Karlsruhe Institute of Technology

Bachelor thesis

Engineering Graph Partitioning Algorithms to Minimize Communication Volume

Daniel Seemaier

Date: November 8, 2017

Supervisors: Prof. Dr. rer. nat. Peter Sanders
M. Sc. Sebastian Schlag
Dr. rer. nat. Christian Schulz

Institute of Theoretical Informatics, Algorithmics
Department of Informatics
Karlsruhe Institute of Technology

Abstract

The graph partitioning problem divides the nodes of a graph $G = (V, E)$ into k blocks such that the blocks are balanced up to some imbalance factor and that an objective is optimized. Most software products available focus on optimizing the edge cut of the partition, that is, they aim to minimize the number of edges that run between different blocks.

This thesis extends the graph partitioning framework KaHIP with the ability to minimize another objective that arises in the context of parallel computation, namely the total communication volume. To that end, we try to adjust techniques that are known to produce good results for the edge cut objective and that are already implemented in the KaHIP graph partitioning framework to the total communication volume objective. We then compare our results with METIS.

Zusammenfassung

Das Graphpartitionierungsproblem partitioniert einen Graphen $G = (V, E)$ in k Blöcke derart, dass die Blöcke der Partition bis auf einen bestimmten Faktor balanciert sind und dass eine Zielfunktion optimiert wird. Viele der verfügbaren Softwareprodukte zur Graphpartitionierung minimieren den Kantenschnitt der Partition, also die Anzahl der Kanten, die zwischen verschiedenen Blöcken verlaufen.

In dieser Bachelorarbeit erweitern wir das Graphpartitionierungsframework KaHIP mit einer neuen Zielfunktion die im Kontext von parallelen Berechnungssystemen auftritt, nämlich dem totalen Kommunikationsvolumen der Partition. Dazu passen wir die Techniken, die bereits in KaHIP implementiert sind und bekannt dafür sind, gut für die Kantenschnitt Zielfunktion zu funktionieren, an die neue Zielfunktion an. Anschließend vergleichen wir unsere Resultate mit METIS.

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, den 8. November 2017

Contents

1 Introduction

Graph partitioning is the problem to partition the nodes of a graph into roughly balanced blocks such that some objective is optimized. While the problem sounds abstract at first, it has applications across various disciplines.

For instance, an algorithm for customizable route planning [3] uses graph partitioning to divide road networks into cells that are then preprocessed such that the shortest paths between two boundary nodes of the same cell are already known in advance. This technique speeds up the well-known Dijkstra algorithm by several orders of magnitude.

In the context of parallel computation, graph partitioning is used to map processes onto processing nodes such that the communication between nodes is minimized.

Various software products are available for general purpose graph partitioning, for instance METIS [9], KaHIP [15] or even software that is spezialized on specific types of graphs, such as PUNCH [4] for road networks.

All of those partitioners optimize a common objective, namely the edge cut. That is, they optimize partitions such that only few edges have endpoints in different blocks. While edge cut is the most prominent objective and is de facto established as the standard objective for benchmarks, it is well known that many real-world applications of graph partitioning ask for objectives that are only loosely approximated by the edge cut objective [6].

In this thesis, we extend KaHIP with the ability to minimize the total communication volume, an objective that arises in the context of parallel computation. As a side effect of our work, we decouple local search algorithms from the objective such that new objectives can be implemented more easily.

1.1 Contribution

We extend KaHIP such that is produces better results for the total communication volume objective. We observe that V-cycles as they are known in literature do not strictly improve the total communication volume of a partition. Thus, we suggest a new V-cycle that is designed to prevent a worsening of the total communication volume of a partition if local search guarantees no worsening.

1.2 Structure of Thesis

In Chapter 2, we present general definitions and notation that are used throughout the rest of this thesis and give a more formal introduction to graph partitioning. Next, we summarize multi-level graph partitioning and other techniques that are relevant to this thesis as well as the graph partitioning framework we use, namely KaHIP, in Chapter 3. When then move on to algorithms that are explicitly designed to minimize communication volume in Chapter 4 before we give an experimental evaluation of them in Chapter 5.

2 Fundamentals

This chapter introduces general definitions and notation that are used in the thesis. Moreover, the graph partitioning problem is introduced formally.

2.1 Graph Theory

An (undirected, weighted) *graph* $G = (V, E, c, \omega)$ consists of a finite set V and a binary relation on V, $E \subseteq \{\{u, v\} \mid u, v \in V\}$. We set $n := |V|$ and $m := |E|$. When we talk about multiple graphs, we use $V(G)$ and $E(G)$ to denote the node and edge set of graph G. The elements of V are called *nodes* and the elements of E are called *edges*. We say that a graph is *simple*, if it is undirected and does not contain any edge $\{v, v\}$ with $v \in V$. If not stated otherwise, every graph is a simple graph throughout this thesis. The function $c : V \to \mathbb{R}$ assigns *weights* to the nodes while $\omega : E \to \mathbb{R}$ assigns weights to the edges. We extend c and ω to sets in a natural way, i.e. $c(V' \subseteq V) := \sum_{v \in V'} c(v)$ and $\omega(E' \subseteq E) := \sum_{e \in E'} \omega(e)$. Note that the input graphs used in this thesis have unit edge weights and thus $c(V) = n$. This, however, changes in the course of the algorithm.

Two nodes $u, v \in V$ are called *adjacent* or *connected* if $\{u, v\} \in E$. Likewise, two edges $i, j \in E$ are said to be *incident* if $i \cap j \neq \emptyset$. Finally, a node $v \in V$ and an edge $e \in E$ are *incident* if $v \in e$. The set of all nodes that are adjacent to a node $v \in V$ is called the *neighborhood* of v and is denoted by $\Gamma(v)$. Its size is the degree of v, denoted by $d(v) = |\Gamma(v)|$. Asymptotic running times sometimes depend on the maximum degree $\Delta(G) := \max_{v \in V} d(v)$ that occurs in G.

A *matching* M is a subset of E, $M \subseteq E$, with the characteristic property that $e \cap e' = \emptyset$ for all $e, e' \in M$. A *maximal* matching is one with the property that there is no $e \in E \setminus M$ such that $M \cup \{e\}$ is still a matching.

A *k–way partition* of V is a set of k disjoint *blocks* $V_1, \ldots, V_k$ that covers V, i.e. $V = \bigcup_i V_i$. A 2–way partition is also called *bipartition*. A node that is adjacent to at least on node in another block is a *boundary node*. Edges that connect nodes in different blocks are *cut edges*. We define the set of all cut edges between two blocks i and j, $E_{ij} := \{\{u, v\} \in E \mid u \in V_i \text{ and } v \in V_j\}$.

2.2 Graph Partitioning

The *graph partitioning problem* takes a graph $G = (V, E, c, \omega)$ and an integer $k \geq 2$ and provides a k–way partition of V that optimizes some objective $J(V_1, \ldots, V_k)$. The partition must further fulfill a *balance constraint* that limits the size of each block, $|V_i| \leq (1 + \epsilon)\lceil \frac{n}{k} \rceil$, where $\epsilon \geq 0$ is some imbalance parameter. This problem is known to be $\mathcal{NP}$-complete [8] and therefore, we focus on heuristics rather than exact algorithms.

The most prominent objective is to minimize the *edge cut*,

$$J = \sum_{i<j} \omega(E_{ij}).$$

In this thesis, we will focus on another objective, namely the *total communication volume* of a partition. It is defined as follows.

Total Communication Volume. This objective models the communication required between applications spread across a parallel computation system. It is given by

$$J = \sum_{v \in V} c(v) D(v),$$

where $D(v)$ denotes the number of blocks that contain elements adjacent to v, excluding the one that contains v. Figure 2.1 shows a graph with its total communication volume.

Maximum Communication Volume. If only the computation unit with the highest communication volume is relevant, the maximum communication volume is a more suitable objective, given by

$$J = \max_i \sum_{v \in V_i} c(v) D(v).$$

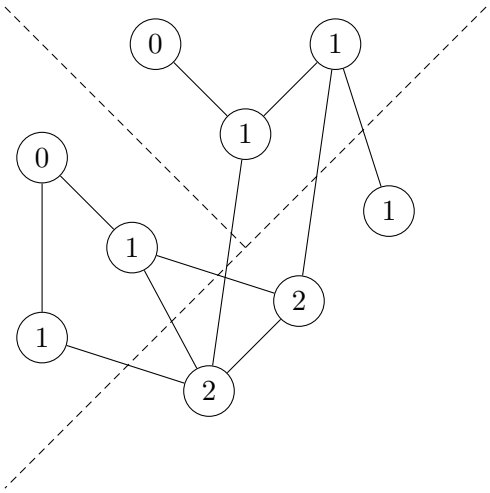


Figure 2.1: A graph with unit node weights. Nodes are labeled with their $D(\cdot)$ value. The total communication volume is $2 + 2 + 5 = 9$ and the maximum communication volume is 5.

3 Related Work

In this chapter, we describe important graph partitioning techniques and software that we use in this thesis. This includes the multi-level partitioning scheme, local search algorithms and KaHIP [15]. We also outline the benefits of these techniques when used with the edge cut objective. Only in Chapter 4, we focus more on the total communication volume objective.

An extensive overview of graph partitioning techniques and heuristics is available at [2]. Here, we only summarize techniques that are relevant to our work.

3.1 Multi-Level Graph Partitioning

Hendrickson and Leland [7] described a multi-level graph partitioning scheme in 1995 that has since been used successfully in multiple graph partitioning software, including KaHIP [15] and METIS [9]. In its most rudimentary form, the scheme partitions a graph $G_0 = (V_0, E_0, c_0, \omega_0)$ in three phases: Coarsening, initial partitioning and uncoarsening.

Coarsening. During the coarsening phase, the scheme produces a hierarchy of smaller graphs $G_1, \ldots, G_N$ by repeatedly *contracting* a set of edges.

Edge contraction describes the following operation: Given an edge $\{v_1, v_2\} \in E$, remove v_1 and v_2 from G and all edges incident to them. Then, insert a new node v with $c(v) = c(v_1) + c(v_2)$ and neighborhood $\Gamma(v) = \Gamma(v_1) \cup \Gamma(v_2)$. If this process would produce two parallel edges, a single edge with weight equal to the sum of both edge weights is inserted instead. Figure 3.1 illustrates the contraction of a single edge in a graph without node weights.

This process is aborted once the number of nodes falls below a certain threshold. In the context of this hierarchy, the input graph G_0 is called the *finest* graph or simply input graph whereas G_N is called the *coarsest* graph. Hendrickson et al. use a maximal matching algorithm to identify the set of edges that are to be contracted. KaHIP also implements label propagation to obtain the set, see Section 3.2.

Initial Partitioning. Once the coarsest graph has been obtained, an initial partitioning algorithm is used to compute a k–way partition. Since the coarsest graph contains few nodes compared to the finest graph, this algorithm can be relatively slow.

Uncoarsening. Finally, the initial partition is iteratively projected onto the next finer graph by uncontracting the contracted edges. After each step, a local search algorithm is executed to optimize the objective. We describe the local search algorithms that we use in Section 3.1.2.

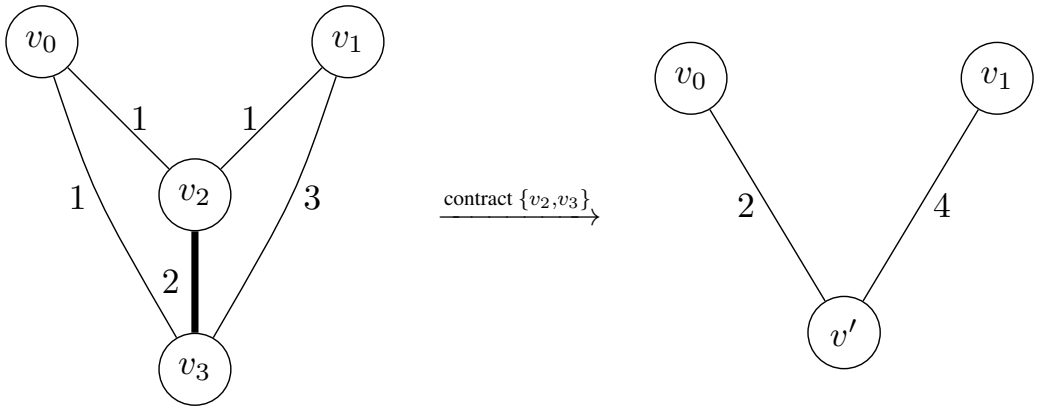


Figure 3.1: Contraction of edge $\{v_2, v_3\}$. Node weights are omitted.

Note that the edge cut is preserved throughout the hierarchy: the weight of an edge $\{v, u\}$ on a coarser level is equal to the number of edges between the nodes that correspond to v and u on the finest level. Thus, an improvement of the edge cut on a coarser level is also an improvement on the input graph.

With this in mind, the intuition behind multi-level graph partitioning is clear. A movement of a coarser node corresponds to the movement of a whole set of nodes on the input graph. Therefore, local search algorithms in a multi-level approach achieve a more global view, leading to improved partition quality compared to an approach that only uses local search on the input graph.

3.1.1 V-Cycles

Walshaw [16] suggested an iterated multi-level partitioning scheme that aims to improve a given partition. A single run of the scheme is known as *V-cycle* and is similar to the multi-level partitioning scheme that computes a partition from scratch. It works as follows.

First, repeat the coarsening phase as described in Section 3.1. This time, however, exclude any boundary edge from contraction. Nodes of the input graph that belong to the same coarsest node now also belong to the same block. Thus, the input partition can be used as initial partition on the coarsest level. Finally, repeat the uncoarsening phase as already described.

If edge cut is the objective, it is useful to execute multiple V-cycles after a partition has been obtained. This is due to the fact that a V-cycle can never worsen the partition as long as local search guarantees no worsening since the edge cut is preserved throughout the

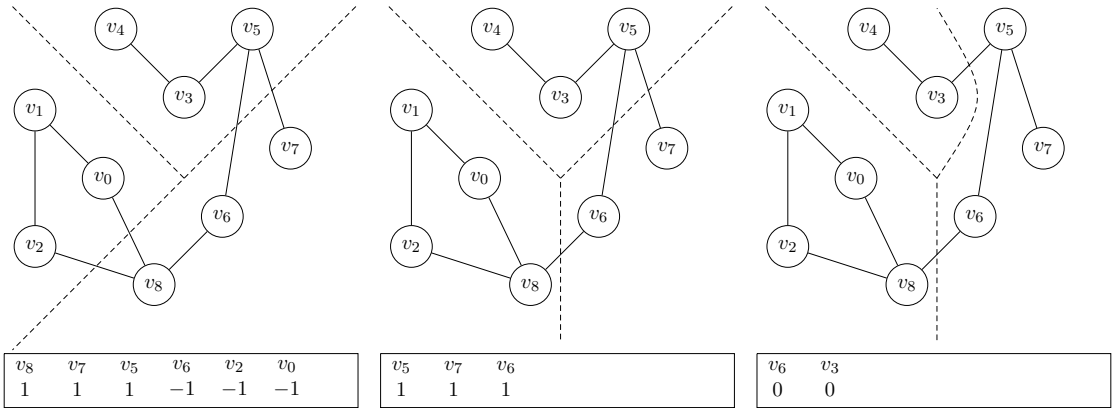


Figure 3.2: The first three steps of the FM algorithm. The boxes show all nodes that are currently in the priority queue (top row) with their corresponding gain values (bottom row, based on the total communication volume objective). In each step, `deleteMax` returns the left-most node. The objective is total communication volume. The balance constraint is assumed to be flexible enough to allow for all movements.

hierarchy. However, a random matching algorithm or random tie-breaking in the coarsening phase leads to different boundaries in each cycle. Hence, local search algorithms can find improvements even though they were unable to climb out of a local optima in the previous cycle. We will discuss V-cycles with the total communication volume objective in mind in Chapter 4.

3.1.2 Local Search

In this section, we describe the local search algorithms that we use in our experiments. Both algorithms assign *gain values* to nodes, that is, the change in the objective when moving a node to another partition. We denote the gain value of a node v when moved to block p by $g_p(v)$. Since we always move nodes to blocks that maximize their gain values, we define $g(v) \coloneqq \max_p g_p(v)$ and say that $g(v)$ is *the* gain value of v.

Note that both local search algorithms are already implemented in KaHIP, but since they are central to the thesis, we cover them here regardless.

Fiduccia-Mattheyses Algorithm

The Fiduccia-Mattheyses (FM) algorithm was first invented by Fiduccia and Mattheyses [5] as a local search algorithm for 2–way partitions. Here, we describe an improved version of the algorithm by Karypis and Kumar [10] that can improve k–way partitions. The algorithm works as follows.

First, some start nodes are inserted into a priority queue with their gain values as keys. Karypis and Kumar use only boundary nodes for initialization. However, since some

objectives could profit from the movement of non-boundary nodes, we declare the nodes that are used for initialization as input parameter of the algorithm. Next, the algorithm retrieves the node v with the highest gain from the priority queue and tries to move the node to its corresponding block. If the movement would violate the balance constraint, it is skipped. Otherwise, the movement is performed and the gain values of other nodes that are affected by the movement and that still reside in the priority queue are updated. If neighbors of v become boundary nodes due to the movement, they are inserted into the priority queue (but only if they were not previously moved). Figure 3.2 illustrates three steps of this algorithm.

After a certain number of nodes were moved without improving the partition, we stop the search after and revert all movements until the best partition that occurred during the process is restored.

Modification. The original version of FM as well as the improved one use *bucket queues* as priority queues. This allows the algorithm to update gain values in amortized constant time. However, bucket queues can only be used if the gain values have certain properties; see [5] for details. Since not all objectives fulfill these requirements (and our implementation should be independent from the objective), we use *binary heaps* instead.

Running Time. The running time of the algorithm depends on the priority queue as well as on the objective. Regarding the priority queue, each node movement requires one `deleteMax` operation and one `insert` operation. After a movement, the gain values of some other nodes might change and thus require a `changeKey` operation for each of them. Assuming that the gain value of single node can be updated in constant time, this yields

$$T = n(T_{\texttt{insert}} + T_{\texttt{deleteMax}}) + n\sigma T_{\texttt{changeKey}},$$

where σ denotes the number of nodes whose gain values might have changed after a movement. This value depends on the objective.

We cover two cases of σ.

- The edge cut objective only requires to update the gain value of each neighbor after a node movement. In this case, $n\sigma$ can be replaced by m since $\sum_{v \in V} d(v) = 2m$.
- Other objectives like total communication volume or graph and index compression additionally require to update the gain value of each neighbor's neighbor. Analogously, this yields $n\sigma \approx m\Delta$.

Binary heaps implement each operation in $\log(n)$ time. Hence, the total running time of the algorithm is $\mathcal{O}(m \log(n))$ in the first case and $\mathcal{O}(m\Delta \log(n))$ in the second case. Note that an implementation for edge cut in $\mathcal{O}(m)$ is possible if bucket queues are used instead; see [5].

Multi-try FM

Sanders et. al. were able to achieve partitions of higher quality by using a highly localized version [12] of the FM algorithm described above. Instead of initializing the priority queue with all boundary nodes, they repeatedly initialize the queue with a single boundary node. Again, after a node was moved, its neighbors are added to the priority queue and thus become eligible for movement. A node that was eligible for movement at some point in time is said to be *touched*. Once the search stopped, a new search with another boundary node that hasn't been touched yet is started. This process is repeated until all nodes or, to reduce running time, a configurable percentage of nodes were touched.

Since already touched nodes don't become eligible for movement in later searches, the algorithm can be implemented with the same asymptotic running time as the normal FM algorithm as described in Section 3.1.2.

We use the *adaptive stopping criteria* introduced in [11] to determine when to stop a search round.

3.2 KaHIP

We use the KaHIP framework (Karlsruhe High Quality Graph Partitioning) [15] to implement and evaluate our experiments. Its manual is available at [14]. KaHIP consists of multiple programs, but we only use its multi-level graph partitioner, KaFFPa (Karlsruhe Fast Flow Partitioner).

The software implements several novel techniques to obtain partitions of high quality. They are described extensively in [13].

KaFFPa offers several pre-configurations that we use in our experiments, namely `fast`, `eco` and `strong`. Each configuration provides a different trade-offs between partition quality and running time with `fast` offering the lowest running time and `strong` the best partition quality. An extensive description of each configuration is available at [13, p. 62f].

Furthermore, there are `-social` pre-configurations specially designed to partition large social networks. Most importantly, they use label propagation to cluster and contract the graph rather than matchings; see [13, p. 121ff]. We use them whenever we experiment on social graphs since they are known to produce the best results for them.

4 Algorithms for Communication Volume

We use the FM and Multi-try FM algorithms to improve the total communication volume of a partition. While both algorithms are already implemented in KaHIP, we change them such that the calculation of gain values is based on the total communication volume objective rather than edge cut. To this end, we describe how to calculate these gain values and how to update them efficiently after a node movement.

Secondly, we suggest a modified V-cycle that is designed to guarantee no worsening of the total communication volume during the course of the V-cycle.

4.1 Computation of Gain Values

In this section, we describe how we calculate and update gain values for the total communication volume objective.

To begin with, let $G = (V, E, c, \omega)$ be a graph with a k-way partition $V = \bigcup_i V_i$. We use $d_p(v) \coloneqq |\Gamma(v) \cap V_p|$ to denote the number of neighbors of node v in block p. Recall that the cost contribution of a single node v is given by $J(v) = c(v)D(v)$. The node weight $c(v)$ is constant and movements only change $D(v)$.

Consider the movement of $v \in V_p$ from block p, the source partition, to p', the target partition. We analyze how this changes the $D(\cdot)$ values of v and its neighbors.

- If v is not adjacent to p', but is adjacent to p, its cost increases. On the other hand, its cost decreases if it is adjacent to p' but not to p. In the other cases, i.e. it is adjacent or not adjacent to both blocks, its cost doesn't change.
- The neighborhood of a neighbor $u \in \Gamma(v)$ changes as follows. It loses a neighbor in block p and gains a neighbor in block p'. The loss decreases u's cost if v is its only neighbor in block p and u itself does not belong to block p. Likewise, u's cost increases if it does not belong to block p' and has no neighbors in that block.

Thus, the gain value of a node can be written as

$$g_{p'}(v) \coloneqq \left(\mathbb{1}_{\{d_p(v)=0 \text{ and } d_{p'}(v)>0\}} - \mathbb{1}_{\{d_p(v)>0 \text{ and } d_{p'}(v)=0\}}\right) c(v) + \sum_{u \in \Gamma(v) \setminus V_p} \mathbb{1}_{\{d_p(v)=1\}} c(u) - \sum_{u \in \Gamma(v) \setminus V_{p'}} \mathbb{1}_{\{d_p(u)=0\}} c(u),$$

where we make use of indicator variables to ease notation, $\mathbb{1}_{\{\text{cond.}\}} \coloneqq \begin{cases} 1, & \text{cond. is met} \\ 0, & \text{otherwise} \end{cases}$.

The running time of this step is as follows. First, we need $\mathcal{O}(m)$ time to compute the $d_p(\cdot)$ values for all blocks p. To calculate the gain value of a single node for a single partition, we must look at all of its neighbors. Therefore we need $\mathcal{O}(mk)$ time to calculate all gain values or $\mathcal{O}(\Delta k)$ time to calculate the gain value of a single node.

Next, consider that node v was moved from partition p to p'. We need to decrease $d_p(u)$ and increase $d_{p'}(u)$ by one for each $u \in \Gamma(v)$. Thus, the gain values of $\Gamma(v)$ and $\bigcup_{u \in \Gamma(v)} \Gamma(u)$ might change. This leads to $\mathcal{O}(\Delta^2)$ updates after a single node movement, compared to $\mathcal{O}(\Delta)$ updates if edge cut is used as objective.

The movement only affects $g_p(\cdot)$ and $g_{p'}(\cdot)$ because other gain values are not influenced by $d_p(v)$ or $d_{p'}(v)$. After updating the gain values of a node, we must check whether a new partition yields the best gain value for that node. This requires $\mathcal{O}(k)$ time if the updates decreased the maximum gain of the node, because then we must compare the new gain value to all other gain values. Thus, the overall worst time complexity of this process is $\mathcal{O}(\Delta^2 k)$.

Since the calculation of the gain value of a single node from scratch takes more than constant time, we rather calculate the differences of all gain values that might have changed. We do this in three steps. First, we observe how the movement of v itself, that is, without considering $d(\cdot)$, affects other gain values. Secondly, we consider the changes in $d(\cdot)$ and describe how they influence the gain values of neighbors of v before we, thirdly, describe how they influence the gain values of neighbors of v's neighbors. In each and every case, we describe why the change occurs and then give a more algorithmic instruction on how to implement it. Note that we use a star as subscript, e.g. $g_\star(x)$, if the change applies to all $g_{\tilde{p}}(x)$ for all blocks $\tilde{p}$.

- If v has only one neighbor u in p, moving that neighbor to another block decreases the partition's cost. This was not previously accounted for in u's gain value and thus, we must increase its gain value by $c(v)$.

  For all $u \in \Gamma(v) \cap V_p$: increase $g_\star(u)$ by $c(v)$ if $d_p(v) = 1$.
- On the other hand, if v has no neighbors in block p, then moving any neighbor $u \in \Gamma(v)$ to block p increases the partition's cost. This was not previously accounted for and thus, we must decrease their gain values by $c(v)$.

  For all $u \in \Gamma(v) \setminus V_p$: decrease $g_p(u)$ by $c(v)$ if $d_p(v) = 0$.
- If v has precisely one neighbor u in block p', then moving v to another block no longer decreases $D(v)$. Thus, we must decrease its gain value by $c(v)$.

  For all $u \in \Gamma(v) \cap V_{p'}$: decrease $g_\star(u)$ by $c(v)$ if $d_{p'}(v) = 1$.
- Finally, if v has no neighbors in block p', then the movement of another neighbor to block p' no longer increases $D(v)$ (but it did before). Thus, we must increase the gain values of all neighbors by $c(v)$.

  For all $u \in \Gamma(v) \setminus V_{p'}$: increase $g_{p'}(u)$ by $c(v)$ if $d_{p'}(v) = 0$.

Next, we describe how the change of the $d(\cdot)$ values affect gain values of v's neighbors. Let $u \in \Gamma(v)$.

- If $d_p(u)$ changed to 0, then u lost its last neighbor in block p. Thus, moving u to block p no longer decreases the number of adjacent partitions of u.

  For all $u \in \Gamma(v) \setminus V_p$: decrease $g_p(u)$ by $c(u)$ if $d_p(u) = 0$.
- If, however, $u \in V_p$ and $d_p(u)$ changed to 0, then moving u to any other partition no longer increases the number of blocks that u is adjacent to. Thus, we need to increase its gain value by $c(u)$.

  For all $u \in \Gamma(v) \cap V_p$: increase $g_\star(u)$ by $c(u)$ if $d_p(u) = 0$.
- If $d_{p'}(u)$ changed to 1, then u gained a new neighbor in block p'. Thus, moving u to block p' no longer decreases the number of blocks that u is adjacent to.

  For all $u \in \Gamma(v) \setminus V_{p'}$: increase $g_{p'}(u)$ by $c(u)$ if $d_{p'}(u) = 1$.
- Lastly, if $u \in V_{p'}$ and $d_{p'}(u)$ changed to 1, then moving u to any other partition no longer increases the number of blocks that u is adjacent to. Thus, we must decrease its gain value by $c(u)$.

  For all $u \in \Gamma(v) \cap V_{p'}$: decrease $g_\star(u)$ by $c(u)$ if $d_{p'}(u) = 1$.

Finally, we need to update the gain values of nodes that share a neighbor with v, i.e. neighbors of neighbors of v. Again, let $u \in \Gamma(v)$.

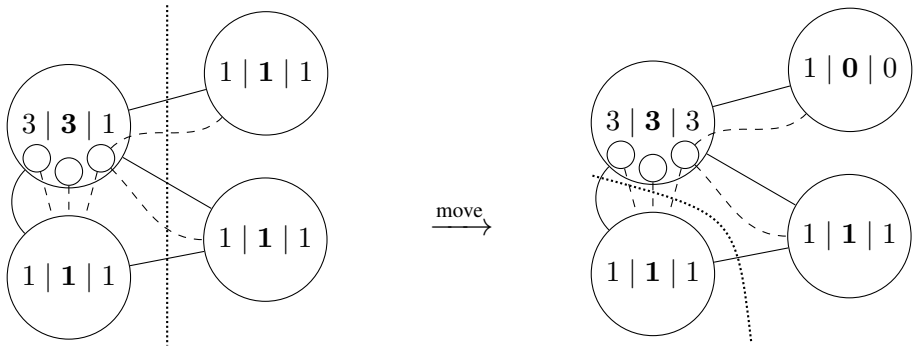
- If $d_p(u)$ changed to 1, there is only one neighbor of u left in block p. When moving that neighbor to another block, $D(u)$ decreases and thus, we must increase that nodes gain value, unless u itself is in block p.

  For all $w \in \Gamma(u) \cap V_p$: increase $g_\star(w)$ by $c(u)$ if $d_p(u) = 1$ and $u \notin V_p$.
- If $d_p(u)$ changed to 0, u has no more neighbors in block p. Thus, moving any other neighbor to block p increases $D(u)$, unless u itself is in block p.

  For all $w \in \Gamma(u) \setminus V_p$: decrease $g_p(w)$ by $c(u)$ if $d_p(u) = 0$ and $u \notin V_p$.
- For neighbors of u in block p', we must look at two cases. Both cases only apply if u itself is not in block p'. First, assume $d_{p'}(u) = 1$: Then there is one neighbor in block p' and moving it decreases $D(u)$ by one. On the other hand, assume that $d_{p'}(u) = 2$. Then before the movement, there was only one neighbor in block p'. Moving that one decreases $D(u)$. But since there are two neighbors now, that can no longer happen due to a single movement. Thus, its gain value decreases.

  If $u \notin V_{p'}$: for all $w \in \Gamma(u) \cap V_{p'}$: increase $g_\star(w)$ by $c(u)$ if $d_{p'}(u) = 1$ and decrease it if $d_{p'}(u) = 2$.
- Finally, consider neighbors of u that are not in block p'. If $d_{p'}(u) = 1$ and u itself is not in block p', moving them to longer increases $D(u)$. Thus, their gain values increase.

  For all $w \in \Gamma(u) \setminus V_{p'}$: increase $g_{p'}(w)$ by $c(u)$ if $d_{p'}(u) = 1$ and $u \notin V_{p'}$.



(a) TCV: 6 on coarser level, 4 on finer level.

(b) TCV: 5 on coarser level, 5 on finer level.

Figure 4.1: Example showing that an improvement on a coarser level might be a worsening on a finer level. Big bubbles represent coarser nodes. Small bubbles inside are finer nodes with unit node weights that correspond to them. Label: **(1st)** node weight, **(2nd)** cost contribution on the coarser level, **(3rd)** cost contribution on the finer level.

4.2 V-Cycle for Communication Volume

Recall that coarsening and uncoarsening preserves the edge-cut and thus a V-cycle cannot worsen the objective. Unfortunately, this does not apply to the total communication volume objective as demonstrated in Figure 4.1.

To fix this, we suggest to modify the contraction scheme such that edges incident to boundary nodes are no longer contracted. We illustrate the difference to the normal V-cycle in Figure 4.2. Indeed, we can show that this prevents a worsening of the objective through repeated V-cycles.

Lemma 4.2.1. *Our modified V-cycle preserves the total communication volume during coarsening.*

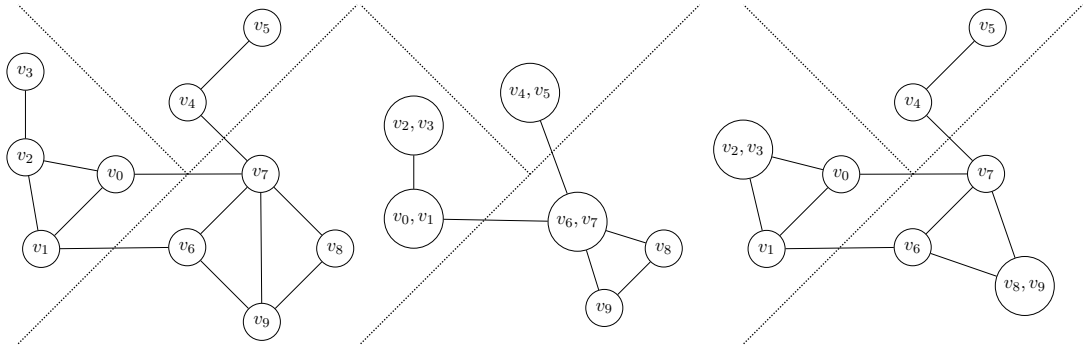
Proof. Since no edges adjacent to boundary nodes are contracted, both graphs share the same boundary nodes. Furthermore, $D(v)$ and $c(v)$ are the same on both graphs for all boundary nodes v. Thus, the total communication volumes of both graphs are the same. □

Lemma 4.2.2. *Let J_i be the total communication volume of graph G_i. Then, $J_{i-1} \leq J_i$.*

Proof. Consider the contribution of a single node $v \in V_i$ to the objective on level $i > 0$, namely $J_i(v) = c_i(v)D_i(v)$. The node v corresponds to a set of nodes $S = \{v'_1, \ldots, v'_N\}$ on level $i-1$ with $c_i(v) = c_{i-1}(S)$. Since $D_i(v) \geq D_{i-1}(v'_j)$, $1 \leq j \leq N$, we have

$$J_{i-1}(S) = \sum_{1 \leq j \leq N} c_{i-1}(v'_j)D_{i-1}(v'_j) \leq \sum_{1 \leq j \leq N} c_{i-1}(v'_j)D_i(v) = c_i(v)D_i(v) = J_i(v).$$

Thus, $J_{i-1} = \sum_{v' \in V_{i-1}} J_{i-1}(v') \leq \sum_{v \in V_i} J_i(v) = J_i$. □



(a) Uncontracted graph. **(b)** Normal V-cycle. **(c)** Our V-cycle.

Figure 4.2: Graph **(b)** could be the result of a contraction made by the normal V-cycle. While the partition is preserved, boundary nodes might get contracted. Graph **(c)** shows a contraction that could be the result of our V-cycle: All boundary nodes are preserved and only edges incident to non-boundary nodes are potentially contracted.

Corollary 4.2.3. *Our modified V-cycle guarantees no worsening of the total communication volume if local search guarantees no worsening.*

Proof. Let J_i and J_i' be the total communication volume on level i before and after local search, respectively. Then, $J_0 = J_N$ by Lemma 4.2.1, $J_i' \leq J_i$ by assumption and $J_{i-1} \leq J_i'$ by Lemma 4.2.2. The claim $J_0' \leq J_0$ follows by induction. □

We implement our novel V-cycle by extending the algorithm that is already implemented in KaHIP. Suppose that KaHIP already generated the coarser graphs $G_1, \ldots, G_i$. Now, it generates a mapping $f : V(G_i) \to V(G_{i+1})$ such that the coarser node v in G_{i+1} corresponds to the node set $f^{-1}(v)$ in G_i. Based on this mapping, it generates the coarser graph.

We alter $V(G_{i+1})$ and f as depicted in Algorithm 1. The idea is to add a node to G_{i+1} for each boundary node in G and change f such that boundary nodes map to their individual nodes. Finally, we remove nodes from G_{i+1} such that f is a surjection.

The running time of this extension is dominated by determining which nodes are boundary nodes. All other steps run linear in the number of nodes.

Note that the algorithm could be simplified if the input mapping was restricted to matchings. We scan for the gaps after the boundary nodes were remapped because KaHIP also implements label propagation, see Section 3.2. In this case, more than two nodes might be mapped to the same coarser node.

Algorithm 1: Maps boundary nodes to their individual coarser nodes.

`Data`: Mapping $f : V(G_i) \to V(G_{i+1})$. Assume $V(G_{i+1}) = \{0, \ldots, |V(G_{i+1})| - 1\}$.
`Result`: Altered coarser mapping such that the boundary of G_i is preserved.

$B \leftarrow$ boundary nodes in G_i
$s \leftarrow |V(G_{i+1})|$

// create new nodes and alter mapping
`foreach` $v \in B$ `do`
    $V(G_{i+1}) \leftarrow V(G_{i+1}) \cup \{s\}$
    $f(v) \leftarrow s$
    $s \leftarrow s + 1$
assert $s = |V(G_{i+1})|$
assert for all $v \in B$: $f(v)$ is unique, i.e. there is no $u \neq v$ with $f(u) = f(v)$

// detect gaps in mapping
`hit` $\leftarrow$ new Array of size s and initialized with `false`
`foreach` $v \in V(G_{i+1})$ `do`
    `hit`$[f(v)] \leftarrow$ `true`

// accumulate gaps to calculate the offsets that close them
`offset` $\leftarrow$ new Array of size s and initialized with 0
`d` $\leftarrow 0$
`foreach` $v \in V(G_{i+1})$ `do`
    `if` *hit*$[v]$ `then`
        `offset`$[v] \leftarrow$ `d`
    `else`
        `d` $\leftarrow$ `d` $+ 1$

// close the gaps
`foreach` $v \in V(G_{i+1})$ `do`
    $f(v) \leftarrow f(v) -$ `offset`$[f(v)]$

assert the last `d` nodes in $V(G_{i+1})$ are no longer mapped
$V(G_{i+1}) \leftarrow V(G_{i+1}) \setminus \{$`s - d`$, \ldots,$ `s` $- 1\}$

5 Experimental Evaluation

The structure of this chapter is as follows. We begin by describing the environment of our experiments. Afterwards, we describe individual steps of our experiments and their tuning parameters without a specific experiment in mind. Finally, we describe and evaluate our experiments using those steps in Section 5.3.

5.1 Environment and Instances

All algorithms were implemented in `C++` within the KaHIP graph partition framework [14] in version 2.0. The code was compiled using GCC in version `4.8.5` and the following compiler flags: `-funroll-loops`, `-fno-stack-limit` and `-O3`.

Hardware. Our experiments were executed on two machines. Machine A has four Intel Xeon E5-4640, 512 GiB ECC main memory and is running Ubuntu 12.04. Machine B has two Intel Xeon E5-2670 v3, 128 GiB ECC main memory and is running Ubuntu 14.04LTS.

5.1.1 Instances

The graphs used in our experiments are divided into two different categories: Social graphs, listed in Table 5.1, and normal graphs, listed in Table 5.2. All graphs are unweighted and were taken from [1].

Social graphs listed in Table 5.1 were initially partitioned using KaHIP's `ecosocial` and (or) `strongsocial` preconfigurations. On the other hand, for graphs listed in Table 5.2, the preconfigurations `eco` and `strong` were used to obtain the initial partition. We do this because these preconfigurations are known to yield the best results for each category.

Whenever we refer to the "`eco(social)`" preconfiguration, we mean `ecosocial` if the graph is in Table 5.1 and `eco` otherwise. The same applies to `strong(social)`.

5.2 Partitioning Steps and Tuning Parameters

Steps described in this section don't represent full experiments but rather building blocks. The experiments that we describe in Section 5.3 are build with these blocks and concrete

Graph	$\|V\|$	$\|E\|$
Web Graphs		
uk-2002	18 520 486	261 787 258
eu-2005	862 664	16 138 468
in-2004	1 382 908	13 591 473
cnr-2000	325 557	2 738 969
Citation Network Graphs		
coPapersCiteseer	434 102	16 036 720
coPapersDBLP	540 486	15 245 729
citationCiteseer	268 495	1 156 647
coAuthorsDBLP	299 067	977 676
coAuthorsCiteseer	227 320	814 134

Table 5.1: Social graphs.

values for tuning parameters. Most of these steps were already described in Chapter 3 and Chapter 4. We repeat them here briefly to lay down the vocabulary used in the following sections.

KaHIP. This step obtains or refines a partition by running KaHIP with a certain preconfiguration. We use this step to obtain an initial partition of the graph. Note that partitions obtained this way are optimized for edge cut rather than total communication volume.

KaHIP supports several preconfigurations that offer a trade of between partition quality and running time. We use two of those to observe the influence of a better initial partition (in terms of edge cut). Namely, we use the preconfigurations `eco`, `ecosocial`, `strong` and `strongsocial`. `strong` finds partitions of higher quality than `eco` but takes longer. The `-social` variants of those preconfigurations use label propagation rather than matchings to contract the graph. This achieves better results for social graphs and thus, we use them whenever we experiment on social graphs. Since it is clear whether we use the `-social` variant of a preconfiguration or not, we omit this difference in naming in the following experiments.

Tuning parameters: Preconfiguration

FM refinement. This step uses the FM algorithm as described in Section 3.1.2. The algorithm is stopped after $\alpha \frac{n}{100}$ (but at least 15) consecutive movements worsened the objective. Thereby is $\alpha = 1$ if the preconfiguration is `eco` and $\alpha = 3$ if it is `strong`. These values are from KaHIP. We schedule rounds until a maximum number of rounds is reached or a round did not yield an improvement. The refinement can be used single-level or with one of the two V-cycles.

Tuning parameters: Maximum number of rounds, refinement type

Graph	$\|V\|$	$\|E\|$
Walshaw Graphs		
auto	448 695	3 314 611
m14b	214 765	1 679 018
144	144 649	1 074 393
wave	156 317	1 059 331
bcsstk30	28 924	1 007 284
bcsstk32	44 609	985 046
598a	110 971	741 934
fe_rotor	99 617	662 431
bcsstk31	35 588	572 914
fe_tooth	78 136	452 591
fe_ocean	143 437	409 593
brack2	62 631	366 559
bcsstk29	13 992	302 748
bcsstk33	8 738	291 583
finan512	74 752	261 120
vibrobox	12 328	165 250
fe_body	45 087	163 734

Graph	$\|V\|$	$\|E\|$
Walshaw Graphs		
fe_pwt	36 519	144 794
wing	62 032	121 544
t60k	60 005	89 440
wing_nodal	10 937	75 488
memplus	17 758	54 196
fe_sphere	16 386	49 152
cti	16 840	48 232
4elt	15 606	45 878
cs4	22 499	43 858
fe_4elt2	11 143	32 818
crack	10 240	30 380
whitaker3	9 800	28 989
data	2 851	15 093
3elt	4 720	13 722
add32	4 960	9 462
add20	2 395	7 462
uk	4 824	6 837

Table 5.2: Normal graphs.

Multi-try FM refinement: This step is similar to the previous one but uses Multi-try FM as described in Section 3.1.2. We stop the search using the adaptive stopping criteria from [11] with $\alpha = 3$ and $\beta = \log(|V|)$.

Tuning parameters: Maximum number of rounds, refinement type

Refinement Types. The FM or Multi-try FM refinement can be used together with one of the following refinement types. They can be used with the normal V-cycle as described in Section 3.1.1, with our novel V-cycle that we describe in Section 4.2 or single-level. In the last case, we only run the local search algorithm on the input graph, i.e. we don't build a graph hierarchy at all.

5.3 Experimentals and Results

This section presents and discusses our main experiments. First, we show the performance of different algorithms and compare them to KaHIP and METIS. Secondly, we compare our novel V-cycle with the normal V-cycle and thirdly, we compare the performance of our novel V-cycle with single-level refinement. In all experiments, all partitions have a maximum imbalance of 3%.

5.3.1 METIS, KaHIP and our Algorithms

In this experiment, we compare the performance of KaHIP, Metis and our own algorithms. We partition the graphs (*uk-2002* from Table 5.1 is not included) with each algorithm into $k = 2, 4, 8, 16, 32$ blocks and compare the total communication volumes of the resulting partitions. This experiment was executed on Machine A. The algorithms are as follows.

KaHIP(Eco), KaHIP(Strong) These numbers were obtained by partitioning the graphs using KaHIP without any modification. In particular, these partitions are optimized for edge cut rather than total communication volume. `KaHIP(Eco)` uses KaHIP's `eco` or `ecosocial` preconfiguration and `KaHIP(Strong)` uses its `strong` or `strongsocial` preconfiguration.

METIS These numbers come from METIS in version 5.1.0. METIS was executed with the `-objtype=vol` argument, i.e. the partitions were optimized for total communication volume rather than edge cut. We ran METIS 10 times and took the median of those runs.

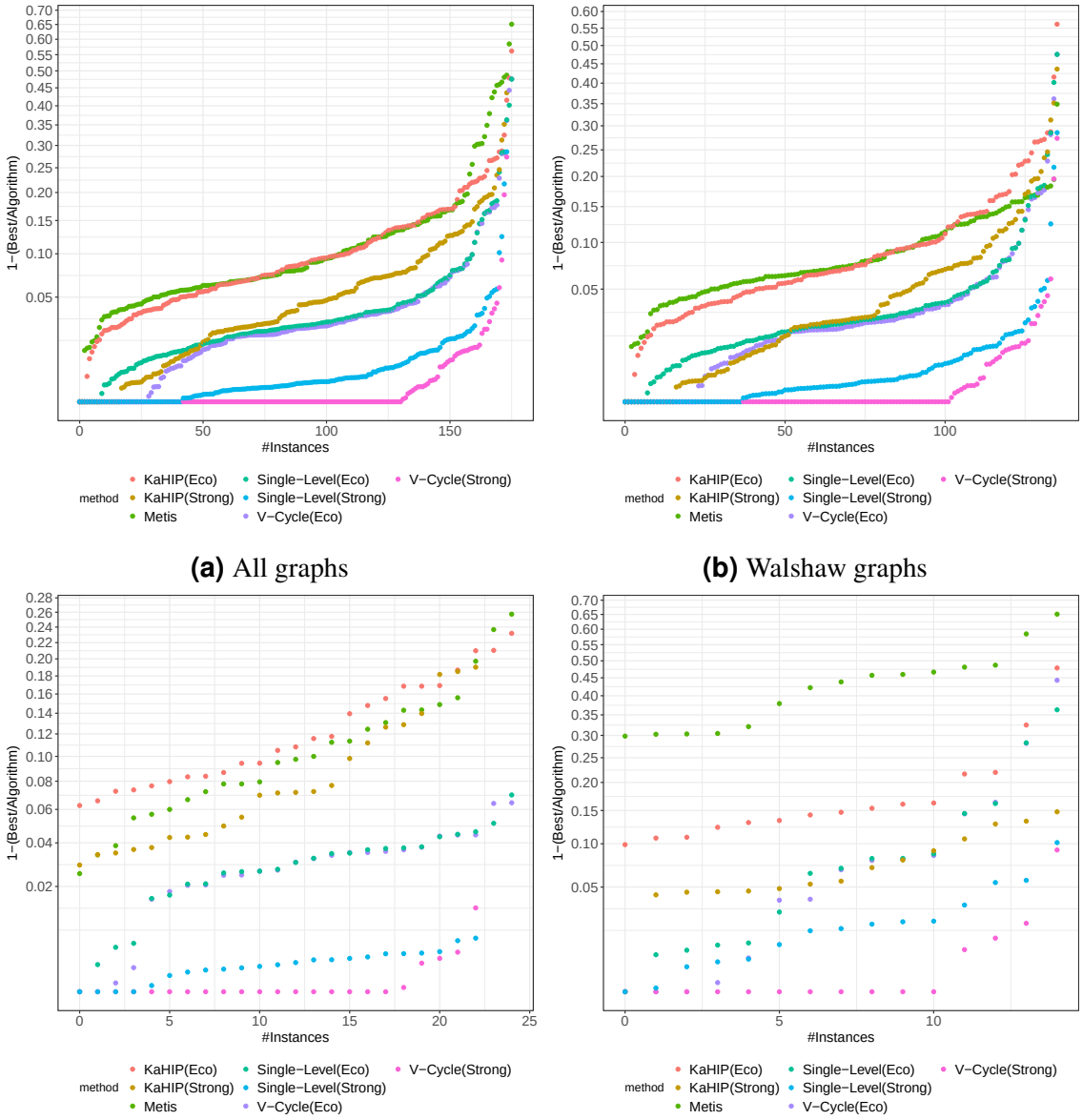
The other experiments are based on our own algorithms. Each of them starts by obtaining a partition by KaHIP. Each time, the suffix indicates whether we use `KaHIP(Eco)` or `KaHIP(Strong)` in this first step.

V-Cycle In these experiments, we refine the initial partition further by using FM with at most 5 rounds and our novel V-cycle. We repeat this configuration 7 times. Then, we use Multi-try FM with our novel V-cycle to further refine the previously obtained partition. Again, we execute 7 cycles with this configuration.

Single-Level These experiments also refine the initial partition further by using FM, but this time, we only use it single-level. Afterwards, we run Multi-try FM single-level. We run both refinements at most 7 times.

The performance of each algorithm is depicted in Figure 5.1. Numeric values are shown in Table 5.3. The raw numbers are available in Appendix B.1 and Appendix B.2. We observe several points.

First, we notice that KaHIP(Strong) outperforms METIS on almost every instance. This does not come with much of a surprise, considering that KaHIP(Strong) achieves much better results than METIS if the objective to minimize is edge cut. Secondly, we can improve the result of KaHIP by up to 29,89% or 2,90% on average. Against METIS, our improvement ranges up to 65,09% and on average 7,59%. We achieve the best results on medium to large sized social graphs. Thirdly, we notice that the performance difference between our novel V-cycle and single-level refinement is virtually absent, with a median improvement of 0,09% (`V-Cycle(Strong)` to `Single-Level(Strong)`). This motivates us to do further experiments in Section 5.3.4.



(a) All graphs

(b) Walshaw graphs

(c) Citation network graphs

(d) Web graphs

Figure 5.1: Performance plots that compare our algorithms to KaHIP and METIS.

Category	Min	0.25-Quantile	Median	0.75-Quantile	Max
Improvement: METIS to V-Cycle(Strong)					
All	−38,04%	5,18%	7,59%	12,44%	65,09%
Walshaw	−38,04%	4,80%	6,99%	11,15%	34,90%
Citation	2,38%	6,00%	9,49%	14,34%	25,74%
Web	28,93%	30,34%	43,37%	48,10%	65,09%
Improvement: KaHIP(Strong) to V-Cycle(Strong)					
All	0,00%	0,75%	2,90%	7,01%	29,89%
Walshaw	0,00%	0,30%	2,48%	6,59%	29,89%
Citation	2,90%	4,29%	7,10%	12,64%	18,86%
Web	0,00%	4,57%	5,59%	7,92%	13,29%
Improvement: V-Cycle(Eco) to V-Cycle(Strong)					
All	−51,44%	0,43%	2,19%	4,17%	47,54%
Walshaw	−51,44%	0,25%	2,04%	3,73%	47,54%
Citation	−0,13%	2,06%	3,09%	3,77%	6,45%
Web	−10,06%	−0,82%	6,79%	14,45%	44,29%

Table 5.3: Improvement of our algorithms over KaHIP and METIS and the influence of a better input partition.

Category	Min	0.25-Quantile	Median	0.75-Quantile	Max
All	0,00%	0,29%	0,84%	1,62%	16,73%
Walshaw	0,00%	0,18%	0,69%	1,38%	16,73%
Citation	0,41%	0,77%	0,96%	1,14%	3,33%
Web	0,45%	2,39%	3,46%	4,45%	7,08%

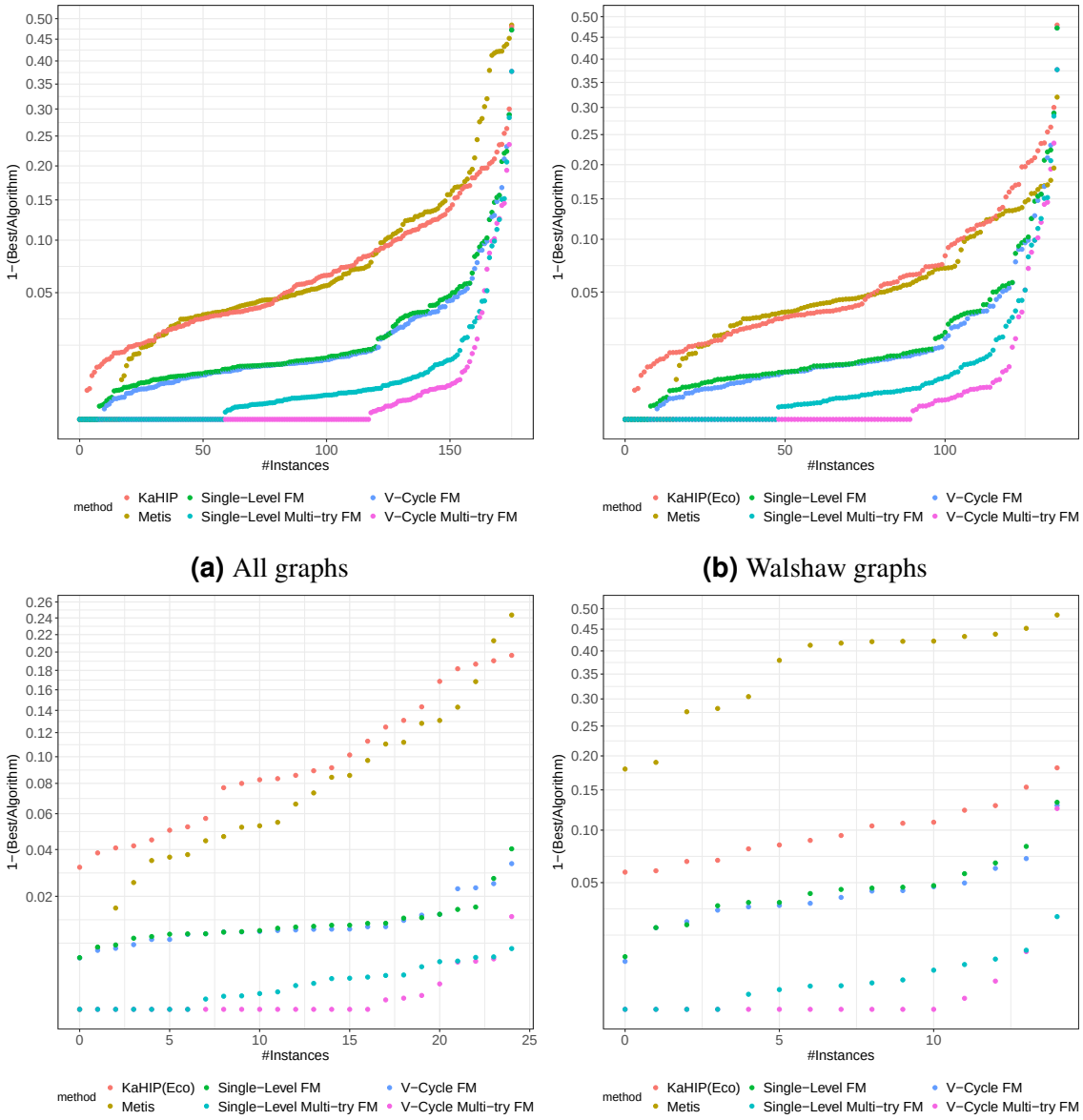
Table 5.4: Improvement achieved by Multi-try FM for all graphs and for each category.

5.3.2 FM and Multi-try FM

We also test the performance gain of Multi-try FM in a separate experiment, depicted in Figure 5.2 and Table 5.4. The labels describe the following algorithms.

FM This algorithm executes novel V-cycles with FM. We execute 7 cycles, each with at most 5 FM rounds. As always, we take a partition found by KaHIP as input partition. KaHIP is run with the preconfiguration `eco(social)`.

Multi-try FM This algorithm takes the resulting partition of FM as input and tries to further improve it by executing Multi-try FM with our novel V-cycle. Again, we execute 7 cycles, each with at most 5 Multi-try FM rounds.



(a) All graphs

(b) Walshaw graphs

(c) Citation network graphs

(d) Web graphs

Figure 5.2: Performance plots that show the effect of Multi-try FM.

5.3.3 Normal V-Cycle and Novel V-Cycle

We only use citation network graphs and web graphs in this experiment. Moreover, we partition the graphs into $k = 4, 8, 16, 32$ blocks, except for `uk-2004`, which we only partition into $k = 4$ blocks. This experiment was executed on Machine B.

We use the following configurations.

KaHIP These numbers were obtained from KaHIP using the `ecosocial` preconfiguration. We use these partitions as initial partitions for the other algorithms.

First V-cycle, 7nth V-cycle We execute 7 normal V-cycles with FM (with up to 5 rounds) as local search on the input partition. The numbers labeled *First V-cycle* are the result of the first v-cycle whereas the numbers labeled *7nth V-cycle* are the results after the last V-cycle.

Novel V-cycle For comparison, we include the performance of our novel V-cycle on the same instances. These numbers are from Section 5.3.1, labeled **V-cycle**.

The results are depicted in Figure 5.3. We observe that normal V-cycles do not always lead to improvements in the partition's quality. In fact, refining KaHIP's partition with only one normal V-cycle worsens the partition in about half of all test instances. More V-cycles seem to yield an improvement, but they never reach the performance of our novel V-cycle or single-level refinement.

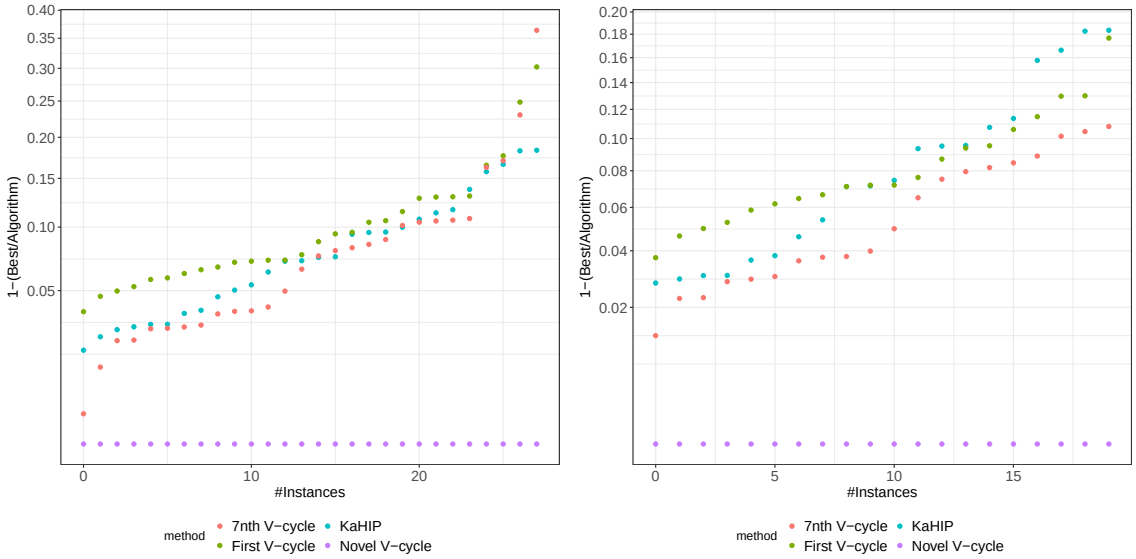
5.3.4 Novel V-Cycles and Single-Level Refinement

Motivated by the observation that our novel V-cycle and single-refinement perform almost the same, we do some more experiments dedicated to this phenomenon. This experiments was executed on Machine B. We use the citation network graphs and web graphs in this experiment, and partition them into $k = 4, 8, 16, 32$ blocks. Except for `uk-2004`, which we only partition into $k = 4$ blocks. We use KaHIP with preconfiguration `fastsocial` to obtain an initial partition.

We coarsen the graph using our novel V-cycle and execute up to 10 FM rounds on the coarsest level of the graph hierarchy. The total communication volume of this step is labeled *Coarsest* in Figures 5.4a and 5.4b. After uncoarsening, we execute additional 1 (Figure 5.4a) or 4 (Figure 5.4b) rounds of FM on the finest level. These results are labeled *Both*. We compare these results with the data set that is labeled *Finest*. To obtain this, we simply execute 1 (Figure 5.4a) or 4 (Figure 5.4b) rounds of FM on the finest level, independent of the other experiment.

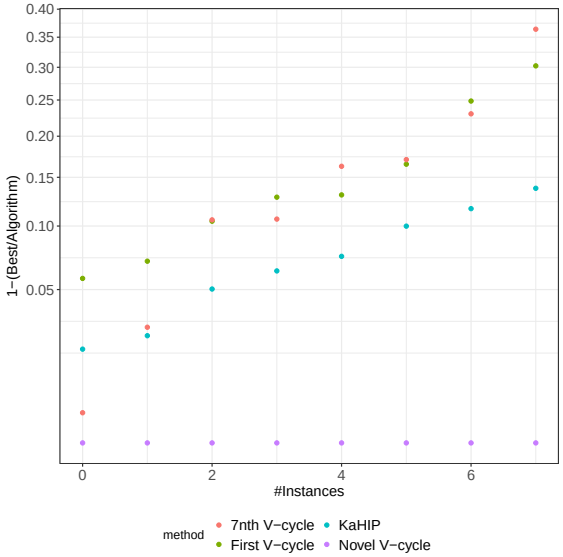
We observe that while the work that is done on the coarsest level benefits the result in Figure 5.4a, the benefit is below 1% in Figure 5.4b and vanishes if replace the work that is done on the coarsest level by another round of FM on the finest level (Figure 5.5).

A closer look reveals that the number of weighted nodes that get moved by the FM algorithm on a coarser level is below one tenth of a percent on all web graphs for $k = 4, 8$. This indi-



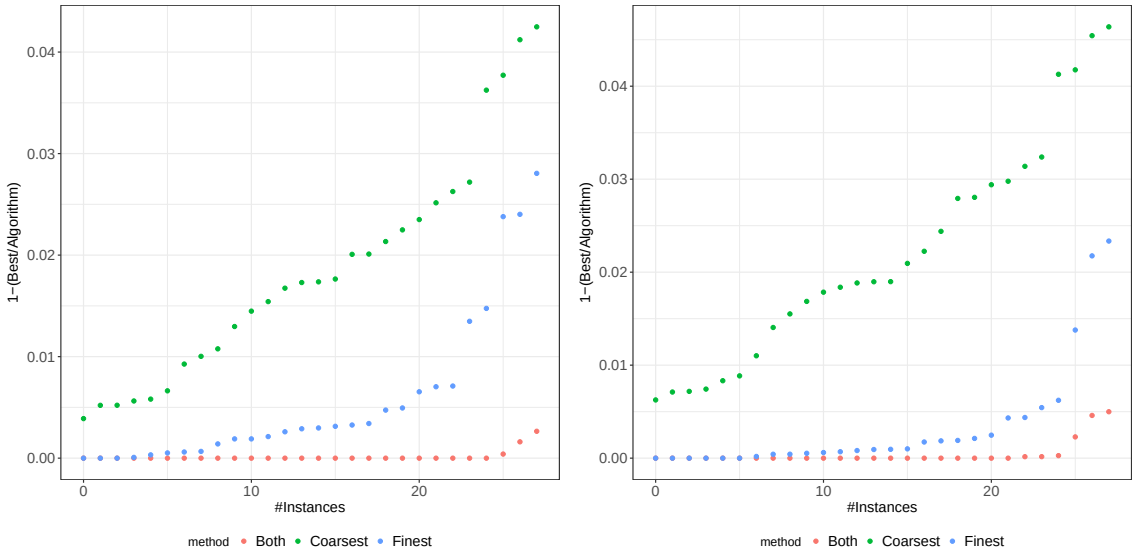
(a) All graphs

(b) Citation network graphs



(c) Web graphs

Figure 5.3: Comparison between KaHIP's result, normal V-cycles and our novel V-cycle.



(a) 10 rounds of FM on the coarsest level, then 1 round on the finest vs single-level

(b) 10 rounds on the coarsest level, 4 on the finest vs 4 rounds single-level

Figure 5.4: Shows the performance difference between V-cycle and single-level refinement. The advantage of our novel V-cycle versus single-level refinement is below 1% on almost every instance.

cates that coarsening the graph limits the refinement to unweighted nodes, i.e. uncontracted nodes that are also present on the input graph. In terms of quality, an improvement seems therefore to be impossible. Furthermore, the time that it takes to construct the coarser graph is several orders of magnitude bigger than the running time of the FM algorithm on all tested graph instances. Thus, it seems doubtful that an improvement by our V-cycle is possible in regards to partition quality or running time.

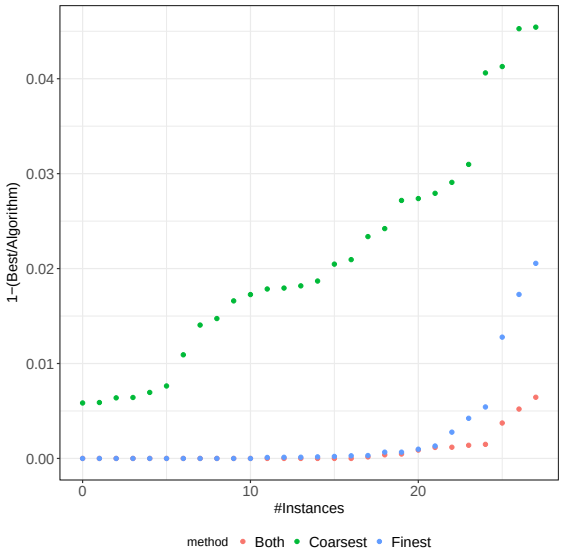


Figure 5.5: Shows the difference between doing 10 FM rounds on the coarsest and 3 FM rounds on the finest level versus only 4 rounds FM on the finest level. The labels are analogous to Figure 5.4.

6 Discussion

6.1 Conclusion

We observe that KaHIP produces partitions with lower total communication volume than METIS on almost every instance that we've tested, even though KaHIP only tries to minimize the edge cut of the partition. We were able to further minimize the total communication volume of partitions found by KaHIP through FM and Multi-try FM.

However, we are unable to observe an advantage of multi-level V-cycles over simple local search on the input graph. After coarsening the graph, the objective is overestimated and an improvement on a coarser level might lead to a worsening on the finest level of the graph hierarchy. Thus, V-cycles in the way they are used to further minimize the edge cut of a partition seem to be unsuitable for the total communication volume objective if no exact gain values are used.

Our own variant of a V-cycle is specially designed to prevent a worsening of the total communication volume but seems to be too restrictive. On coarser levels, experiments show that movements are practically restricted to the uncontracted boundary. Thus, it can be seen as a restricted local search on the finest level of the graph. In terms of running time, the contraction of the graph dominates the V-cycle. In conclusion, local search that is only executed on the finest level of the graph is faster and produces results of equal quality than our novel V-cycle.

6.2 Future Work

The main drawback of our algorithm, so it seems, is that the objective isn't preserved throughout the graph hierarchy and thus, improvements on a coarser level might not correspond to an improvement on the finest level. We believe that an improvement is possible by using precise gain values on coarser levels. In other words, one should calculate the objective such that the gain values of coarser nodes are the same as the change in the objective on the finest level of the graph hierarchy when the movements are made. To this end, one has to develop a way to calculate these precise gain values efficiently.

Another field of future work is to implement further objectives within KaHIP. We designed the implementation of the local search algorithms that we use such that the calculation of gain values can exchanged easily.

A Implementation Details

Our compiled software must be called with the following arguments,

```
$ ./app [--seed=SEED] CONFIG GRAPH K
```

where `--seed` is optional and specifies the random seed, `CONFIG` is the path to a configuration file (see below), `GRAPH` is the path to the graph that should be partitioned and `K` is the number of blocks.

The configuration file specifies and configures the steps that should be executed to partition the graph. The file uses the well-known INI file format[1] and is structured as follows.

The sections are numbered consecutively starting at 0 and represent processing steps. The first section is responsible for calculating the initial partition. All other sections get the resulting partition from the preceding section as input partition if not specified otherwise. The entries that a section configure are listed and described in Table A.1. If a section omits an entry, the value from the previous section is implicitly copied. If an entry is never specified, the default value is used.

Upon completion, a file containing some metrics of the resulting partitions of each step is written to the hard disk.

[1] See `https://technet.microsoft.com/en-us/library/cc731332.aspx`

Key	Possible Values	Description
Preconfiguration	`standard`, `fast`, `eco`, `strong`, `fastsocial`, `ecosocial`, `strongsocial`	Preconfiguration that is passed to KaHIP. Default: `standard`
Refinement	`KaHIP`, `fm`, `multitry`	If `KaHIP`, use KaHIP to obtain a partition (edge cut). Otherwise, use FM or Multi-try FM as specified. Default: none, error if omitted
Objective	`edgecut`, `totalcommvol`, `null`	Objective for the FM or Multi-try FM algorithm. Has no effect if `Refinement` is set to `KaHIP`. Default: `null`
FmStopRule	`Simple`, `Adaptive`	Stopping criteria for the FM or Multi-try FM algorithm. Default: `Simple`
NumberOfFmRounds	$\mathbb{N}_0$	Maximal number of rounds that the FM or Multi-try FM algorithm should execute. Default: 1
NumberOfVCycles	$\mathbb{N}_0$	Number of V-Cycles that should be executed with this configuration. Default: 1
UseModifiedVCycle	`false`, `true`	Whether our modified V-Cycle should be used or not. Default: `false`
OnlyRefineFinest	`false`, `true`	Whether V-Cycles should be executed at all; if set to `true`, refinement is only done on the finest level of the graph. Default: `false`
Reset	$\mathbb{N}_0 \cup \{-1\}$	If set, the resulting partition from the specified section is used as input partition rather than the resulting partition from the previous section. Default: -1 (no effect)

Table A.1: Structure of the configuration file.

B Detailed Experimental Results

All numbers are medians of 8 runs with different seeds. The prozentual improvements of *V-cycle* and *Finest* is in regards to the left-most *Vol* column. See Section 5.3.1 for evaluation.

B.1 Eco

		V-cycle			Finest			
Graph	**Vol**	**FM**	**MT**	**Vol**	**FM**	**MT**	**Vol**	**METIS**
$k = 2$								
eu-2005	18 278	5,98	0,51	**16 653**	5,98	0,4	16 698	28 762
in-2004	2 763	5,19	1,48	**2 503**	4,59	1,7	2 515	4 853
cnr-2000	468	5,42	0,36	438	4,77	1,06	**383**	699
$k = 4$								
eu-2005	55 896	7,88	3,29	49 822	6,8	3,32	**49 803**	86 186
in-2004	7 491	4,12	2,94	**6 890**	4,08	2,67	6 902	11 830
cnr-2000	2 234	3,39	5,99	2 021	2,46	5,83	**2 000**	2 785
$k = 8$								
eu-2005	113 063	6,77	2,33	**99 103**	5,99	2,42	100 192	159 648
in-2004	12 279	3,77	2,29	**11 443**	3,81	2,66	11 468	20 179
cnr-2000	4 742	2,72	2,61	4 425	2,41	2,4	**4 414**	5 449
$k = 16$								
eu-2005	210 818	8,02	3,32	**188 085**	7,74	3,22	189 269	270 445
in-2004	18 288	3,84	1,63	**17 216**	3,84	1,55	17 237	29 317
cnr-2000	7 700	4,47	2,21	**7 052**	4,41	2,43	7 057	8 598
$k = 32$								
eu-2005	450 310	8,82	4,39	**392 093**	8,35	3,97	402 878	541 359
in-2004	24 579	3,66	1,67	**23 109**	3,62	1,52	23 148	39 962
cnr-2000	36 194	7,07	3,44	**30 631**	6,55	4,03	30 874	54 533

Table B.1: Web graphs with `ecosocial` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 2$								
coPapersCiteseer	104 776	13,42	1,56	90 071	14,25	1,32	**89 756**	98 173
citationCiteseer	31 025	8,11	0,62	28 442	8,06	0,75	**28 438**	29 173
coPapersDBLP	204 070	19,19	0,92	167 477	21,15	0,92	**165 219**	166 205
coAuthorsDBLP	41 659	4,79	1,53	38 606	4,83	1,64	38 593	**38 452**
coAuthorsCiteseer	21 504	4,73	2,56	**19 586**	4,4	2,76	19 700	19 906
$k = 4$								
coPapersCiteseer	196 897	12,87	1,09	171 316	13,73	0,88	**171 142**	192 353
citationCiteseer	64 669	8,21	0,71	**58 755**	8,25	0,75	58 778	61 639
coPapersDBLP	387 927	19,74	0,95	**317 421**	19,48	1,03	317 892	332 215
coAuthorsDBLP	77 460	4,42	1,01	**73 036**	4,41	1,15	73 071	75 661
coAuthorsCiteseer	38 169	3,61	1,26	**36 177**	3,61	1,21	36 244	38 193
$k = 8$								
coPapersCiteseer	275 645	12,47	0,93	241 313	12,77	0,66	**241 240**	290 055
citationCiteseer	100 148	8,1	0,59	**91 555**	8,03	0,63	91 581	95 124
coPapersDBLP	589 037	21,59	0,84	475 122	21,95	0,82	**473 401**	524 287
coAuthorsDBLP	110 388	3,97	0,66	**104 838**	4,12	0,63	105 027	110 917
coAuthorsCiteseer	55 155	3,05	0,77	**52 844**	3,28	0,67	52 900	57 712
$k = 16$								
coPapersCiteseer	344 682	11,15	0,79	**305 850**	10,75	0,69	307 175	388 575
citationCiteseer	141 179	7,35	0,87	**129 517**	7,2	1,03	129 632	134 394
coPapersDBLP	777 186	20,49	0,81	**632 067**	19,95	0,85	634 326	737 583
coAuthorsDBLP	140 347	3,54	0,55	**134 034**	3,52	0,64	134 253	143 508
coAuthorsCiteseer	68 990	3,0	0,48	**66 346**	3,08	0,4	66 447	74 691
$k = 32$								
coPapersCiteseer	402 869	9,67	0,76	**362 021**	9,16	0,74	363 054	478 638
citationCiteseer	189 165	6,88	0,82	**174 041**	6,99	0,98	174 088	183 579
coPapersDBLP	952 567	17,72	0,98	**792 050**	16,87	1,45	795 411	911 204
coAuthorsDBLP	169 676	3,4	0,47	162 780	3,45	0,5	**162 748**	175 635
coAuthorsCiteseer	80 641	2,69	0,34	**78 091**	2,76	0,31	78 104	89 567

Table B.2: Citation Network graphs with `ecosocial` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

		V-cycle			Finest			
Graph	**Vol**	**FM**	**MT**	**Vol**	**FM**	**MT**	**Vol**	**METIS**
$k = 2$								
3elt	91	0,0	0,0	**88**	0,0	0,0	**88**	95
auto	5 193	2,51	0,91	**4 993**	2,48	0,62	5 004	5 325
data	132	0,0	0,0	132	0,0	0,0	132	**130**
add32	16	0,0	0,0	16	0,0	0,0	16	**11**
cs4	779	10,5	0,0	700	10,63	0,0	697	**671**
fe-pwt	**240**	0,0	0,0	**240**	0,0	0,0	**240**	244
t60k	166	8,65	0,0	**145**	10,57	0,0	147	170
fe-tooth	2 590	1,11	0,16	**2 543**	1,11	0,09	2 558	2 588
fe-ocean	622	6,76	0,26	568	7,24	0,0	**567**	620
wing-nodal	933	3,83	0,0	894	3,95	0,1	**890**	925
wave	5 172	5,13	1,26	4 779	5,14	1,33	**4 778**	4 859
vibrobox	2 510	4,49	9,62	**2 080**	4,58	9,24	2 128	2 282
crack	**185**	0,0	0,0	**185**	0,0	0,0	**185**	202
598a	1 300	1,8	0,07	**1 273**	1,72	0,0	1 274	1 332
uk	37	0,0	0,0	**36**	0,0	0,0	**36**	42
cti	632	18,42	0,0	**532**	17,91	0,0	**532**	606
add20	308	23,57	2,47	234	34,39	0,65	227	**184**
bcsstk29	**360**	0,0	0,0	**360**	0,0	0,0	**360**	426
fe-body	273	2,64	0,65	263	3,18	0,17	263	**229**
fe-rotor	1 262	3,02	0,16	1 216	3,18	0,16	1 212	**1 188**
m14b	1 846	2,04	0,04	1 800	2,1	0,07	**1 797**	1 875
4elt	**139**	0,0	0,0	**139**	0,0	0,0	**139**	149
fe-4elt2	**132**	0,0	0,0	**132**	0,0	0,0	**132**	**132**
fe-sphere	**384**	0,0	0,0	**384**	0,0	0,0	**384**	429
144	3 373	1,81	0,35	3 296	1,79	0,24	**3 291**	3 442
bcsstk30	**527**	0,0	0,0	**527**	0,0	0,0	**527**	644
bcsstk31	708	1,14	0,0	**690**	1,14	0,0	695	887
bcsstk32	1 140	2,08	1,67	**1 082**	2,16	2,43	1 083	1 113
whitaker3	129	0,0	0,0	**128**	0,0	0,0	129	134
brack2	**461**	0,1	0,0	**461**	0,42	0,0	**461**	491
finan512	151	3,4	0,0	**146**	3,76	0,0	**146**	148
memplus	3 067	0,21	0,44	**3 029**	0,08	0,35	3 038	3 293
wing	1 584	9,94	0,0	**1 422**	10,54	0,03	**1 422**	1 459
bcsstk33	920	0,0	0,32	**908**	0,0	0,0	**908**	1 047

Table B.3: Walshaw graphs with `eco` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 4$								
3elt	214	0,0	0,0	**211**	0,0	0,0	**211**	227
auto	14 486	2,0	0,47	**14 099**	1,95	0,44	14 111	14 368
data	259	1,16	0,0	**243**	1,5	0,0	**243**	282
add32	73	1,33	9,49	61	1,33	6,62	61	**38**
cs4	2 020	13,98	0,38	**1 745**	13,49	0,04	1 762	1 774
fe-pwt	546	0,18	0,0	543	0,18	0,0	543	**515**
t60k	496	0,0	0,0	494	0,0	0,0	496	**478**
fe-tooth	4 488	0,55	0,07	**4 450**	0,56	0,03	**4 450**	4 989
fe-ocean	3 281	20,58	1,94	**2 511**	19,77	2,27	2 515	2 627
wing-nodal	2 087	2,85	0,13	**2 010**	2,84	0,06	2 016	2 098
wave	10 995	2,39	0,74	10 640	2,33	0,89	**10 624**	10 863
vibrobox	4 681	0,88	0,0	**4 628**	0,89	0,0	4 634	5 286
crack	380	0,0	0,0	**379**	0,0	0,0	**379**	420
598a	4 458	2,03	0,18	**4 343**	2,01	0,22	4 346	4 562
uk	89	1,11	0,0	**86**	1,11	0,0	**86**	101
cti	1 941	18,05	0,12	1 624	17,76	0,08	**1 620**	1 685
add20	590	0,0	5,4	526	0,0	0,09	531	**451**
bcsstk29	1 334	3,23	6,51	**1 164**	1,07	5,65	1 170	1 329
fe-body	588	2,77	1,47	560	2,89	1,18	562	**551**
fe-rotor	4 529	1,96	0,25	4 395	2,0	0,14	**4 394**	4 536
m14b	6 346	1,72	0,07	**6 204**	1,76	0,13	6 209	6 568
4elt	359	0,0	0,0	356	0,0	0,0	**355**	378
fe-4elt2	362	0,0	0,0	**357**	0,0	0,0	360	363
fe-sphere	832	0,4	0,3	**806**	0,48	0,36	808	851
144	8 761	2,18	0,52	**8 487**	2,15	0,58	8 491	8 529
bcsstk30	1 547	0,9	0,0	**1 521**	0,84	0,23	**1 521**	1 817
bcsstk31	1 987	2,29	0,95	1 889	1,95	0,74	**1 875**	2 139
bcsstk32	1 864	1,32	1,79	**1 769**	1,02	2,25	1 779	2 602
whitaker3	415	0,12	0,0	407	0,36	0,0	409	**405**
brack2	1 840	0,45	0,1	**1 829**	0,42	0,0	1 830	2 039
finan512	**292**	0,0	0,0	**292**	0,0	0,0	**292**	296
memplus	5 323	0,49	1,2	**5 185**	0,31	1,15	5 198	5 275
wing	3 329	10,5	0,27	**2 995**	10,17	0,18	3 010	3 110
bcsstk33	2 497	2,58	0,0	**2 434**	2,46	0,0	**2 434**	2 889

Table B.4: Walshaw graphs with `eco` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 8$								
3elt	382	0,0	0,0	**378**	0,0	0,0	**378**	392
auto	25 414	2,04	0,51	**24 697**	2,02	0,56	24 715	25 135
data	466	3,0	0,0	**439**	2,32	0,0	440	472
add32	130	0,0	0,0	119	0,0	0,4	127	**91**
cs4	3 110	11,73	0,17	2 767	11,89	0,05	**2 762**	2 787
fe-pwt	1 019	0,29	0,0	**1 016**	0,29	0,0	**1 016**	1 083
t60k	988	1,38	0,2	959	1,75	0,0	**956**	1 049
fe-tooth	7 988	1,18	0,3	**7 844**	1,12	0,26	7 849	8 189
fe-ocean	8 117	21,39	1,29	**6 442**	21,05	1,2	6 451	6 754
wing-nodal	3 343	3,4	0,29	3 220	3,38	0,27	**3 219**	3 382
wave	17 777	2,41	0,68	**17 250**	2,47	0,61	17 274	17 760
vibrobox	10 897	11,12	2,85	**9 235**	11,33	2,97	9 256	9 341
crack	719	0,19	0,0	715	0,2	0,13	**714**	765
598a	9 705	3,01	1,42	**9 224**	2,83	1,34	9 241	9 332
uk	186	2,96	0,0	**174**	3,04	0,0	176	201
cti	3 482	16,4	0,38	**2 893**	16,44	0,28	2 909	3 134
add20	1 048	3,8	0,78	968	2,06	0,38	984	**781**
bcsstk29	3 541	4,63	15,12	**2 753**	2,97	15,92	2 837	2 910
fe-body	993	2,72	1,99	937	2,83	1,5	**929**	955
fe-rotor	7 718	1,83	0,47	**7 463**	1,85	0,59	**7 463**	8 034
m14b	13 072	1,75	0,61	**12 720**	1,74	0,69	12 727	13 387
4elt	570	0,0	0,0	564	0,0	0,0	**563**	628
fe-4elt2	648	0,07	1,05	**636**	0,0	0,22	640	657
fe-sphere	1 298	2,68	0,4	**1 253**	1,38	0,35	1 258	1 296
144	14 043	2,3	0,84	**13 575**	2,24	0,87	13 594	14 159
bcsstk30	3 378	0,86	0,0	**3 330**	0,91	0,0	3 342	3 845
bcsstk31	3 946	2,59	2,43	**3 694**	2,31	1,76	3 773	4 109
bcsstk32	4 492	1,68	3,76	**4 201**	1,31	2,41	4 264	5 051
whitaker3	692	0,21	0,07	**686**	0,14	0,07	**686**	718
brack2	4 753	0,99	0,4	**4 662**	0,91	0,09	4 672	4 858
finan512	657	0,0	0,0	657	0,0	0,0	657	**592**
memplus	7 850	1,48	1,7	7 517	1,25	1,28	7 609	**6 989**
wing	5 228	10,37	0,22	4 701	10,32	0,28	**4 696**	4 888
bcsstk33	4 877	6,43	1,21	**4 419**	5,43	0,17	4 512	5 364

Table B.5: Walshaw graphs with `eco` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 16$								
3elt	632	0,45	0,08	**621**	0,87	0,07	**621**	650
auto	42 173	2,36	0,74	**40 814**	2,31	0,73	40 835	43 527
data	825	3,53	0,18	**778**	3,06	0,0	**778**	837
add32	193	0,0	0,0	192	0,0	0,0	193	**169**
cs4	4 497	12,33	0,3	**3 952**	12,3	0,11	3 964	4 045
fe-pwt	1 978	1,55	0,07	**1 924**	1,27	0,22	1 925	2 000
t60k	1 801	2,16	0,41	**1 731**	2,28	0,23	**1 731**	2 002
fe-tooth	12 268	1,46	0,31	11 988	1,53	0,46	**11 974**	12 397
fe-ocean	15 706	20,77	0,81	12 626	20,57	0,97	**12 619**	13 144
wing-nodal	5 274	3,18	0,41	**5 074**	3,0	0,24	5 089	5 338
wave	26 692	2,29	0,74	**25 769**	2,25	0,73	25 789	26 946
vibrobox	15 614	12,93	1,57	**13 450**	11,74	1,6	13 577	14 159
crack	1 214	0,93	0,2	**1 194**	0,7	0,04	1 199	1 238
598a	16 035	2,57	1,07	15 386	2,55	1,11	**15 376**	15 872
uk	348	4,06	0,29	324	4,55	0,6	**323**	337
cti	5 796	23,71	0,17	4 604	23,21	0,27	**4 572**	4 723
add20	1 686	5,37	0,48	1 495	3,94	0,09	1 513	**1 343**
bcsstk29	5 384	2,31	13,44	**4 468**	1,48	9,66	4 671	5 002
fe-body	1 796	3,38	2,15	1 674	3,32	2,11	**1 667**	1 715
fe-rotor	12 994	1,84	0,52	**12 626**	1,86	0,47	12 639	13 033
m14b	22 868	1,89	0,75	22 170	1,89	0,85	**22 158**	23 479
4elt	1 051	0,09	0,04	1 044	0,18	0,04	**1 041**	1 075
fe-4elt2	1 094	0,09	0,44	**1 079**	0,0	0,26	1 082	1 147
fe-sphere	1 961	2,46	0,59	1 880	2,79	0,67	**1 874**	1 962
144	21 692	2,25	0,91	**20 909**	2,18	0,91	20 967	22 046
bcsstk30	7 979	1,39	0,0	**7 839**	1,17	0,0	7 871	8 992
bcsstk31	6 730	2,23	3,24	6 391	1,91	2,95	**6 361**	7 447
bcsstk32	7 911	1,94	3,63	**7 332**	1,95	3,13	7 409	9 106
whitaker3	1 193	1,0	0,33	**1 168**	0,7	0,36	1 169	1 210
brack2	7 988	1,33	0,42	**7 809**	1,24	0,42	7 826	8 160
finan512	**1 168**	0,0	0,0	**1 168**	0,0	0,0	**1 168**	1 257
memplus	9 628	0,87	0,78	9 299	0,61	0,26	9 379	**8 969**
wing	8 271	11,78	0,31	7 307	11,84	0,39	**7 298**	7 464
bcsstk33	8 327	8,02	0,85	**7 531**	5,88	0,14	7 630	9 045

Table B.6: Walshaw graphs with `eco` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

		V-cycle			Finest			
Graph	**Vol**	**FM**	**MT**	**Vol**	**FM**	**MT**	**Vol**	**METIS**
$k = 32$								
3elt	1 097	0,72	0,0	1 081	0,85	0,27	**1 078**	1 115
auto	68 203	2,38	0,95	**65 776**	2,36	0,85	65 878	68 110
data	1 349	3,03	0,38	1 289	2,67	0,03	**1 286**	1 355
add32	374	0,28	0,0	360	0,0	0,12	363	**329**
cs4	6 246	12,12	0,15	**5 520**	11,75	0,09	5 532	5 722
fe-pwt	3 984	2,31	0,21	**3 841**	2,01	0,13	3 850	4 437
t60k	2 956	3,44	0,38	**2 814**	3,43	0,55	2 818	3 192
fe-tooth	17 803	1,67	0,51	**17 358**	1,67	0,52	17 370	18 190
fe-ocean	26 368	30,59	0,71	19 447	30,06	0,85	**19 425**	20 941
wing-nodal	7 909	3,13	0,28	**7 627**	2,99	0,14	7 635	7 910
wave	37 941	2,6	0,87	**36 500**	2,66	0,78	36 584	38 744
vibrobox	21 194	10,05	0,79	**18 836**	8,76	0,98	18 964	19 780
crack	1 894	0,82	0,12	**1 863**	0,86	0,07	1 866	1 904
598a	24 502	2,65	1,05	**23 528**	2,7	0,96	23 557	24 619
uk	581	4,13	0,36	**543**	3,43	0,26	544	575
cti	8 312	22,25	0,19	**6 675**	22,3	0,2	**6 675**	6 765
add20	2 454	0,27	0,34	2 414	0,22	0,03	2 432	**2 063**
bcsstk29	7 502	0,94	5,6	**6 819**	0,44	3,59	7 045	7 839
fe-body	2 769	4,03	2,48	**2 568**	3,64	2,45	2 573	2 760
fe-rotor	20 322	2,46	0,57	19 701	2,34	0,7	**19 638**	20 740
m14b	36 271	2,01	0,91	**35 010**	2,0	0,83	35 103	36 202
4elt	1 742	0,35	0,34	**1 718**	0,53	0,05	1 726	1 791
fe-4elt2	1 804	0,4	0,28	1 771	0,33	0,47	**1 769**	1 818
fe-sphere	2 844	2,57	0,56	**2 734**	2,47	0,43	2 736	2 863
144	32 401	2,22	0,97	**31 232**	2,19	0,98	31 292	32 761
bcsstk30	13 988	3,19	0,1	**13 426**	2,97	0,0	13 551	15 306
bcsstk31	11 798	3,73	3,88	**10 824**	2,77	3,92	10 900	12 535
bcsstk32	13 014	2,78	3,99	**12 073**	2,42	3,23	12 207	14 538
whitaker3	1 841	0,75	0,19	1 814	0,63	0,09	**1 813**	1 853
brack2	12 243	1,24	0,41	**12 007**	1,23	0,34	12 025	12 849
finan512	**2 336**	0,0	0,0	**2 336**	0,0	0,0	**2 336**	2 367
memplus	12 435	0,66	1,31	11 949	0,26	0,65	12 031	**11 709**
wing	11 939	11,8	0,4	10 548	11,82	0,37	**10 540**	10 937
bcsstk33	12 938	4,16	1,62	**11 981**	2,75	1,48	12 128	14 216

Table B.7: Walshaw graphs with `eco` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

B.2 Strong

Tables in this section use the `strong` or `strongsocial` preconfiguration.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 2$								
coPapersCiteseer	101 187	12,43	1,91	87 292	13,38	1,5	**87 045**	98 173
citationCiteseer	30 656	7,03	0,3	28 480	7,03	0,29	**28 471**	29 173
coPapersDBLP	192 367	19,06	1,12	158 773	20,45	0,94	**156 756**	166 205
coAuthorsDBLP	39 314	3,9	1,0	**37 151**	3,83	1,09	37 200	38 452
coAuthorsCiteseer	19 179	3,08	1,2	**18 323**	2,93	1,18	18 357	19 906
$k = 4$								
coPapersCiteseer	187 921	12,61	1,48	**163 718**	12,41	1,24	164 195	192 353
citationCiteseer	61 944	7,27	0,41	**57 186**	7,07	0,56	57 190	61 639
coPapersDBLP	378 255	20,51	1,12	306 935	21,1	1,07	**306 317**	332 215
coAuthorsDBLP	75 263	3,22	0,9	**71 533**	3,14	1,07	71 677	75 661
coAuthorsCiteseer	36 129	2,96	0,96	**34 570**	3,0	0,9	34 660	38 193
$k = 8$								
coPapersCiteseer	266 557	11,94	1,37	**232 868**	12,33	1,06	233 226	290 055
citationCiteseer	98 485	6,82	0,43	91 462	6,48	0,77	**91 459**	95 124
coPapersDBLP	568 802	19,67	1,17	**465 413**	19,03	1,13	466 672	524 287
coAuthorsDBLP	106 859	3,19	0,63	**102 270**	3,02	0,77	102 458	110 917
coAuthorsCiteseer	52 352	2,49	0,57	**50 532**	2,36	0,68	50 575	57 712
$k = 16$								
coPapersCiteseer	333 912	10,14	1,37	**296 583**	9,69	1,05	297 987	388 575
citationCiteseer	136 095	6,11	0,79	**126 330**	5,66	1,19	126 662	134 394
coAuthorsDBLP	134 197	2,75	0,68	**129 160**	2,23	1,0	129 443	143 508
coAuthorsCiteseer	66 229	2,33	0,61	**63 984**	2,31	0,53	64 043	74 691
$k = 32$								
coPapersCiteseer	394 168	8,62	1,3	**355 459**	8,48	1,19	357 313	478 638
citationCiteseer	184 196	5,78	0,92	**171 352**	5,53	1,08	171 529	183 579
coAuthorsDBLP	164 518	2,76	0,57	**158 509**	2,48	0,69	158 620	175 635
coAuthorsCiteseer	77 848	2,27	0,41	**75 592**	2,25	0,4	75 628	89 567

Table B.8: Citation Network graphs with `strongsocial` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 2$								
eu-2005	16 087	4,18	0,2	**15 342**	3,96	0,0	15 417	28 762
in-2004	2 629	3,66	1,48	**2 490**	3,48	0,0	2 543	4 853
cnr-2000	265	8,48	0,68	**244**	8,24	0,0	245	699
$k = 4$								
eu-2005	58 443	4,89	0,14	**54 833**	5,11	0,0	55 434	86 186
in-2004	6 802	4,13	1,43	**6 422**	3,8	0,07	6 533	11 830
cnr-2000	2 089	4,03	0,96	**1 942**	3,57	0,1	2 011	2 785
$k = 8$								
eu-2005	108 991	7,27	0,37	**100 421**	7,67	0,01	101 411	159 648
in-2004	10 942	3,62	0,69	**10 473**	3,72	0,31	10 503	20 179
cnr-2000	3 889	3,32	0,68	**3 700**	3,38	0,13	3 738	5 449
$k = 16$								
eu-2005	210 545	5,25	2,65	**192 193**	4,42	0,76	199 383	270 445
in-2004	16 598	4,08	0,54	**15 840**	3,65	0,62	15 841	29 317
cnr-2000	6 319	3,66	0,98	**6 033**	3,52	0,07	6 145	8 598
$k = 32$								
eu-2005	434 863	15,25	0,03	**377 090**	9,36	1,36	385 712	541 359
in-2004	**16 598**	0,0	0,0	**16 598**	0,0	0,0	**16 598**	39 962
cnr-2000	35 146	6,37	4,78	**30 881**	5,34	1,38	32 395	54 533

Table B.9: Web graphs with `strongsocial` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 2$								
auto	5 054	2,14	0,36	**4 921**	2,01	0,4	4 926	5 325
3elt	**88**	0,0	0,0	**88**	0,0	0,0	**88**	95
data	**122**	0,0	0,0	**122**	0,0	0,0	**122**	130
cs4	712	7,85	0,0	**653**	7,77	0,0	657	671
add32	**10**	0,0	0,0	**10**	0,0	0,0	**10**	11
fe-pwt	243	0,0	0,0	**240**	0,0	0,41	**240**	244
fe-tooth	2 486	0,24	0,0	**2 475**	0,28	0,0	2 477	2 588
t60k	146	0,0	0,0	**144**	0,0	0,0	**144**	170
fe-ocean	622	6,5	0,0	**566**	7,13	0,0	571	620
wing-nodal	918	2,28	0,11	895	2,23	0,0	**893**	925
wave	4 814	2,77	0,13	**4 654**	2,75	0,06	4 671	4 859
vibrobox	3 665	12,11	2,6	3 150	10,18	3,33	3 141	**2 282**
crack	**187**	0,0	0,0	**187**	0,0	0,0	**187**	202
598a	1 295	1,88	0,0	**1 268**	1,95	0,0	**1 268**	1 332
uk	**38**	0,0	0,0	**38**	0,0	0,0	**38**	42
cti	632	14,12	0,0	**532**	12,98	0,0	**532**	606
add20	187	0,82	0,0	**184**	1,09	0,0	**184**	**184**
bcsstk29	**360**	0,0	0,0	**360**	0,0	0,0	**360**	426
fe-body	232	2,72	1,12	**218**	3,14	0,22	220	229
fe-rotor	1 061	1,24	0,0	**1 045**	1,28	0,0	**1 045**	1 188
4elt	**138**	0,0	0,0	**138**	0,0	0,0	**138**	149
m14b	1 841	1,65	0,09	**1 804**	1,65	0,0	1 805	1 875
fe-4elt2	**132**	0,0	0,0	**132**	0,0	0,0	**132**	**132**
fe-sphere	**384**	0,0	0,0	**384**	0,0	0,0	**384**	429
144	3 352	2,24	0,18	**3 267**	2,29	0,17	**3 267**	3 442
bcsstk30	528	0,18	0,0	**527**	0,18	0,0	**527**	644
bcsstk31	820	3,14	0,31	**766**	3,14	0,12	792	887
bcsstk32	1 050	1,05	3,75	**977**	1,05	3,28	979	1 113
whitaker3	**127**	0,0	0,0	**127**	0,0	0,0	**127**	134
brack2	**460**	0,0	0,0	**460**	0,0	0,0	**460**	491
finan512	166	12,49	0,0	148	12,53	0,0	**147**	148
memplus	2 930	0,64	0,08	**2 904**	0,51	0,09	2 916	3 293
wing	1 562	7,46	0,0	**1 457**	7,51	0,0	1 458	1 459
bcsstk33	**908**	0,0	0,0	**908**	0,0	0,0	**908**	1 047

Table B.10: Walshaw graphs with `strong` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 4$								
auto	14 076	1,92	0,34	13 742	1,88	0,36	**13 741**	14 368
3elt	**209**	0,0	0,0	**209**	0,0	0,0	**209**	227
data	255	0,2	0,0	**249**	0,0	0,0	252	282
cs4	1 900	7,9	0,07	**1 749**	7,51	0,0	1 751	1 774
add32	**32**	0,0	0,0	**32**	0,0	0,0	**32**	38
fe-pwt	482	0,0	0,0	**480**	0,0	0,0	**480**	515
fe-tooth	4 530	0,59	0,02	**4 493**	0,56	0,0	4 499	4 989
t60k	**414**	0,0	0,0	**414**	0,0	0,0	**414**	478
fe-ocean	3 289	23,47	0,16	2 494	23,25	0,22	**2 480**	2 627
wing-nodal	2 066	2,15	0,06	**2 008**	2,12	0,06	2 013	2 098
wave	9 594	3,24	0,23	**9 240**	3,2	0,2	9 249	10 863
vibrobox	7 138	5,8	3,94	6 370	5,11	3,04	6 476	**5 286**
crack	**368**	0,0	0,0	**368**	0,0	0,0	**368**	420
598a	4 378	1,97	0,12	**4 280**	1,95	0,16	4 283	4 562
uk	**84**	0,0	0,0	**84**	0,0	0,0	**84**	101
cti	1 747	7,58	0,06	**1 615**	7,15	0,0	1 616	1 685
add20	484	3,75	0,0	**433**	2,82	0,11	438	451
bcsstk29	2 064	2,47	22,03	1 447	1,52	21,46	1 486	**1 329**
fe-body	501	2,16	1,58	469	1,9	0,92	**468**	551
fe-rotor	4 177	0,94	0,07	**4 127**	0,95	0,06	4 128	4 536
4elt	**335**	0,0	0,0	**335**	0,0	0,0	**335**	378
m14b	6 303	1,88	0,15	**6 156**	1,8	0,16	6 157	6 568
fe-4elt2	**356**	0,0	0,0	**356**	0,0	0,0	**356**	363
fe-sphere	796	0,0	0,12	**790**	0,12	0,12	791	851
144	8 208	1,82	0,39	**8 023**	1,79	0,33	8 024	8 529
bcsstk30	1 500	0,21	0,0	**1 489**	0,21	0,0	1 497	1 817
bcsstk31	1 946	1,84	1,22	**1 850**	1,63	1,41	1 861	2 139
bcsstk32	1 816	1,46	3,98	**1 694**	1,2	3,34	1 709	2 602
whitaker3	**382**	0,0	0,0	**382**	0,0	0,0	**382**	405
brack2	1 799	0,05	0,0	**1 796**	0,05	0,0	1 797	2 039
finan512	331	12,17	0,0	**296**	11,98	0,0	**296**	**296**
memplus	5 391	0,67	2,08	**5 226**	0,42	1,78	5 249	5 275
wing	3 239	7,7	0,01	**3 014**	7,68	0,03	3 015	3 110
bcsstk33	2 840	7,49	0,3	**2 534**	7,17	0,39	2 564	2 889

Table B.11: Walshaw graphs with `strong` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Graph	Vol	V-cycle FM	V-cycle MT	V-cycle Vol	Finest FM	Finest MT	Finest Vol	METIS
$k = 8$								
auto	24 941	1,96	0,39	**24 303**	1,93	0,37	24 326	25 135
3elt	368	0,0	0,0	**367**	0,0	0,0	368	392
data	443	2,99	0,24	**430**	2,0	0,0	431	472
cs4	2 887	7,62	0,07	**2 670**	7,47	0,03	2 672	2 787
add32	**76**	0,0	0,0	**76**	0,0	0,0	**76**	91
fe-pwt	1 022	0,19	0,0	**1 016**	0,19	0,0	**1 016**	1 083
fe-tooth	7 461	0,74	0,02	**7 398**	0,68	0,0	7 405	8 189
t60k	**919**	0,0	0,0	**919**	0,0	0,0	**919**	1 049
fe-ocean	7 829	17,32	0,56	**6 327**	17,26	0,34	6 396	6 754
wing-nodal	3 277	2,44	0,13	**3 187**	2,46	0,08	3 192	3 382
wave	16 322	2,25	0,15	**15 899**	2,26	0,15	15 902	17 760
vibrobox	10 005	4,1	2,79	**9 055**	3,81	2,04	9 283	9 341
crack	710	0,0	0,0	**709**	0,0	0,0	710	765
598a	8 968	2,04	0,25	8 762	1,95	0,29	**8 755**	9 332
uk	**168**	0,0	0,0	**168**	0,0	0,0	**168**	201
cti	3 481	18,77	0,32	**2 876**	18,65	0,28	2 907	3 134
add20	794	5,74	0,0	**747**	4,03	0,0	749	781
bcsstk29	4 006	2,73	15,43	2 927	1,7	15,44	3 145	**2 910**
fe-body	903	2,94	1,82	**856**	2,56	1,77	858	955
fe-rotor	7 426	1,13	0,08	7 327	1,09	0,1	**7 318**	8 034
4elt	560	0,0	0,0	**558**	0,0	0,0	559	628
m14b	12 885	1,51	0,24	**12 621**	1,47	0,31	12 622	13 387
fe-4elt2	**623**	0,0	0,0	**623**	0,0	0,0	**623**	657
fe-sphere	**1 222**	0,0	0,0	**1 222**	0,0	0,0	**1 222**	1 296
144	13 646	1,82	0,43	13 278	1,81	0,49	**13 274**	14 159
bcsstk30	3 376	0,6	0,0	**3 293**	0,6	0,0	3 324	3 845
bcsstk31	3 802	2,19	1,12	**3 619**	1,67	0,96	3 620	4 109
bcsstk32	4 550	2,85	2,67	**4 252**	2,6	2,61	4 283	5 051
whitaker3	677	0,0	0,0	**676**	0,0	0,0	**676**	718
brack2	4 572	0,7	0,07	**4 527**	0,7	0,06	4 532	4 858
finan512	669	12,32	0,0	**592**	12,42	0,0	**592**	**592**
memplus	7 082	0,47	1,85	**6 905**	0,18	1,42	6 945	6 989
wing	4 930	7,02	0,02	4 586	6,92	0,03	**4 584**	4 888
bcsstk33	4 891	6,37	0,95	**4 451**	3,05	0,7	4 500	5 364

Table B.12: Walshaw graphs with `strong` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

		V-cycle			Finest			
Graph	**Vol**	**FM**	**MT**	**Vol**	**FM**	**MT**	**Vol**	**METIS**
$k = 16$								
auto	41 131	2,07	0,43	**40 011**	2,03	0,46	40 022	43 527
3elt	611	0,0	0,0	**607**	0,0	0,0	608	650
data	783	3,08	0,25	**749**	1,83	0,0	751	837
cs4	4 141	6,99	0,06	**3 855**	6,98	0,06	3 856	4 045
add32	**138**	0,0	0,0	**138**	0,0	0,0	**138**	169
fe-pwt	1 945	0,53	0,0	1 924	0,69	0,0	**1 922**	2 000
fe-tooth	11 562	0,86	0,07	**11 439**	0,8	0,03	11 441	12 397
t60k	1 693	0,2	0,11	1 688	0,16	0,05	**1 687**	2 002
fe-ocean	14 947	24,93	0,45	**11 442**	23,58	0,39	11 489	13 144
wing-nodal	5 173	2,33	0,2	**5 008**	2,42	0,14	5 018	5 338
wave	25 079	2,02	0,29	**24 443**	2,0	0,21	24 461	26 946
vibrobox	15 688	9,56	2,77	**13 525**	8,48	2,83	13 814	14 159
crack	1 172	0,0	0,0	**1 170**	0,0	0,0	**1 170**	1 238
598a	15 252	2,02	0,27	**14 863**	1,97	0,37	14 873	15 872
uk	317	0,3	0,0	**313**	0,16	0,0	**313**	337
cti	5 567	21,9	0,14	**4 474**	20,57	0,13	4 482	4 723
add20	1 399	6,33	0,0	**1 238**	4,31	0,21	1 263	1 343
bcsstk29	5 381	0,95	11,99	**4 603**	0,98	9,42	4 699	5 002
fe-body	1 636	2,99	1,34	**1 545**	3,0	1,5	1 552	1 715
fe-rotor	12 359	1,35	0,15	**12 158**	1,3	0,09	12 171	13 033
4elt	1 002	0,0	0,0	**1 001**	0,0	0,0	**1 001**	1 075
m14b	21 823	1,64	0,4	**21 348**	1,62	0,39	21 356	23 479
fe-4elt2	**1 045**	0,0	0,0	**1 045**	0,0	0,0	**1 045**	1 147
fe-sphere	1 789	0,07	0,02	**1 778**	0,05	0,02	1 784	1 962
144	20 992	1,78	0,44	**20 442**	1,79	0,43	20 480	22 046
bcsstk30	8 212	2,83	0,09	**7 926**	2,65	0,01	7 985	8 992
bcsstk31	6 822	2,85	2,36	**6 359**	2,31	2,49	6 414	7 447
bcsstk32	7 909	2,52	3,11	**7 410**	2,15	2,8	7 450	9 106
whitaker3	1 164	0,0	0,0	**1 163**	0,0	0,0	1 164	1 210
brack2	7 677	0,62	0,07	**7 604**	0,6	0,02	7 606	8 160
finan512	1 344	12,2	0,0	1 184	12,24	0,03	**1 182**	1 257
memplus	8 892	1,22	1,33	**8 638**	1,4	0,81	8 683	8 969
wing	7 599	6,73	0,06	**7 103**	6,66	0,01	7 114	7 464
bcsstk33	8 619	4,78	1,84	**7 881**	3,39	1,42	7 995	9 045

Table B.13: Walshaw graphs with `strong` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

		V-cycle			Finest			
Graph	**Vol**	**FM**	**MT**	**Vol**	**FM**	**MT**	**Vol**	**METIS**
$k = 32$								
auto	65 560	1,99	0,53	**63 733**	1,97	0,48	63 809	68 110
3elt	1 062	0,0	0,0	**1 059**	0,0	0,0	**1 059**	1 115
data	1 324	3,0	0,23	**1 264**	3,65	0,14	1 265	1 355
cs4	5 788	6,78	0,03	**5 382**	6,69	0,03	5 392	5 722
add32	**298**	0,0	0,0	**298**	0,0	0,0	**298**	329
fe-pwt	4 180	3,78	0,16	3 984	3,7	0,09	**3 979**	4 437
fe-tooth	17 104	1,06	0,12	**16 886**	0,98	0,09	16 887	18 190
t60k	2 777	0,46	0,23	**2 748**	0,38	0,17	2 756	3 192
fe-ocean	24 366	21,51	0,49	19 297	21,42	0,61	**19 277**	20 941
wing-nodal	7 731	2,42	0,27	**7 513**	2,2	0,19	7 520	7 910
wave	36 628	1,93	0,3	**35 718**	1,9	0,27	35 750	38 744
vibrobox	21 109	7,86	1,34	**19 117**	7,15	0,86	19 175	19 780
crack	1 838	0,0	0,02	**1 835**	0,0	0,0	1 838	1 904
598a	23 426	2,0	0,31	**22 805**	1,99	0,41	22 814	24 619
uk	532	0,18	0,28	**530**	0,09	0,0	531	575
cti	7 970	21,67	0,21	**6 411**	20,56	0,08	6 459	6 765
add20	2 159	1,13	0,24	2 087	0,98	0,14	2 088	**2 063**
bcsstk29	7 386	0,56	5,36	**6 779**	0,52	4,0	6 914	7 839
fe-body	2 661	3,31	1,75	**2 515**	2,99	1,71	2 517	2 760
fe-rotor	19 622	1,6	0,24	**19 196**	1,52	0,27	19 207	20 740
4elt	1 676	0,02	0,02	**1 673**	0,0	0,0	**1 673**	1 791
m14b	34 600	1,53	0,45	**33 823**	1,52	0,42	33 842	36 202
fe-4elt2	1 738	0,02	0,02	**1 735**	0,0	0,0	1 738	1 818
fe-sphere	2 669	0,06	0,03	**2 663**	0,05	0,07	**2 663**	2 863
144	31 320	1,81	0,49	**30 584**	1,74	0,49	30 610	32 761
bcsstk30	13 913	3,08	0,1	**13 392**	2,38	0,1	13 486	15 306
bcsstk31	11 632	3,73	2,92	**10 774**	2,73	3,36	10 841	12 535
bcsstk32	12 579	1,88	3,11	11 913	1,78	2,48	**11 901**	14 538
whitaker3	1 796	0,13	0,05	**1 791**	0,07	0,0	1 794	1 853
brack2	11 568	0,7	0,04	**11 468**	0,71	0,02	11 476	12 849
finan512	2 670	12,75	0,02	**2 364**	12,75	0,04	2 365	2 367
memplus	11 414	0,59	0,53	**11 220**	0,45	0,34	11 280	11 709
wing	11 036	6,65	0,04	**10 332**	6,42	0,01	10 344	10 937
bcsstk33	13 153	3,43	2,05	**12 276**	2,95	1,6	12 477	14 216

Table B.14: Walshaw graphs with `strong` preconfiguration. From left to right: ⟨**Vol**⟩ TCV after KaHIP, ⟨**FM**⟩ %-improvement with V-cycle with FM, ⟨**MT**⟩ %-improvement with V-cycle with Multi-try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**FM**⟩ %-improvement with single-level with FM, ⟨**MT**⟩ %-improvement with single-level with Multi-Try FM to ⟨FM⟩, ⟨**Vol**⟩ resulting TCV, ⟨**METIS**⟩ TCV after METIS.

Bibliography

[1] David A Bader, Andrea Kappes, Henning Meyerhenke, Peter Sanders, Christian Schulz, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. *In Encyclopedia of Social Network Analysis and Mining*, pages 73 – 82, Springer, 2014.

[2] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. *CoRR*, abs/1311.3144, 2013.

[3] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In *Proceedings of the 10th International Conference on Experimental Algorithms*, SEA'11, pages 376–387, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] Daniel Delling, Andrew V. Goldberg, Ilya Razenshteyn, and Renato F. Werneck. Graph partitioning with natural cuts. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, pages 1135–1146, Washington, DC, USA, 2011. IEEE Computer Society.

[5] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[6] Bruce Hendrickson. *Graph partitioning and parallel solvers: Has the emperor no clothes?*, pages 218–225. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[7] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, New York, NY, USA, 1995. ACM.

[8] L. Hyafil and R. Rivest. Graph partitioning and constructing optimal decision trees are polynomial complete problems. IRIA-Laboria, Rocquencourt, France, 1973.

[9] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.

[10] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48(1):96–129, January 1998.

[11] Vitaly Osipov and Peter Sanders. n-level graph partitioning. *CoRR*, abs/1004.4024, 2010.

[12] Vitaly Osipov, Peter Sanders, and Christian Schulz. *Engineering Graph Partitioning Algorithms*, pages 18–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[13] Peter Sanders and Christian Schulz. High quality graph partitioning, 2013.

[14] Peter Sanders and Christian Schulz. Kahip v0.53 - karlsruhe high quality partitioning - user guide. *CoRR*, abs/1311.1714, 2013.

[15] Peter Sanders and Christian Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *LNCS*, pages 164–175. Springer, 2013.

[16] Chris Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research*, 131(1):325–372, Oct 2004.