A New Diffusion-based Multilevel Algorithm for Computing Graph Partitions*

Henning Meyerhenke[1] and Burkhard Monien[2] and Thomas Sauerwald[3]

[1] NEC Europe Ltd., NEC Laboratories Europe, Rathausallee 10, 53757 Sankt Augustin, Germany,
meyerhenke@it.neclab.eu

[2] Department of Computer Science, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany,
bm@upb.de

[3] International Computer Science Institute, 1947 Center Street Suite 600, 94704 Berkeley, CA, USA,
sauerwal@icsi.berkeley.edu

Abstract. Graph partitioning requires the division of a graph's vertex set into k equally sized subsets s. t. some objective function is optimized. High-quality partitions are important for many applications, whose objective functions are often $\mathcal{NP}$-hard to optimize. Most state-of-the-art graph partitioning libraries use a variant of the Kernighan-Lin (KL) heuristic within a multilevel framework. While these libraries are very fast, their solutions do not always meet all user requirements. Moreover, due to its sequential nature, KL is not easy to parallelize. Its use as a load balancer in parallel numerical applications therefore requires complicated adaptations. That is why we developed previously an inherently parallel algorithm, called Bubble-FOS/C (Meyerhenke et al., IPDPS'06), which optimizes partition shapes by a diffusive mechanism. However, it is too slow for practical use, despite its high solution quality.

In this paper, besides proving that Bubble-FOS/C converges towards a local optimum of a potential function, we develop a much faster method for the improvement of partitionings. This faster method called TruncCons is based on a different diffusive process, which is restricted to local areas of the graph and also contains a high degree of parallelism. By coupling TruncCons with Bubble-FOS/C in a multilevel framework based on two different hierarchy construction methods, we obtain our new graph partitioning heuristic DibaP. Compared to Bubble-FOS/C, DibaP shows a considerable acceleration, while retaining the positive properties of the slower algorithm. Experiments with popular benchmark graphs show that DibaP computes consistently better results than the state-of-the-art libraries METIS and JOSTLE. Moreover, with our new algorithm, we have improved the best known edge-cut values for a significant number of partitionings of six widely used benchmark graphs.

Keywords: Graph partitioning, load balancing heuristic, disturbed diffusion.

* A preliminary version of this paper appeared at the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'08). Parts of this work have been done while the first and the third author were affiliated with the Department of Computer Science, University of Paderborn.

1 Introduction

Graph partitioning is a widespread technique in computer science, engineering, and related fields. The most common formulation of the graph partitioning problem for an undirected graph $G = (V, E)$ asks for a division of V into k pairwise disjoint subsets (*partitions*) of size at most $\lceil |V|/k \rceil$ such that the *edge-cut*, i.e., the total number of edges having their incident nodes in different subsets, is minimized. Among others, its applications include dynamical systems [8], VLSI circuit layout [11], and image segmentation [35]. We mainly consider its use for balancing the load in numerical simulations (e. g., fluid dynamics), which have become a classical application for parallel computers. There, our task is to compute a partitioning of the (dual) mesh derived from the domain discretization [34].

Despite some successes on approximation algorithms (e. g., [1,20]) for this $\mathcal{NP}$-hard problem, simpler heuristics are preferred in practice. Several different algorithms have been proposed, see Schloegel et al. [34] for an overview. They can be categorized as either global or local optimizers. Spectral methods [14,38] and space-filling curves [46] are representatives of global methods. While space-filling curves work extremely fast, they do not yield satisfying partitionings for complicated domains with holes or fissures. Spectral algorithms have been widely used, but are relatively slow and thus have been mostly superseded by faster local improvement algorithms. Integrated into a multilevel framework, these local optimizers such as Kernighan-Lin (KL) [19] can be found in several state-of-the-art graph partitioning libraries, which we describe in more detail in Section 2.

Motivation. Implementations of multilevel KL yield good solutions in very short time, but the computed partitionings do not necessarily meet the requirements of all users: As Hendrickson has pointed out [13], the number of *boundary vertices* (vertices that have a neighbor in a different partition) models the communication volume between processors in numerical simulations more accurately than the edge-cut. Moreover, the edge-cut is a *summation norm*, while often the *maximum norm* is of much higher importance (e. g., for parallel numerical solvers the worst partition determines the overall application time). Finally, for some applica-

tions, the *shape* of the partitions, in particular small aspect ratios [9], but also connectedness and smooth boundaries, plays a significant role. Nevertheless, most partitioning-based load balancers do not take these facts fully into account.

While the total number of boundary vertices can be minimized by hypergraph partitioning [4], an optimization of partition shapes requires additional techniques (e. g., [9,27]), which are far from being mature. Furthermore, due to their sequential nature, the heuristic KL is difficult to parallelize. Although significant progress has been made [5,33,44], an inherently parallel graph partitioning algorithm for load balancing can be expected to yield better solutions, possibly also in shorter time.

These issues have led us to the development of the partitioning heuristic Bubble-FOS/C in previous work [23] (also see Section 3 of this paper). It is based on a disturbed diffusion scheme that determines how well connected two nodes are in a graph. Well connected refers to the property that nodes or regions are connected to each other by many paths of small length. Using this notion, Bubble-FOS/C aims at the *optimization of partition shapes* and results in partitionings with very often connected partitions that have short boundaries, good edge-cut values and aspect ratios. Moreover, it contains a high degree of natural parallelism and can be used for parallel load balancing, resulting in *low migration costs* [25]. Yet, its partly global approach makes it too slow for practical relevance. It is therefore highly desirable to develop a significantly faster algorithm retaining the good properties of Bubble-FOS/C.

Contribution. The contribution of this paper consists of both theoretical and practical advances in graph partitioning with diffusive shape optimization. In order to understand Bubble-FOS/C better, we prove its convergence in Section 4 as our main theoretical result. The convergence proof relies on a potential function argument and a load symmetry result for the disturbed diffusion scheme FOS/C.

Due to Bubble-FOS/C's high running time, its excellent solution quality could previously not be exploited for large graphs. We present in this work a much faster new diffusive method for the local improvement of partitionings in Section 5. The combination of

Bubble-FOS/C and this new diffusive method within a multilevel framework with two different hierarchy construction algorithms, called DibaP, constitutes our main algorithmic achievement. This combined algorithm is much faster than Bubble-FOS/C and computes multi-way graph partitionings of very high quality on large graphs in a very reasonable amount of time. In Section 6 we show in experiments on well-known benchmark graphs with small average degree that our algorithm delivers better solutions than the state-of-the-art partitioners METIS [17,18] and JOSTLE [43] in terms of edge-cut *and* number of boundary vertices, both in the summation *and* in the maximum norm. Certainly notable is the fact that DibaP also improves for six benchmark graphs a large number (more than 80 out of 144) of their best known partitionings w. r. t. the edge-cut. These six graphs are among the eight largest in a popular benchmark set [37,42].

2 Related Work

In this introductory section we focus on practical state-of-the-art general purpose graph partitioning algorithms and libraries. General purpose means here that these algorithms and libraries only require the adjacency information about the graph and no additional problem-related information. We concentrate on implementations included in the experimental evaluation in Section 6 and on methods with related techniques for improving partitions. For a broader overview the reader is referred to Schloegel et al. [34]. The previous work of the authors on diffusion-based shape-optimizing graph partitioning is described in Section 3.

It should be noted that a number of metaheuristics have been used for graph partitioning recently, e. g., [2,6,37]. These algorithms focus on low edge-cuts instead of good partition shapes and most of them require very high running times to yield high quality results.

2.1 Graph Partitioning by Local Improvement with the Multilevel Paradigm

Refining a given partitioning by local considerations usually yields better running times on large graphs than global approaches. The problem of how to obtain a good starting solution

is overcome by the multilevel approach [3,15], which consists of three phases. Instead of computing a partitioning immediately for large input graphs, one computes a hierarchy of graphs $G_0, \ldots, G_l$ by recursive coarsening in the first phase. G_l is supposed to be very small in size, but similar in structure to the input graph G_0. In the second phase a very good initial solution for G_l is computed, which is easy due to the small size of G_l. Finally, in the third phase, the solution is interpolated to the next-finer graph, where it is improved using a local improvement algorithm. This process of interpolation and local improvement is repeated recursively up to G_0.

A very common local improvement algorithm is based on the method by Fiduccia and Mattheyses (FM) [11], a running time optimized version of the Kernighan-Lin heuristic (KL) [19]. The main idea of both is to migrate nodes between partitions – ordered by the magnitude of the possible cost reductions – while maintaining (almost) balanced partition sizes. After every node has been moved once, the best solution found so far is chosen. This is repeated several times until no further improvements are found.

State-of-the-art graph partitioning libraries such as METIS [17,18] and JOSTLE [43] use KL/FM for local improvement and edge contractions based on matchings for coarsening. With this combination these libraries compute solutions of a good quality very fast. However, as argued in the introduction, for some applications their solutions are not totally satisfactory.

To address the load balancing problem in parallel applications, distributed versions of the above two libraries [33,44] and parallel hypergraph partitioners such as Zoltan [5] or Parkway [40] have been developed. This parallelization is very complex due to inherently sequential parts in KL/FM improvement, e. g., no two neighboring vertices should change their partitions simultaneously.

2.2 Bubble Framework and Shape-optimizing Graph Partitioning

The Bubble framework is related to Lloyd's k-means algorithm [21] (well-known in cluster analysis) and transfers its ideas to graphs. Its first step is to choose initial partition representatives (centers), one for each partition. As illustrated in Figure 1, all remaining vertices

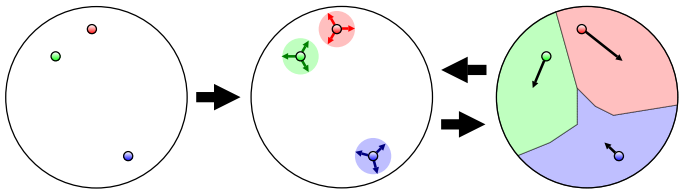


Fig. 1. Sketch of the main BUBBLE framework operations: Determine initial centers for each partition (left), assign each node to the partition of the nearest center (middle), and compute new partition centers (right).

are assigned to their closest center vertex w. r. t. some distance (or similarity) measure. After that, each partition computes its new center for the next iteration. The two operations *assigning vertices to partitions* and *computing new centers* can be repeated alternately a fixed number of times or until a stable state is reached.

For graph partitioning the algorithm has been introduced under the name BUBBLE by Diekmann et al. [9] (which provides references to previous related ideas like Walshaw et al. [45]). To compute new subdomains, Diekmann et al. lets the smallest subdomain with at least one adjacent unassigned vertex grab the vertex with the smallest Euclidean distance to its center. The new center of a partition is computed as the vertex for which the (approximate) sum of Euclidean distances to all other vertices of the same partition is minimal. Thus, by including coordinates in the choice of the next vertex, the subdomains are usually geometrically well-shaped. As a downside, this implementation is only applicable if coordinates are provided. Moreover, the Euclidean distance of two nodes might not coincide with the graph structure at all, leading to unsatisfactory solutions in case of holes or fissures. Also note that a parallelization is not easy due to the strictly serial assignment process. Different implementations of the framework operations exist, see our previous work [23] for how they have evolved.

2.3 Diffusive Approaches to Partitioning

In the area of graph clustering there exist techniques for dividing nodes into groups based on random walks. Their common idea is that a random walk stays a very long time in a dense

graph region before leaving it via one of the few outgoing edges. Somewhat related to our new diffusive method is the algorithm by Harel and Koren [12], which computes separator edges iteratively based on the similarity of their incident nodes. This similarity is derived from the sum of transition probabilities of random walks with very few steps. The procedure focuses on clusters of different sizes and we do not know of any attempt to use it for the graph partitioning problem.

Schamberger [30,31] developed the two diffusive schemes FOS/L and FOS/A. Integrated into the Bubble framework, both schemes shall reflect how well-connected vertices of the graph are to each other. Schamberger's experiments show a promising partitioning quality of his methods. However, he also points out that the practical relevance of his methods is very limited. The major drawbacks are either the dependency on a crucial parameter that is hard to determine or a very high running time.

Recently, Pellegrini [27] has addressed some drawbacks of the KL/FM heuristic. His approach aims at improved partition shapes, based on a diffusive mechanism used together with FM improvement. For the diffusion process the algorithm replaces whole partition regions not close to partition boundaries by one super-node. This reduces the number of diffusive operations and results in an acceptable overall speed. The implementation described is only capable of recursive bisection. As Pellegrini points out, a "full k-way algorithm is therefore required" [27, p. 202] since recursive bisectioning yields in general inferior results compared to direct k-way methods [36], in particular for large k. In this paper we fill this gap by providing a related full k-way partitioning algorithm.

3 Disturbed Diffusion and Bubble-FOS/C

This section describes our own previous work on graph partitioning with diffusive mechanisms. Such a description is necessary to understand the results of this paper. In particular, we explain the partitioning algorithm Bubble-FOS/C, for which we prove two important properties regarding convergence and connectedness in Section 4.

3.1 Disturbed Diffusion FOS/C

Diffusive processes can model transport phenomena such as heat flow. Another application is iterative local load balancing in parallel computations [7]. Diffusion is used here within the BUBBLE framework as a similarity measure that overcomes drawbacks of previous BUBBLE implementations. For this reason a disturbance based on drain [23] has been introduced into the first order diffusion scheme (FOS) [7] for load balancing to yield the FOS/C algorithm (C for constant drain). FOS/C reaches a non-balanced load distribution in the steady state, which can represent similarities of graph nodes reflecting their connectedness.

Definition 1. *[23] Let $[x]_v$ denote the component of the vector x corresponding to node v. Let $G = (V, E)$ be a connected and undirected graph free of self-loops with n nodes and m edges. Associated to G are a set of source nodes $\emptyset \neq S \subset V$ and constants $0 < \alpha \leq (\mathrm{maxdeg}(G)+1)^{-1}$ and $\delta > 0$.[4] Let the initial load vector $w^{(0)}$ and the drain vector d (which is responsible for the disturbance) be defined as follows:*

$$[w^{(0)}]_v = \begin{cases} \frac{n}{|S|} & v \in S \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad [d]_v = \begin{cases} \frac{\delta n}{|S|} - \delta & v \in S \\ -\delta & \text{otherwise} \end{cases}$$

Then, the FOS/C iteration in timestep $t \geq 1$ is defined as $w^{(t)} = \mathbf{M} w^{(t-1)} + d$, where $\mathbf{M} = \mathbf{I} - \alpha\mathbf{L}$ is the doubly-stochastic diffusion matrix [7] and $\mathbf{L}$ the Laplacian matrix of G, defined as $\mathbf{L}_{u,u} = \deg(u)$, $\mathbf{L}_{u,v} = -1$ for $\{u, v\} \in E$ and $\mathbf{L}_{u,v} = 0$ otherwise.

Note that the extension of FOS/C to edge-weighted graphs (with edge weight function $\omega : E \to \mathbb{R}$) is straightforward. One simply uses the edge-weighted variant of the Laplacian matrix, i. e., $\mathbf{L}_{u,v} = -\omega(u, v)$ for $\{u, v\} \in E$, $\mathbf{L}_{u,u} = \sum_{v \neq u} \mathbf{L}_{u,v}$, and $\mathbf{L}_{u,v} = 0$ otherwise. Node weights can be incorporated by a proportional weighting of the drain vector entries.

[4] Here, the maximum degree of G is defined as $\mathrm{maxdeg}(G) := \max_{u \in V} \deg(u)$.

Theorem 2. *[23] The FOS/C iteration reaches a steady state for any $d \perp (1, \ldots, 1)^T$. This steady state can be computed by solving the linear system $\mathbf{L}w = d$ for w and normalizing w such that $\sum_{v \in V} [w]_v = n$.*

Definition 3. *If $|S| = 1$ ($|S| > 1$), we call the FOS/C iteration to the steady state or its corresponding linear system a* single-source (multiple-source) FOS/C procedure. *Also, let $[w^{(t)}]_v^u$ ($[w]_v^u$) denote the load on node v in timestep t (in the steady state) of a single-source FOS/C procedure with node u as source.*

Definition 4. *A* random walk *on a graph $G = (V, E)$ is a discrete time stochastic process, which starts on an initial node and performs the following step in each iteration. It chooses one of the neighbors of the current node randomly and then proceeds to the neighbor just chosen to start the next iteration. The transition probabilities are given by a stochastic transition matrix $\mathbf{P}$, whose entry (u, v) denotes the probability to move from node u to node v. The random walk may stay on the current node v with positive probability if $\mathbf{P}_{vv} > 0$.*

Remark 5. Note: $[w]_v^u = \lim_{t \to \infty}([\mathbf{M}^t w^{(0)}]_v^u + n\delta(\sum_{l=0}^{t-1} \mathbf{M}_{v,u}^l) - t\delta)$ [24], where $\mathbf{M}_{v,u}^l$ is the probability of a random walk (defined by the stochastic diffusion matrix $\mathbf{M}$) starting at v to be on u after l steps. Since $[\mathbf{M}^t w^{(0)}]_v^u$ converges towards the balanced load distribution [7], the important part of a FOS/C load in the steady state is $\sum_{l=0}^{\infty} \mathbf{M}_{v,u}^l$, which is the sum of transition probabilities of random walks with increasing lengths.

3.2 Bubble-FOS/C with Algebraic Multigrid

Bubble-FOS/C implements the operations of the Bubble framework with FOS/C procedures, single-source ones for `AssignPartition` and multiple-source ones for `ComputeCenters`. Its outline is shown in Figure 2, where $\Pi = \{\pi_1, \ldots, \pi_k\}$ denotes the set of partitions and $Z = \{z_1, \ldots, z_k\}$ the set of the corresponding center nodes. First, the algorithm determines pairwise disjoint initial centers (line 1), which can be done in an arbitrary manner. After that, with the new centers at hand, the main loop is executed. It determines in alternating calls a new partitioning (`AssignPartition`, lines 3-6) and new

Algorithm Bubble-FOS/C$(G, k) \rightarrow \Pi$

01 $Z = \text{InitialCenters}(G, k)$ /* Arbitrary disjoint initial centers */
02 **for** $\tau = 1, 2, \ldots$ **until** convergence
  /* `AssignPartition` */
03 **parallel for** each partition π_c
04 Initialize d_c $(S = \{z_c\})$, solve and normalize $\mathbf{L}w_c = d_c$
  /* after synchronization: update Π */
05 **for** each node $v \in \pi_c$
06 $\Pi(v) = p : [w_p]_v \geq [w_q]_v \, \forall q \in \{1, \ldots, k\}$
  /* `ComputeCenters` */
07 **parallel for** each partition π_c
08 Initialize d_c $(S = \pi_c)$ and solve $\mathbf{L}w_c = d_c$
09 $z_c = \text{argmax}_{v \in \pi_c}[w_c]_v$
10 **return** Π

Fig. 2. Sketch of the main Bubble-FOS/C algorithm.

centers (`ComputeCenters`, lines 7-9). The loop can be iterated until convergence is reached or, if running time is important, a constant number of times.

It turns out that this iteration of two alternating operations yields very good partitions. The ability of distinguishing sparsely from densely connected components can be explained by FOS/C's connection to random walks pointed out above. As random walks tend to stay a long time within a dense region once they have reached it, Bubble-FOS/C usually obtains partition centers in dense regions and boundaries tend to be in sparse ones (as desired). Moreover, since the isolines of the FOS/C load in the steady state tend to have a circular shape, the final partitions are very compact and have short boundaries. Additional operations not originating from the Bubble framework (such as balancing procedures and tie breaking[5]) can be integrated into Bubble-FOS/C [25], but are omitted here for ease of presentation.

[5] Ties within `AssignPartition` and `ComputeCenters` are handled as follows. If a node has received the same highest load from more than one FOS/C procedure within `AssignPartition`, it chooses the subdomain it already belongs to, or – if the current subdomain is not among the candidates – the one with the smallest index. In case more than one node is a candidate for the new center within `ComputeCenters`, we proceed analogously.

Most work performed by Bubble-FOS/C consists in solving linear systems. It is therefore necessary to employ a very efficient solver. Multigrid methods [41] are among the fastest algorithms for preconditioning and solving linear systems of equations arising from certain partial differential equations. Algebraic multigrid (AMG) [39] is an extension to cases where no problem-related information such as geometry is available. It constructs a multilevel hierarchy based on weighted interpolation with a carefully chosen set of nodes for the coarser level. The actual solution process is performed by iterative algorithms traversing this hierarchy, e. g., V-cycles or FMV-cycles [41, p. 46ff.]. We use AMG as a linear solver since the same system matrix $\mathbf{L}$ is used repeatedly, so that the hierarchy construction is amortized. Furthermore, as an AMG hierarchy is also a sequence of coarser graphs retaining the structure of the original one, we use it in our Bubble-FOS/C implementation for providing a multilevel hierarchy (instead of the standard matching approach). For Bubble-FOS/C this alternative hierarchy construction method hardly influences the solution quality, but speeds up computations significantly [22, p. 79].

4 Convergence and Connectedness Results for Bubble-FOS/C

4.1 Convergence towards a Local Optimum

In this section we settle the question if the algorithm Bubble-FOS/C depicted in Figure 2 converges, in the affirmative. The proof relies on load symmetry and a potential function, which provide a solid characterization of our algorithm and the solutions it computes.

Definition 6. *Let the function $F(\Pi, Z, \tau)$ for timestep τ be defined as follows:*

$$F(\Pi, Z, \tau) := \sum_{c=1}^{k} \sum_{v \in \pi_c(\tau)} [w]_v^{z_c(\tau)},$$

where $\pi_c(\tau)$ and $z_c(\tau)$ denote the c-th partition and center node in iteration τ, respectively.

Observe that $[w]_v^{z_c(\tau)}$ acts as a similarity measure, i.e., it represents how well-connected the vertices $v \in \pi_c(\tau)$ and their respective center $z_c(\tau)$ are. Hence, each partition contributes to F a sum of similarities between all nodes of a partition and its center node.

Our objective is now to maximize F. It is obvious that F has a finite upper bound on any finite graph. Thus, in order to prove convergence of BUBBLE-FOS/C, it is sufficient to show that the operations `AssignPartition` and `ComputeCenters` each maximize the value of F w.r.t. their input. This is clear for `AssignPartition` since nodes are assigned to partitions sending the highest amount of load. Yet, it is not obvious for `ComputeCenters`, so that we require first the following result on the load symmetry between two single-source FOS/C procedures.

Lemma 7. *For any undirected graph $G = (V, E)$ and two arbitrary nodes $u, v \in V$ holds $[w]_v^u = [w]_u^v$.*

Proof. Consider an FOS/C procedure with source node u. Recall that its drain vector d is defined as $d = (-\delta, \ldots, -\delta, \delta(n-1), -\delta, \ldots, -\delta)^T$, where $\delta(n-1)$ appears in row u. The FOS/C iteration scheme in timestep $t+1$ for node v and source u can be written as [24]:

$$\begin{aligned}[w^{(t+1)}]_v^u &= [\mathbf{M}^{t+1} w^{(0)}]_v^u + [(\mathbf{I} + \mathbf{M}^1 + \cdots + \mathbf{M}^t) d]_v^u \\ &= [\mathbf{M}^{t+1} w^{(0)}]_v^u + n\delta \sum\nolimits_{l=0}^{t} \mathbf{M}_{v,u}^l - (t+1)\delta.\end{aligned}$$

Observe that $\mathbf{M}^{t+1} w^{(0)}$ converges towards $\overline{w} = (1, \ldots, 1)^T$, the balanced load distribution [7], even in the edge-weighted case [10]. Hence, we obtain:

$$[w]_u^v - [w]_v^u = n\delta \left(\sum\nolimits_{l=0}^{t} \mathbf{M}_{u,v}^l - \mathbf{M}_{v,u}^l \right).$$

Since $\mathbf{M}$ and therefore also its powers are symmetric, all summands vanish. □

The generality of this load symmetry is somewhat surprising, because one would not expect such a property in graphs without any symmetry. Consider for example a lollipop graph L,

i. e., a clique with $n/2$ vertices attached to a path of $n/2$ vertices. Let u be any vertex in the clique and v be the end-point of this path. Although L is far from being symmetric, Lemma 7 implies that the load of v of a FOS/C procedure with single source u is the same as the load of u after a FOS/C procedure with single source v. In that regard Lemma 7 seems to be of independent interest for the disturbed diffusion scheme FOS/C. Here it allows us to prove the next crucial lemma.

Lemma 8. *The output of* `ComputeCenters` *maximizes the value of F for a given k-partitioning Π.*

Proof. Let Π be the current partitioning. `ComputeCenters` solves for each partition $\pi_c, c \in \{1, \ldots, k\}$, a multiple-source FOS/C procedure, where the whole respective partition acts as source. Consider one of these partitions π_c and its multiple-source FOS/C procedure, which computes w in $\mathbf{L}w = d$ with its respective drain vector d. Our aim is to split this procedure into subprocedures that solve $\mathbf{L}w_i = d_i, i \in \pi_c$, and that satisfy $\sum_{i \in \pi_c} d_i = d$. Such a splitting $\mathbf{L}(w_1 + w_2 + \cdots + w_{|\pi_c|}) = d_1 + d_2 + \cdots + d_{|\pi_c|}$ indeed exists; each subprocedure $\mathbf{L}w_i = d_i$ corresponds to a single-source procedure, where the drain vector is scaled by $\frac{1}{|\pi_c|}$ (cf. Definition 1 with $|S|$ replaced by 1):

$$[d_i]_v = \begin{cases} \frac{\delta n}{|\pi_c|} - \frac{\delta}{|\pi_c|} & v \in \pi_c,\ v \text{ source of subprocedure } i \\ -\frac{\delta}{|\pi_c|} & \text{otherwise} \end{cases}$$

It is easy to verify that $\sum_{i \in \pi_c} d_i = d$ and $d_i \perp (1, \ldots, 1)^T$ hold, so that each subprocedure has a solution. Due to this and the linearity of $\mathbf{L}$, we also have $\sum_{i \in \pi_c} w_i = w$. Recall that the new center of partition π_c is the node with the highest load of the considered multiple-source FOS/C procedure. From the above it follows that this is the node u for which $[w]_u = \sum_{i \in \pi_c} [w]_u^i$ is maximal. Due to Lemma 7 we have $\sum_{i \in \pi_c} [w]_u^i = \sum_{i \in \pi_c} [w]_i^u$, so that the new center z_c is the node u for which the most load remains within partition π_c in

a single-source FOS/C procedure. Consequently, the contribution $\sum_{v \in \pi_c} [w]_v^{z_c}$ of each π_c to F is maximized. □

Proposition 9. *Consider the load vector w in the steady state of FOS/C. The maximum load value in w belongs to the set of source nodes S. Consequently, after selecting k pairwise disjoint initial center nodes, there are always exactly k different center nodes and exactly k partitions during the execution of* Bubble-FOS/C.

Proof. The first statement can be verified by taking into account that the steady state of FOS/C is equivalent to a $\|\cdot\|_2$-minimal-flow problem where the source nodes send load to the remaining nodes (see [24, p. 431]) and must therefore have a higher load. For the second statement "$\leq$" is obvious, so that it remains to show that there are at least k different center nodes and partitions in each iteration.

The initial placement of centers can ensure easily that k different nodes are selected. In any case the centers determined by `ComputeCenters` belong to their own partition and must therefore be different. Also, `AssignPartition` keeps each center in its current partition: Consider two arbitrary, but distinct centers z_i and z_j. Due to Lemma 7 we know that $[w]_{z_i}^{z_j} = [w]_{z_j}^{z_i}$. As $[w]_{z_i}^{z_i} > [w]_{z_j}^{z_i}$, we obtain $[w]_{z_i}^{z_i} > [w]_{z_i}^{z_j}$. Hence, the claim follows. □

The main theorem follows now directly from the results above.

Theorem 10. Bubble-FOS/C *converges and produces a k-way partitioning. This partitioning is a local optimum of the potential function F.*

If Bubble-FOS/C is used for partitioning within a multilevel hierarchy, it converges very quickly. Our experiments indicate that usually three to five iterations of the main loop are sufficient to reach convergence on each level.

Maximizing F has the following connection to the minimization of the traditional edge-cut metric. Recall from Remark 5 that each entry $[w]_u^s$ of the load vector $[w]^s$ can be interpreted as the sum over the l-step transition probabilities from source s to node u, where l ranges from 0 to infinity. Moreover, observe again that random walks stay in densely connected regions

for a long time before they leave them via one of the few external edges. This observation indicates why in most cases higher values occur in $[w]^s$ for nodes within the same dense region as s than for nodes of different regions. The process of locally maximizing F leads to the identification of such dense regions and thus implicitly yields partitions with few external edges, i. e., with a low edge-cut. However, as no explicit edge-cut minimization takes place, additional iterations of Bubble-FOS/C do not necessarily improve the edge-cut every time.

5 Accelerating Bubble-based Diffusive Partitioning

Our previous work on shape-optimizing graph partitioning [23,25] has already indicated that shape optimization is able to compute high-quality partitionings meeting the requirements mentioned in the introduction. The main reason for Bubble-FOS/C's very high running time is the repeated solution of linear systems on the whole graph (or at least on an approximation of the whole graph, as in [26]). Yet, once a reasonably good solution has been found, alterations during an improvement step take place mostly at the partition boundaries. That is why we introduce in the following a local approach considering only these boundary regions. Our idea is to use the high-quality but slow algorithm Bubble-FOS/C on the coarse levels of a multilevel hierarchy and a faster local scheme on its finer levels.

5.1 A New Local Improvement Method: TruncCons

As a mixture of `AssignPartition` and `ComputeCenters`, the `Consolidation` operation is used to determine a new partitioning from a given one. The operation is illustrated in Figure 3 (a) with an example of a path graph, its partitioning and $k = 3$ (topmost row). Note that different colors indicate different partition assignments and that in the second and third row of the figure the input graph is shown k times to illustrate that the corresponding operations are performed independently on each partition π_c. One starts with the initialization of the source set S with π_c. The nodes of π_c receive an equal amount of initial load $n/|S|$, while the other nodes' initial load is set to 0 (second row). Then, a diffusive method (FOS/C

is possible, but should be avoided for large graphs due to its high running time) is used to distribute this load within the graph (third row). To restrict the computational effort to areas close to the partition boundaries, we use a small number ψ of FOS [7] iterations.

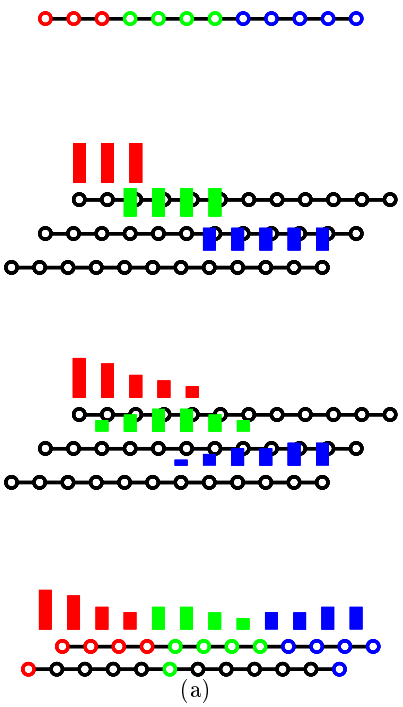
The final load of a node v for π_c is then just $[w_c^{(\psi)}]_v = [\mathbf{M}^\psi \cdot w_c^{(0)}]_v$, where $\mathbf{M}$ and $w^{(0)}$ are as in Definition 1. This can be computed by iterative load exchanges:

$$[w_c^{(t)}]_v = [w_c^{(t-1)}]_v - \alpha \sum_{\{u,v\} \in E} ([w_c^{(t-1)}]_v - [w_c^{(t-1)}]_u) \text{ for } 1 \leq t \leq \psi.$$

After the load is distributed this way for all k partitions, we assign each node v to the partition from which it has obtained the highest load (bottommost row of Figure 3 (a)). This completes one `Consolidation` operation, which can be repeated several times to facilitate sufficiently large movements of the partitions. We denote the number of repetitions by Λ and call the whole method with this particular diffusive process TruncCons (*truncated diffusion consolidations*), see Figure 3 (b). This new approach makes Schamberger's idea to use a diffusive scheme within a Bubble related framework [30] robust, practicable, and fast. Moreover, although showing some differences, our new algorithm can be viewed as a k-way extension of Pellegrini's work [27] mentioned in Section 2.3.

To understand why TruncCons works well, consider the following analogy. Recall that the stochastic diffusion matrix $\mathbf{M}$ can be seen as the transition matrix of a random walk. For each node $v \in \pi_c$ we have one random walk starting on v. Then, the final load on node u is proportional to the sum of the probabilities for each random walk to reach u after ψ steps. Since random walks need relatively long to leave dense regions, each node should be assigned to the partition with the highest load because it is most densely connected with it. Another important observation is that nodes of the same dense region are connected to each other by many paths of small length. This notion of connectivity is reflected by transition probabilities of random walks with small lengths.

Since an actual load exchange happens only at the partition boundary, not all nodes have to take part in this process. Instead, one keeps track of *active nodes*.



(a)

Algorithm TRUNCCONS($\mathbf{M}$, k, Π, Λ, ψ) $\rightarrow \Pi$

```
01  for τ = 1 to Λ
02    parallel for each partition π_c
03      S = π_c; w_c = (0,...,0)^T /* initial load */
04      for each v ∈ S /* initial load */
05        [w_c]_v = n/|S|
06      for t = 1 to ψ /* FOS iterations */
07        w_c = M · w_c
        /* after synchronization: update Π */
08      for each v ∈ π_c
09        Π(v) = argmax_{1≤c'≤k} [w_c']_v
10  return Π
```

(b)

Fig. 3. (a) Schematic view of one `Consolidation` and (b) Algorithmic sketch of TRUNCCONS

Definition 11. *A node $v \in V$ is called* active *in FOS iteration $t > 0$ if it has a neighbor $u \in V$ with the property: $[w^{(t-1)}]_u \neq [w^{(t-1)}]_v$. Nodes that are not active are called* inactive.

All load exchanges of an inactive node result in a flow of 0 on its incident edges. Hence, they do not change the load situation at all and can be ignored. Clearly, in iteration t all nodes with distance more than t to the boundary of the current partition are inactive. This observation might not give away all inactive nodes, but one can expect that exceptions to this rule are rare. So, by keeping track of active and inactive nodes using the above observation, we are able to ignore nearly all load exchange computations that do not change the respective loads on the incident nodes. In this way, the diffusive process of partition improvement is restricted to local areas close to the subdomain boundaries and its complexity is greatly reduced in practice. The effectiveness of the reduction depends on several factors such as the iteration number t, the number of subdomains k, and graph properties such as size, sparseness, and general structure.

5.2 The New Algorithm DibaP: Combining Bubble-FOS/C and TruncCons

Now that we have a slow, but high-quality partitioner and a faster local improvement algorithm, we combine them to obtain an efficient multilevel graph partitioning algorithm that we call DibaP (*Diffusion-based Partitioning*), see Figure 4. The fine levels of its multilevel hierarchy are constructed by approximate maximum weight matchings [28]. Once the graphs are sufficiently small, we switch the construction mechanism to the more expensive AMG coarsening. This is advantageous because we use Bubble-FOS/C as the improvement strategy on the coarse levels and employ AMG to solve the occurring linear systems. That is why such a hierarchy needs to be built anyway. On the finer parts of the hierarchy, the faster TruncCons is used as the local improvement algorithm. Since this does not involve the solution of linear systems, AMG is not required, so that it is much cheaper to use a matching hierarchy instead.

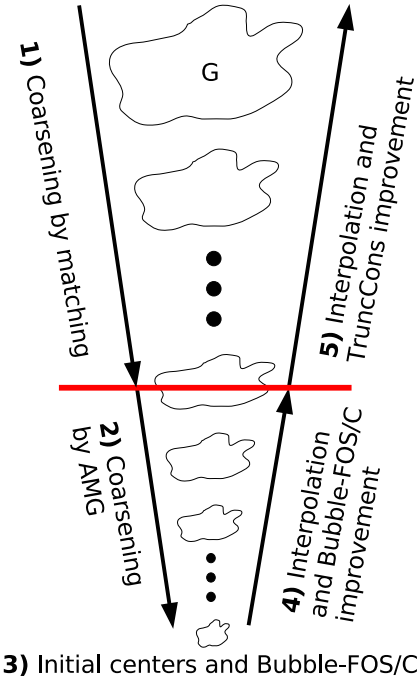


Fig. 4. Sketch of the combined multilevel hierarchy and the corresponding algorithms used within DibaP.

It is highly doubtful that a multilevel approach solely based on TruncCons can be adapted to partition graphs from scratch with an equally high quality as Bubble-FOS/C and DibaP. Our experiments clearly indicate that the partition shapes and other important properties of the solutions (such as edge-cut and the number of boundary vertices) suffer in quality if TruncCons is used on too coarse levels or even exclusively. The quality also declines if Bubble-FOS/C is replaced by kMETIS for providing initial solutions for TruncCons. One reason is that TruncCons's starting solution should have compact and connected partitions with low diameter. If the initial solution does not fulfill these requirements, elongated or disconnected partitions with an inferior solution quality occur much

more often. Furthermore, replacing Bubble-FOS/C on the coarse levels by some faster method yields only very small performance increases. The reason for this is simply that on large enough graphs (such as the benchmark graphs in Section 6) most of DibaP's running time is spent on the fine levels.

Initial Centers and Initial Bubble-FOS/C Solutions. Instead of selecting all initial center vertices randomly or to coarsen the graph until the number of nodes is k, we employ the following procedure to distribute the centers while taking the graph structure into account. After choosing only one center randomly, we select new centers one after another, where the newest one is chosen farthest away (i. e., with minimum FOS/C loads: $\text{argmin}_v\{\sum_{z\in Z}[w]_v^z\}$) from all already chosen centers in the set Z. On a very coarse graph of a multilevel hierarchy, this is inexpensive and can even be repeated to choose the best set of centers from a sample. By this repetition, outliers with a rather poor solution quality hardly occur in our experiments.

The same idea of multiple initial solutions is pursued on a finer level as well. Before starting multilevel partitioning with TruncCons, we call Bubble-FOS/C a number of times and keep only the best of the solutions. Since the graph on the coarsest TruncCons level (the finest Bubble-FOS/C level) is relatively small, Bubble-FOS/C returns a solution quite fast. Besides a higher average quality our experiments reveal also a lower variance (and hence a higher reliability) in the solution quality.

Repartitioning. For cases where a partitioning is part of the input and needs to be repartitioned (e. g., to restore its balance), we propose the following procedure. The initial partitioning is sent down the multilevel hierarchy, where the maximum hierarchy depth depends on the input quality. If the input is not too bad, Bubble-FOS/C does not need to be used and we can solely employ the faster TruncCons with a matching hierarchy. Experiments show that repartitioning this way is about three times faster than partitioning from scratch. On the other hand, if the quality of the input requires larger movements of the partitions, Bubble-FOS/C is necessary since TruncCons does not generate such large movements.

Implementation Details. The `Consolidation` operation can also be used with FOS/C in lines 6 and 7 instead of a few FOS iterations. This again global operation can be optionally integrated into Bubble-FOS/C after an `AssignPartition` operation.

To ensure that the balance constraints are definitely met by DibaP, explicit balancing procedures are integrated into the improvement process. They are mostly based on our previous implementation [25], but are slightly adapted to TruncCons. This also holds for the smoothing operation, which improves partitions by moving current boundary vertices once if this results in fewer cut-edges. Keeping track of active nodes is currently done with an array in which we store for each node its status. This could be improved by a faster data structure considering only the active nodes.

There are several important parameters controlling the quality and running time of DibaP; their values have been determined experimentally. Multilevel hierarchy levels with graphs of more than `switch` nodes are coarsened by matchings and improved with TruncCons (`switch` is a user-definable parameter). Once they are smaller than this threshold, we switch to Bubble-FOS/C with AMG coarsening. Details about the implementation of Bubble-FOS/C and the AMG implementation (which does not use any external libraries) can be found in our previous work [23]. It should be noted, however, that we have made some changes to our AMG implementation. A detailed description of these modifications is outside the scope of this paper. One change to mention is the choice of the interpolation scheme, which is now *classical interpolation* as in Safro et al. [29].

In the experiments presented in this paper, Bubble-FOS/C has performed two iterations of `ComputeCenters` and `AssignPartition`, followed by two `Consolidations` with FOS/C as similarity measure. The AMG coarsening is stopped when the graph has at most $24k$ nodes to compute an initial set of centers. The most important parameters for the finer parts of the multilevel hierarchy are Λ (the number of `Consolidations`) and ψ (the number of FOS iterations). As most other parameters, they can be specified by the user, whose choice should consider the time-quality trade-off. Two possible choices (6/9 and 10/14) are

used in the experiments presented next. Experimental observations suggest that values larger than 20 for Λ and ψ hardly yield any further improvements. Whether an automatic choice depending on graph properties can be made, remains an object of further investigation.

Computational Complexity. For sufficiently large graphs it is clear that the running time of DibaP is dominated by that of TruncCons. The reason is simply that the size of the hierarchy level on which the algorithm switch takes place can be fixed with a constant. That is why the Bubble-FOS/C part of DibaP requires nearly always a very similar amount of time for the same amount of subdomains, regardless of the input graph size.

Within TruncCons one performs for each subdomain Λ times ψ FOS iterations. In the (unrealistic) worst case, for each edge of the graph a load exchange takes place in every iteration. Hence, in this case the running time is proportional to $k \cdot \Lambda \cdot \psi \cdot |E|$. The affiliation of nodes to subdomains requires $\mathcal{O}(n \cdot k)$ operations. If Λ and ψ are seen as constants, the asymptotic running time is bounded by $\mathcal{O}(k \cdot |E|)$. The linear dependence on the factor k – instead of an additive penalty for increasing the number of subdomains – can be seen as the major drawback in the running time of TruncCons compared to optimized KL/FM partitioners.

Furthermore, the product $\Lambda \cdot \psi$ might be quite large, depending on the user choice. On the other hand, due to the notion of (in)active nodes, the number of operations actually performed will be much smaller in practice. Since the savings depend on many factors that differ from input to input, a theoretical worst-case analysis is not likely to predict the final running time accurately.

6 Experimental Results

In this section we present some of our experiments with the new DibaP implementation. After comparing it to METIS and JOSTLE, two state-of-the-art partitioning tools, we show that DibaP performs extremely well on six popular benchmark graphs. For these graphs,

DibaP has computed a large number of partitionings with the best known edge-cut values, improving records derived from numerous algorithms.

Settings. The experiments have been conducted on a desktop computer equipped with an Intel Core 2 Duo 6600 CPU and 1 GB RAM. The operating system is Linux (openSUSE 10.2, Kernel 2.6.18) and the main code has been compiled with Intel C/C++ compiler 10.0 using level 2 optimization. We distinguish DibaP-short ($\Lambda = 6$, $\psi = 9$) and DibaP-long ($\Lambda = 10$, $\psi = 14$) to determine how the quality is affected by different settings in the new method. Bubble-FOS/C is used to compute three coarse solutions on the first level with less than `switch`=5,000 nodes. The best solution w. r. t. the edge-cut is used as input for the multilevel improvement process with TruncCons.

6.1 Comparison of Partitioning Quality

	Size		Degree			
Graph	$\|V\|$	$\|E\|$	min	max	avg	Origin
tooth	78,136	452,591	3	39	11.59	FEM 3D
rotor	99,617	662,431	5	125	13.30	FEM 3D
598a	110,971	741,934	5	26	13.37	FEM 3D
ocean	143,437	409,593	1	6	5.71	FEM 3D
144	144,649	1,074,393	4	26	14.86	FEM 3D
wave	156,317	1,059,331	3	44	13.55	FEM 3D
m14b	214,765	1,679,018	4	40	15.64	FEM 3D
auto	448,695	3,314,611	4	37	14.77	FEM 3D

Fig. 5. Benchmark graphs.

For the further presentation we utilize the eight widely used benchmark graphs shown in Figure 5. We have chosen them because they are publicly available from Chris Walshaw's well-known graph partitioning archive [37,42] and are the eight largest therein w. r. t. the number of nodes. More importantly, they represent the general trends in our experiments for graphs with low average degree (< 20) and constitute a good sample since they model large enough problems from 3-dimensional numerical simulations (e. g., according to [16], *598a* and *m14b* are meshes of submarines and *auto* of a GM Saturn). Graphs with larger average degrees such as *bcsstk32* (avg. degree 44.16) (also available from the same archive) are excluded since DibaP's implementation needs more tuning to handle graphs with larger average degrees equally well.

We compare our algorithm DIBAP to JOSTLE [43] and METIS (more precisely KMETIS[6] [18], which implements direct k-way KL/FM improvement) because these two are the most popular sequential general purpose graph partitioners due to their speed and adequate quality. Detailed comparisons to the library SCOTCH are not included since both the average quality and the running time of SCOTCH are consistently worse than KMETIS's. JOSTLE and KMETIS are used with default settings so that their optimization objective is the edge-cut. We allow all four programs to generate partitionings with at most 3% imbalance, i. e., whose largest partition is at most 3% larger than the average partition size. To specify this is important because a higher imbalance can result in better partitionings.

The order in which the vertices of the graph are stored has a great impact on the partitioning quality of KL/FM partitioners because it affects the order of the node exchanges. Hence, METIS and JOSTLE are run ten times on the same graph, but with a randomly permuted vertex set. For DIBAP the order of the vertices is insignificant. This is because the diffusive partitioning operations are only affected by it in the rare case of ties in the load values. That is why we perform ten runs on the same graph with different random seeds, resulting in different choices for the first center vertex.

Evaluation Methodology. How to measure the quality of a partitioning, depends mostly on the application. Besides the edge-cut, we also include the number of boundary nodes since this measures communication costs in parallel numerical simulations more accurately [13]. To assess the partition shapes, we also include results on the partition diameter. The measures for a partition p are defined as:

$$ext(p) := |\{e = \{u, v\} \in E : \Pi(u) = p \wedge \Pi(v) \neq p\}| \quad \text{(external edges or cut-edges)},$$

$$bnd(p) := |\{v \in V : \Pi(v) = p \wedge \exists \{u, v\} \in E : \Pi(u) \neq p\}| \quad \text{(boundary nodes)},$$

$$diam(p) := \max \{\text{dist}(u, v) : \Pi(u) = \Pi(v) = p\} \quad \text{(diameter)}.$$

[6] The variant of METIS which yields shorter boundaries than KMETIS is not chosen because its results are still worse than those of DIBAP regarding boundary length and they show much higher edge-cut values than KMETIS.

Table 1. Average edge-cut (EC) and number of boundary nodes (BN) for ten randomized runs on the eight benchmark graphs in the *summation norm* (EC, the edge-cut, denotes half the ℓ_1-norm value); the values for JOSTLE, DIBAP-short, and DIBAP-long are relative to the respective value obtained by KMETIS.

	KMETIS		JOSTLE		DIBAP-short		DIBAP-long	
k	EC	BN	EC (rel.)	BN (rel.)	EC (rel.)	BN (rel.)	EC (rel.)	BN (rel.)
4	13836.9	7486.9	1.014	1.025	0.971	0.937	**0.926**	**0.928**
8	24079.0	13032.8	1.021	1.030	**0.945**	0.960	0.952	**0.938**
12	32605.2	17486.9	0.987	0.996	**0.910**	0.937	0.931	**0.919**
16	39013.7	20895.1	0.977	0.988	0.927	0.934	**0.924**	**0.914**
20	44952.6	23953.8	0.980	0.991	**0.923**	0.938	0.931	**0.917**
32	58275.7	30858.0	0.971	0.981	**0.909**	0.939	0.939	**0.918**
64	82292.8	42766.7	0.971	0.978	0.980	0.940	**0.948**	**0.919**
avg (rel)	1.0	1.0	0.989	0.998	0.938	0.941	**0.936**	**0.922**

Note that the edge-cut is the summation norm of the external edges divided by 2 to account for counting each edge twice. For some applications not only the summation norm ℓ_1 of *ext* and *bnd* over all k partitions has to be considered, but also the maximum norm ℓ_∞. This is particularly the case for parallel simulations, where all processors have to wait for the one computing longest. That is why we record *ext* and *bnd* in both norms.

For a succinct presentation Tables 1 (ℓ_1-norm) and 2 (ℓ_∞-norm) show the results in a very condensed form. For all values obtained for a graph (external edges and boundary nodes in both norms) we use the results of KMETIS as standard of reference to simplify the evaluation. This means that each value of the other partitioners is divided by the respective value of KMETIS. Then, an average value of these ratios over all ten runs and all graphs is computed and displayed in the two tables. Larger values of k are not included since such large partitionings are typically computed by parallel partitioners. If the latter are based on KL techniques (as PARMETIS and parallel JOSTLE), they tend to have a worse solution quality than their sequential counterparts, while DIBAP's solution quality is not affected by parallel execution.

Results. Table 1 shows that, in the summation norm, DIBAP-short improves on KMETIS in all cases and on JOSTLE in all cases but one (EC for $k = 64$). It is re-

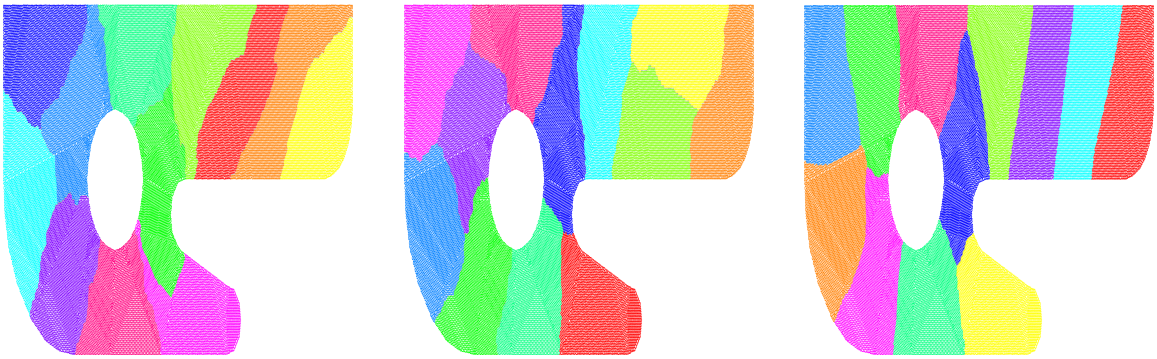
Table 2. Average number of external edges (EE) and of boundary nodes (BN) for ten randomized runs on the eight benchmark graphs in the *maximum norm*; the values for JOSTLE, DibaP-short, and DibaP-long are relative to the respective value obtained by kMETIS.

	kMETIS		JOSTLE		DibaP-short		DibaP-long	
k	EE	BN	EE (rel.)	BN (rel.)	EE (rel.)	BN (rel.)	EE (rel.)	BN (rel.)
4	8793.4	2381.0	1.027	1.034	0.968	**0.937**	**0.949**	0.948
8	8220.0	2238.1	1.096	1.097	0.950	0.971	**0.934**	**0.922**
12	7662.6	2070.2	1.006	1.008	0.910	0.943	**0.909**	**0.902**
16	6826.2	1824.4	1.037	1.040	0.931	0.940	**0.907**	**0.895**
20	6457.0	1718.1	1.056	1.051	0.929	0.949	**0.918**	**0.903**
32	5436.3	1425.9	1.085	1.075	**0.924**	0.954	0.953	**0.926**
64	**3835.4**	968.1	1.096	1.079	1.002	**0.957**	1.018	0.990
avg (rel)	1.0	1.0	1.058	1.055	0.945	0.950	**0.941**	**0.927**

markable that DibaP-short even computes better edge-cut values than DibaP-long in four cases. The computationally more expensive DibaP-long is always better than kMETIS and JOSTLE in the ℓ_1-norm and thus achieves the remaining best values for all k and both measures. A comparable improvement obtained by DibaP over kMETIS and JOSTLE can be observed for the maximum norm in Table 2. In this norm DibaP-short and DibaP-long are again superior to the other two partitioners with one exception only (EE for $k = 64$).

Note that additional iterations of TruncCons do not always lead to better solutions as DibaP-short computes the best solutions in several cases. The main reason is that the implicit optimization process of TruncCons does not correspond directly to the optimization of the metrics under consideration. Furthermore, the rebalancing process and the fact that TruncCons might have been stopped during hill-climbing *before* a better local optimum is reached play also an important role.

The average improvement to kMETIS w. r. t. the number of boundary nodes in the maximum norm – which can be considered a more accurate measure for communication in parallel numerical solvers than the edge-cut – is 5.5% for DibaP-short and 7.3% for DibaP-long. The gain on JOSTLE is even more than 10%. Given that DibaP's running time is reasonable (see Section 6.2) and that JOSTLE and particularly kMETIS are well-



(a) KMETIS (b) JOSTLE (c) DIBAP-long

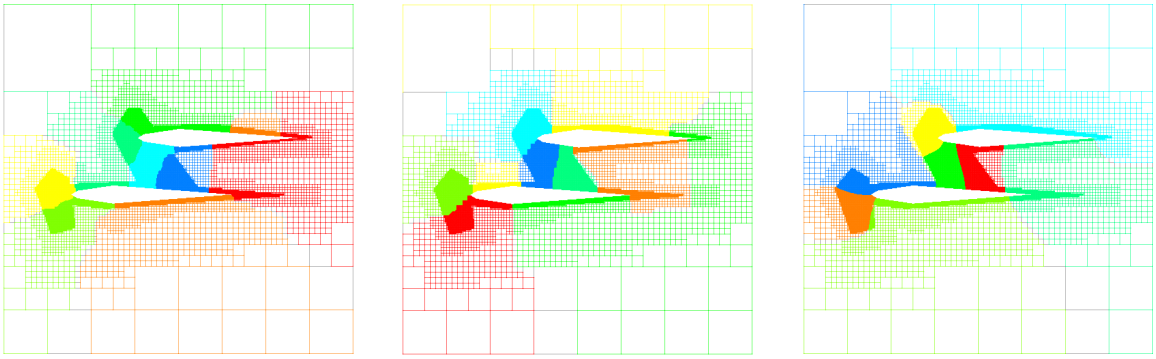
Fig. 6. Partitionings of the graph t60k ($|V| = 60005$, $|E| = 89440$) into $k = 12$ subdomains with the three partitioners.

established partitioning tools, these improvements are quite remarkable. Note that if we are willing to invest more running time, we can often improve DIBAP's average solution quality by computing more initial solutions with BUBBLE-FOS/C or by using higher values of ψ and Λ to circumvent the premature stop of the hill-climbing process [22, Ch. 5.4.1].

A detailed comparison of the diameter values reveals similar results on our benchmark graphs for $k = 16$. DIBAP-long is on average 4.4% (ℓ_1-norm) and 5.9% (ℓ_∞-norm) better than JOSTLE, respectively. On KMETIS the improvements are slightly larger. Since disconnected subdomains (whose diameter is set to ∞) do not enter into these comparisons, the real values of KMETIS and JOSTLE tend to be worse than those computed and used for the comparison above. Moreover, our algorithm yields disconnected subdomains in only 2.1% of the experiments, while KMETIS exhibits a more than doubled ratio of 4.4%. Much worse is JOSTLE, which produces disconnected subdomains in 22.3% of the runs.

Visual Comparison. To provide the reader with a visual impression on how DIBAP's results differ from those of METIS and JOSTLE, we include a 12-partitioning of the 2D graph t60k (also available from Walshaw's archive; our benchmark set contains only 3D graphs) as Figure 6. The partitioning computed by DIBAP ($\Lambda = 12$, $\psi = 18$) has not only fewer cut-edges and boundary nodes in both norms than the other libraries. Its partition boundaries

also appear to be smoother and the subdomains have a smaller maximum diameter (165, compared to 253 (KMETIS) and 179 (JOSTLE)).



(a) KMETIS (b) JOSTLE (c) DIBAP-long

Fig. 7. Partitionings of biplane9 ($|V| = 21701$, $|E| = 42038$) into $k = 8$ subdomains with the three partitioners.

Figure 7 – an 8-partitioning of the smaller graph biplane9 – confirms the above observation on the diameter, resulting from a different consideration of partition shapes. While the maximum diameter values of DIBAP-long and of JOSTLE are close together, the value of KMETIS is worse. Also note the again smoother boundaries produced with DIBAP-long and that both other libraries generate a partition with two large disconnected node sets.

6.2 Running Times

The average running times required by the implementations to partition the "average graph" of the benchmark set are given in Figure 8. Clearly, KMETIS is the fastest and JOSTLE a factor of roughly 2.5 slower. Compared to this, the running times of DIBAP-short and DIBAP-long are significantly higher. This is in particular true for larger k, which is mainly due to the fact that k enters into the running time of DIBAP– in contrast to KMETIS and JOSTLE, where the effect of k on the running time is rather small. The running time dependence of DIBAP on k is nearly linear since doubling k also means doubling the number of diffusion systems to solve. Significantly sublinear time increases are due to an overproportionally large number of inactive nodes. A remedy of the dependence on k

is part of future work. Nonetheless, DIBAP constitutes a vast improvement over previous implementations that use only BUBBLE-FOS/C for partitioning ([23], [32, p. 112]). The acceleration factor lies around two orders of magnitude for the benchmark graphs used here.

Parallel DIBAP. The use of POSIX threads for the three most time-consuming tasks (AMG hierarchy construction, solving linear systems for BUBBLE-FOS/C, and the FOS diffusion calculations within TRUNC-CONS) in our DIBAP implementation yields on average a speedup of 1.55 on the employed dual-core processor. This means that using both cores makes the execution of the program 55% faster compared to the non-threaded version. (Hence, the running times of DIBAP in Figure 8 have to be divided by 1.55 to obtain their approximate parallel counterparts.) Experiments on advanced SMP machines with more cores are planned to investigate the scalability of the thread parallelization. As the FOS diffusion calculations within TRUNCCONS are independent for each partition, we expect a reasonable scalability.

k	KMETIS	JOSTLE	DIBAP-short	DIBAP-long
4	0.33	0.62	4.31	9.80
8	0.34	0.70	7.42	17.66
12	0.35	0.77	10.18	24.86
16	0.36	0.83	12.91	31.67
20	0.37	0.89	15.55	38.32
32	0.39	1.04	20.28	53.64
64	0.43	1.35	34.15	92.76
avg	**0.37**	**0.89**	**14.97**	**38.39**

Fig. 8. Average *non-threaded* running times (in seconds) on Intel Core 2 Duo 6600

While the speed gap compared to the established partitioners is still quite high, the absolute running times of DIBAP are already quite satisfactory (a few seconds to a few minutes for the benchmark graphs). Among other improvements, we plan for an even higher exploitation of the algorithm's natural parallelism. This includes a distributed-memory parallelization and the use of accelerators such as general purpose graphics hardware for the simple diffusive operations within TRUNCCONS. If one assumes a parallel load balancing scenario with k processors for k partitions, one may divide the sequential running times of DIBAP by $k \cdot e$ (where $0 < e \leq 1$ denotes the efficiency of the parallel program). Hence, its parallel running time on k processors can be expected (and is observed in preliminary

experiments) to be in the order of seconds, which is certainly acceptable.

6.3 Best Known Edge-Cut Results

Walshaw's benchmark archive also collects the best known partitionings for each of the 34 graphs contained therein, i. e., partitionings with the lowest edge-cut. Currently, results of more than 20 algorithms are considered. Many of these algorithms are significantly more time-consuming than METIS and JOSTLE used in our experiments above.

With each of the 34 graphs 24 partitionings are recorded, one for six different numbers of subdomains ($k \in \{2, 4, 8, 16, 32, 64\}$) in four different imbalance settings ($0\%, 1\%, 3\%, 5\%$). Using DIBAP in various parameter settings ($\Lambda \leq 15$, $\psi \leq 20$), we have been able to improve more than 80 of these currently best known edge-cut values for six of the eight largest graphs in the archive. The complete list of improvements with the actual edge-cut values and the corresponding partition files are available from Walshaw's archive [42].

Note that none of our records is for $k = 2$. We conjecture that this is the case because the starting solutions computed by BUBBLE-FOS/C are often not really good for $k = 2$. Moreover, these records are mostly held by very time-consuming tailor-made bipartitioning algorithms. Unless they are extended to $k > 2$, their high quality is not likely to sustain for larger k because recursive bipartitioning typically yields inferior results compared to direct k-way methods for large k [36].

7 Conclusions

In this paper we have developed the new heuristic algorithm DIBAP for multilevel graph partitioning. Based on an accelerated diffusion-based local improvement procedure, it attains a very high quality on widely used benchmark graphs: For six of the eight largest graphs of a well-known benchmark set, DIBAP improves the best known edge-cut values in more than 80 (out of 144) settings. Additionally, the very high quality of our new algorithm has been verified in extensive experiments, which demonstrate that DIBAP delivers better parti-

tionings than METIS and JOSTLE – two state-of-the-art sequential partitioning libraries using the KL heuristic. These results show that diffusive shape optimization is a successful approach for providing partitionings of superior quality and very promising to overcome the drawbacks of traditional KL-based algorithms. It should therefore be explored further, both in theory and in practice.

Future Work. To improve the speed of DibaP, a remedy for the nearly linear dependence of the running time on k is of utmost importance. A future MPI parallelization and an implementation of TruncCons on very fast general purpose graphics hardware can be expected to exploit our algorithm's inherent parallelism better and thereby accelerate it significantly in practice. Moreover, it would be interesting to examine how DibaP acts as a load balancer compared to related libraries. Theoretically, starting from our convergence result of this paper, it would be interesting to obtain more knowledge on the relation of the Bubble framework and disturbed diffusion schemes. Of particular concern is how to enforce connected partitions with TruncCons.

Acknowledgments. This work is partially supported by German Research Foundation (DFG) Research Training Group GK-693 of the *Paderborn Institute for Scientific Computation* (PaSCo), by Integrated Project IST-15964 *AEOLUS* of the European Union, and by DFG Priority Programme 1307 *Algorithm Engineering*. Also, the authors would like to thank Stefan Schamberger for many fruitful discussions on the topic and the reviewers for their helpful comments and suggestions.

References

1. S. Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC'07)*, pages 227–236. ACM, 2007.
2. C.-E. Bichot. A new method, the fusion fission, for the relaxed k-way graph partitioning problem, and comparisons with some multilevel algorithms. *Journal of Mathematical Modelling and Algorithms*, 6(3):319–344, 2007.
3. T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *6th SIAM Conference on Parallel Processing for Scientific Computing (PPSC)*, pages 445–452, 1993.
4. U. Catalyurek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, 1999.

5. U. Catalyurek, E. Boman, K. Devine, D. Bozdag, R. Heaphy, and L. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS'07)*. IEEE Computer Society, 2007. Best Algorithms Paper Award.
6. C. Chevalier and F. Pellegrini. Improvement of the efficiency of genetic algorithms for scalable parallel graph partitioning in a multi-level framework. In *Proceedings of the 12th International Euro-Par Conference*, volume 4128 of *Lecture Notes in Computer Science*, pages 243–252. Springer-Verlag, 2006.
7. G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Parallel and Distributed Computing*, 7:279–301, 1989.
8. M. Dellnitz, M. Hessel-von Molo, P. Metzner, R. Preis, and C. Schütte. Graph algorithms for dynamical systems. In *Modeling and Simulation of Multiscale Problems*, pages 619–646. Springer-Verlag, 2006.
9. R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw. Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM. *Parallel Computing*, 26:1555–1581, 2000.
10. R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35:305–320, 2002.
11. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Conference on Design automation (DAC'82)*, pages 175–181. IEEE Press, 1982.
12. D. Harel and Y. Koren. On clustering using random walks. In *Proceedings of 21st Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)*, volume 2245 of *Lecture Notes in Computer Science*, pages 18–41. Springer-Verlag, 2001.
13. B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *Proceedings of Irregular'98*, volume 1457 of *Lecture Notes in Computer Science*, pages 218–225. Springer-Verlag, 1998.
14. B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.
15. B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Proceedings Supercomputing '95*, page 28 (CD). ACM Press, 1995.
16. S. Huang, E. Aubanel, and V. C. Bhavsar. PaGrid: A mesh partitioner for computational grids. *Journal of Grid Computing*, 4(1):71–88, 2006.
17. G. Karypis and V. Kumar. *MeTiS: A Software Package for Partitioning Unstrctured Graphs, Partitioning Meshes, [...], Version 4.0*, 1998.
18. G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.
19. B. W. Kernighan and S. Lin. An efficient heuristic for partitioning graphs. *Bell Systems Technical Journal*, 49:291–308, 1970.
20. R. Khandekar, S. Rao, and U. Vazirani. Graph partitioning using single commodity flows. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 385–390. ACM, 2006.
21. S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–136, 1982.

22. H. Meyerhenke. *Disturbed Diffusive Processes for Solving Partitioning Problems on Graphs*. PhD thesis, Universität Paderborn, 2008.

23. H. Meyerhenke, B. Monien, and S. Schamberger. Accelerating shape optimizing load balancing for parallel FEM simulations by algebraic multigrid. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS'06)*, page 57 (CD). IEEE Computer Society, 2006.

24. H. Meyerhenke and T. Sauerwald. Analyzing disturbed diffusion on networks. In *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC'06)*, volume 4288 of *Lecture Notes in Computer Science*, pages 429–438. Springer-Verlag, 2006.

25. H. Meyerhenke and S. Schamberger. Balancing parallel adaptive FEM computations by solving systems of linear equations. In *Proceedings of the 11th International Euro-Par Conference*, volume 3648 of *Lecture Notes in Computer Science*, pages 209–219. Springer-Verlag, 2005.

26. H. Meyerhenke and S. Schamberger. A parallel shape optimizing load balancer. In *Proceedings of the 12th International Euro-Par Conference*, volume 4128 of *Lecture Notes in Computer Science*, pages 232–242. Springer-Verlag, 2006.

27. F. Pellegrini. A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries. In *Proceedings of the 13th International Euro-Par Conference*, volume 4641 of *Lecture Notes in Computer Science*, pages 195–204. Springer-Verlag, 2007.

28. R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *Lecture Notes in Computer Science*, pages 259–269. Springer-Verlag, 1999.

29. I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

30. S. Schamberger. On partitioning FEM graphs using diffusion. In *Proceedings of the HPGC Workshop of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*. IEEE Computer Society, 2004.

31. S. Schamberger. A shape optimizing load distribution heuristic for parallel adaptive FEM computations. In *8th International Conference on Parallel Computing Technologies (PaCT'05)*, number 2763 in Lecture Notes in Computer Science, pages 263–277. Springer-Verlag, 2005.

32. S. Schamberger. *Shape Optimized Graph Partitioning*. PhD thesis, Universität Paderborn, 2006.

33. K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219–240, 2002.

34. K. Schloegel, G. Karypis, and V. Kumar. Graph partitioning for high performance scientific simulations. In *The Sourcebook of Parallel Computing*, pages 491–541. Morgan Kaufmann, 2003.

35. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

36. H. D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

37. A. J. Soper, C. Walshaw, and M. Cross. A combined evolutionary search and multilevel optimisation approach to graph partitioning. *Journal of Global Optimization*, 29(2):225–241, 2004.

38. D. A. Spielman and S.-H. Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *Proceedings of the 37th Symposium on Foundations of Computer Science (FOCS'96)*, pages 96–105. IEEE Computer Society, 1996.
39. K. Stüben. An introduction to algebraic multigrid. In U. Trottenberg, C. W. Oosterlee, and A. Schüller, editors, *Multigrid*, pages 413–532. Academic Press, 2000. Appendix A.
40. A. Trifunovic and W. Knottenbelt. Parallel multilevel algorithms for hypergraph partitioning. *Journal of Parallel and Distributed Computing*, 68:563–581, 2008.
41. U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2000.
42. C. Walshaw. The graph partitioning archive. http://staffweb.cms.gre.ac.uk/ c.walshaw/partition/, 2007.
43. C. Walshaw and M. Cross. Mesh partitioning: a multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.
44. C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *Parallel Computing*, 26(12):1635–1660, 2000.
45. C. Walshaw, M. Cross, and M. G. Everett. A Localised Algorithm for Optimising Unstructured Mesh Partitions. *International Journal of High Performance Computing Applications*, 9(4):280–295, 1995.
46. G. Zumbusch. *Parallel Multilevel Methods: Adaptive Mesh Refinement and Loadbalancing*. Teubner, 2003.