	28965	17699	19389	18993	**17554**
loc-brightkite	16	55011	53003	82750	53705	58538	55588	**52961**

Table 5 continued

Graph	k	CCBC-EcoV/B	CCBC-strong	KaFFPaEco	KaFFPaStrong	Scotch	Metis	hMetis
loc-brightkite	64	73777	**69984**	85610	70324	80182	74161	72965
loc-gowalla	2	57859	51177	72844	**49219**	60466	56402	57880
loc-gowalla	16	234468	**212197**	341709	214402	230757	231982	230258
loc-gowalla	64	333231	**298688**	406117	300270	336532	332497	327012
PGPgiantcompo	2	344	360	565	366	656	390	**356**
PGPgiantcompo	16	1570	**1478**	1682	1495	2003	1692	1529
PGPgiantcompo	64	2957	**2796**	3097	2837	3222	3092	**2796**

Best results are highlighted by bold font

References

Abou-Rjeili, A., Karypis, G.: Multilevel Algorithms for Partitioning Power-Law Graphs. In: Proceedings of 20th International Parallel and Distributed Processing Symposium (2006)

Bader, D.A., Meyerhenke, H., Sanders, P., Schulz, C., Kappes, A., Wagner, D.: Benchmarking for graph clustering and partitioning. In: Alhajj, R., Rokne, J. (eds.) Encyclopedia of Social Network Analysis and Mining, pp. 73–82. Springer, New York (2014)

Bichot, C., Siarry, P. (eds.): Graph Partitioning. Wiley, Hoboken (2011)

Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Inf. Process. Lett. **42**(3), 153–159 (1992)

Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent advances in graph partitioning. In: Algorithm Engineering—Selected Topics, to app. ArXiv:1311.3144 (2014)

Chevalier, C., Safro, I.: Comparison of coarsening schemes for multilevel graph partitioning. In: Proceedings of the 3rd International Conference on Learning and Intelligent Optimization of LNCS, vol. 5851, pp. 191–205 (2009)

Costa, L.F., Oliveira Jr., O.N., Travieso, G., Rodrigues, F.A., Boas, P.R.V., Antiqueira, L., Viana, M.P., Rocha, L.E.C.: Analyzing and modeling real-world phenomena with complex networks: a survey of applications. Adv. Phys. **60**(3), 329–412 (2011)

Dhillon, I.S., Guan, Y., Kulis, B.: Weighted graph cuts without eigenvectors: a multilevel approach. IEEE Trans. Pattern Anal. Mach. Intell. **29**(11), 1944–1957 (2007)

Gilbert, J., Reinhardt, S., Shah, V.: High-Performance Graph Algorithms from Parallel Sparse Matrices. Springer, Berlin (2006)

Hendrickson, B., Kolda, T.G.: Graph partitioning models for parallel computing. Parallel Comput. **26**(12), 1519–1534 (2000)

Holtgrewe, M., Sanders, P., Schulz, C.: Engineering a scalable high quality graph partitioner. In: Proceedings of the 24th International Parallel and Distributed Processing symposium, pp. 1–12 (2010)

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

Karypis, G., Kumar, V.: Multilevel k-way hypergraph partitioning. In: Proceedings of the 36th ACM/IEEE Design Automation Conference, pp. 343–348. ACM, New York (1999)

Kirmani, S., Raghavan, P.: Scalable parallel graph partitioning. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13), p. 51. ACM, New York (2013)

Kothapalli, K., Pemmaraju, S., Sardeshmukh, V.: On the analysis of a label propagation algorithm for community detection. In: Frey, D., Raynal, M., Sarkar, S., Shyamasundar, R., Sinha, P. (eds.) Distributed Computing and Networking. Lecture Notes in Computer Science, vol. 7730, pp. 255–269. Springer, Berlin (2013)

Laboratory of Web Algorithms, University of Milano. Datasets, http://law.di.unimi.it/datasets.php (2016)

Lancichinetti, A., Fortunato, S., Kertész, J.: Detecting the overlapping and hierarchical community structure in complex networks. New J. Phys. **11**(3), 033015 (2009)

LaSalle, D., Karypis, G.: Multi-threaded modularity based graph clustering using the multilevel paradigm. J. Parallel Distrib. Comput. **76**, 66–80 (2014). doi:10.1016/j.jpdc.2014.09.012. (To appear)

Leskovec, J.: Stanford Network Analysis Package (SNAP). http://snap.stanford.edu/data/index.html (2016)

Meyerhenke, H., Monien, B., Schamberger, S.: Accelerating shape optimizing load balancing for parallel FEM simulations by algebraic multigrid. In: Proceedings of 20th International Parallel and Distributed Processing Symposium (2006)

Meyerhenke, H., Sanders, P., Schulz, C.: Partitioning complex networks via size-constrained clustering. In: Proceedings of 13th International Symposium on Experimental Algorithms (SEA 2014), LNCS, vol. 8504, pp. 351–363. Springer, New York (2014)

Meyerhenke, H., Sanders, P., Schulz, C.: Parallel graph partitioning for complex networks. In: Proceedings of 29th IEEE International Parallel & Distributed Processing symposium (IPDPS 2015), IEEE (2015)

Ovelgönne, M.: Distributed community detection in web-scale networks. In: 2013 International Conference on Advances in Social Networks Analysis and Mining, pp. 66–73 (2013)

Ovelgönne, M., Geyer-Schulz, A.: An ensemble learning strategy for graph clustering. In: Graph Partitioning and Graph Clustering in Contemporary Mathematics, vol. 588. AMS and DIMACS (2013)

Pellegrini, F.: Scotch Home Page. https://www.labri.fr/perso/pelegrin/scotch/ (2012a)

Pellegrini, F.: Scotch and PT-scotch graph partitioning software: an overview. In: Naumann, U., Schenk, O. (eds.) Combinatorial Scientific Computing, pp. 373–406. CRC Press, Boca Raton (2012b)

Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Phys. Rev. E **76**(3), 036106 (2007)
Rotta, R., Noack, A.: Multilevel local search algorithms for modularity clustering. J. Exp. Algorithmics **16**, 2–3 (2011)
Sanders, P., Schulz, C.: KaHIP—Karlsruhe high qualtity partitioning homepage. http://algo2.iti.kit.edu/documents/kahip/index.html (2016)
Sanders, P., Schulz, C.: Engineering multilevel graph partitioning algorithms. In: Proceedings of the 19th European Symposium on Algorithms of LNCS, vol. 6942, pp. 469–480. Springer, New York (2011)
Sanders, P., Schulz, C.: Think locally, act globally: highly balanced graph partitioning. In: Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'12), LNCS. Springer, New York (2013)
Schloegel, K., Karypis, G., Kumar, V.: Graph partitioning for high performance scientific simulations. In: The Sourcebook of Parallel Computing, pp. 491–541 (2003)
Schulz, C.: High Quality Graph Partititioning. PhD thesis, KIT (2013)
Slota, G., Madduri, K., Rajamanickam, S.: Pulp: Scalable multi-objective multi-constraint partitioning for small-world networks. In: IEEE international conference on big data (big data), pp. 481–490 (2014)
Southwell, R.V.: Stress-calculation in frameworks by the method of "systematic relaxation of constraints". Proc. R. Soc. Lond. Ser. A **151**(872), 56–95 (1935)
Staudt, C.L., Meyerhenke, H.: Engineering parallel algorithms for community detection in massive networks. IEEE Trans. Parallel Distrib. Syst. **27**(1), 171–184 (2016)
Ugander, J., Backstrom, L.: Balanced label propagation for partitioning massive graphs. In: 6'th International Conference on Web Search and Data Mining (WSDM'13), pp. 507–516. ACM, New York (2013)
Walshaw, C.: Multilevel refinement for combinatorial optimisation problems. Ann. Oper. Res. **131**(1), 325–372 (2004)
Walshaw, C., Cross, M.: Mesh partitioning: a multilevel balancing and refinement algorithm. SIAM J. Sci. Comput. **22**(1), 63–80 (2000)
Walshaw, C., Cross, M.: JOSTLE: parallel multilevel graph-partitioning software—an overview. Mesh Partit. Tech. Domain Decompos. Tech. pp. 27–58 (2007)