Aggregative Coarsening for Multilevel Hypergraph Partitioning

Ruslan Shaydulin
School of Computing, Clemson University, Clemson, SC
rshaydu@g.clemson.edu
https://orcid.org/0000-0002-8657-2848

Ilya Safro
School of Computing, Clemson University, Clemson, SC
isafro@g.clemson.edu
https://orcid.org/0000-0001-6284-7408

Abstract

Algorithms for many hypergraph problems, including partitioning, utilize multilevel frameworks to achieve a good trade-off between the performance and the quality of results. In this paper we introduce two novel aggregative coarsening schemes and incorporate them within state-of-the-art hypergraph partitioner Zoltan. Our coarsening schemes are inspired by the algebraic multigrid and stable matching approaches. We demonstrate the effectiveness of the developed schemes as a part of multilevel hypergraph partitioning framework on a wide range of problems.

2012 ACM Subject Classification Mathematics of computing → Hypergraphs, Mathematics of computing → Graph algorithms, Mathematics of computing → Matchings and factors

Keywords and phrases hypergraph partitioning, multilevel algorithms, coarsening, matching, combinatorial scientific computing

Digital Object Identifier 10.4230/LIPIcs.SEA.2018.2

Related Version A full version of the paper is available at [51], `https://arxiv.org/abs/1802.09610`.

Funding This material is based upon work supported by the National Science Foundation under Grant No. 1522751.

1 Introduction

Hypergraph is a generalization of graph. Whereas in a graph each edge connects only two vertices, in a hypergraph a hyperedge can connect an arbitrary number of vertices. In many cases this allows hypergraph to better capture the underlying structure of the problem. In graph partitioning (GP), the goal is to split the vertex set of a graph into approximately even parts while minimizing the number of the edges in a cut [8]. Hypegraph partitioning problem (HGP) extends it to hypergraphs. Hypergraph partitioning has many applications in fields ranging from VLSI design [30] to parallel matrix multiplication [10] to classification [55] to optimizing distributed systems [33, 13], among others [16, 28].

Hypergraph partitioning is NP-hard [20] and relies on heuristics in practice. Many state-of-the-art graph and hypergraph partitioners utilize the multilevel approach [8]. In multilevel methods, the original problem is iteratively coarsened by creating a hierarchy of smaller problems, until it becomes small enough to be solved. Then the coarsest problem is solved and the solution is iteratively projected onto finer levels and refined. Multilevel algorithms

for HGP are typically generalizations of multilevel algorithms for graph partitioning, those in turn drawing inspiration from multigrid and other multiscale optimization techniques [6]. Hypergraph partitioning is less well-studied than graph partitioning [13] and there is a relative lack of advanced coarsening schemes compared to GP.

Our *main contribution* are two novel aggregative coarsening schemes for HGP that are inspired by algebraic multigrid and stable matching methods. We expand and build on the insights from an unfinished attempt to build a coarsening scheme for HGP using algebraic multigrid ideas, published in Sandia Labs Summer Reports [7]. At each coarsening level we split the set of vertices into the set of seeds and the set of non-seeds. Each seed becomes a center of an aggregate which will, in turn, create a node at the next, coarser level. Aggregation rules are established to specify which aggregate a non-seed joins. We investigate two approaches to establishing aggregation rules. One approach is algebraic multigrid-based generatlization of an inner-product matching similar to the matching scheme used in Zoltan[11] and PaToH[10]. Another approach is inspired by stable matching. Both approaches take advantage of the algebraic distance on hypergraphs when making coarsening decisions. Algebraic distance is a vertex similarity measure that extends simple measures such as hyperedge weights to better capture the structure of the hypergraphs [50]. While we outperform existing solvers on many instances, it is clear that final performance of HGP solvers heavily depends on the refinement. It is not the goal of this paper to outperform all existing HGP solvers. Instead, we would like to demonstrate that given similar uncoarsening schemes, the proposed coarsening schemes are at least as beneficial as traditional matching-based approaches.

2 Preliminaries

A hypergraph is an ordered pair of sets (V, E), where V is the set of vertices and E is the set of hyperedges. Each hyperedge $e \in E$ is a nonempty subset of V. In this paper we make use of a graph representation of a hypergraph called "star expansion". Star expansion graph (V', E') of a hypergraph (V, E) is an undirected bipartite graph with hypergraph vertices V on one side, hyperedges E on another and edges connecting hyperedges with the vertices they contain. Concretely, $V' = V \cup E$, $E' = \{(v, e) \mid e \in E, v \in e \subset V\}$. We will be referring to hyperedges as simply edges where it does not cause confusion. Both vertices and edges of the hypergraph are positively weighted. By $w(v)$ and $w(e)$, we denote weighting functions for nodes and edges, where $v \in V$ and $e \in E$. For both nodes and edges, a weight of zero practically means that corresponding nodes or edges do not exist (or do not affect the optimization and solution).

2.1 Hypergraph partitioning

In hypergraph k-partitioning the goal is to split the set of vertices V into k disjoint subsets $(V_1, \ldots, V_k)$ such that a metric on the cut is minimized subject to imbalance constraint. Here the cut is defined as the set of edges that span more than one partition, i.e.,

$$E_{\text{cut}} = \{e \in E \mid \exists i \neq j \text{ and } k \neq l \text{ for which } v_i, v_j \in e, v_i \in V_k \text{ and } v_j \in V_l\}. \tag{1}$$

There are multiple ways to define imbalance constraint. We will follow the definition used by the developers of state-of-the-art hypergraph partitioner Zoltan [15]. The imbalance is therefore defined as the ratio between the total weight of vertices in the largest partition

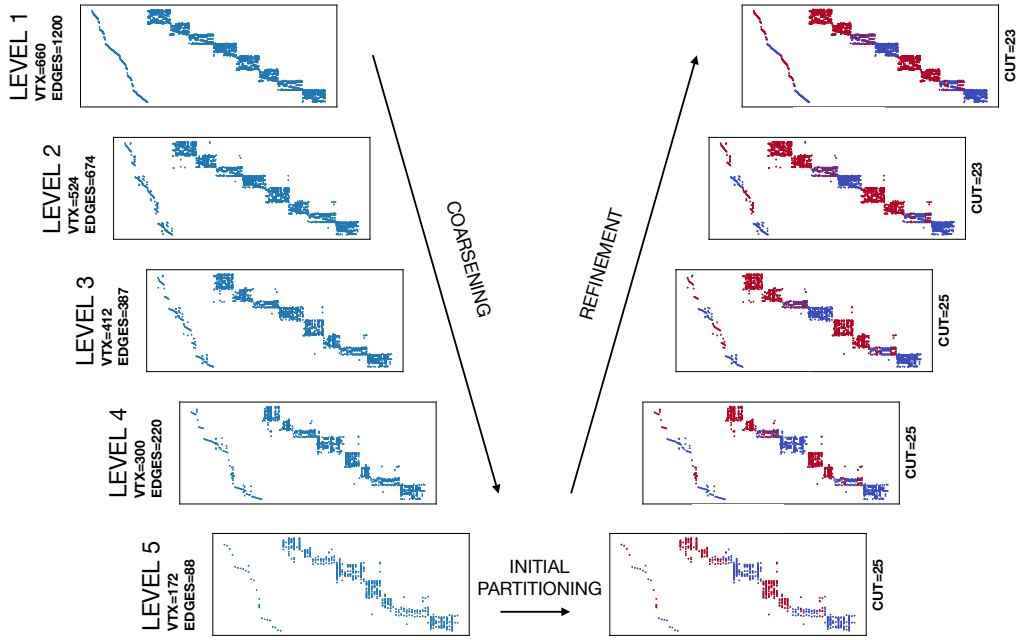

Figure 1 Multilevel partitioning of a hypergraph constructed from `LPnetlib/lp_scfxm2` matrix from SuiteSparse Matrix Collection [14] using row-net model: each column becomes a vertex and each row becomes a hyperedge. On the left side of the "V" the hypergraph (represented here as the sparsity pattern of the underlying matrix) is iteratively coarsened. At the bottom of the "V" the hypergraph is partitioned into two parts. This is represented by coloring the columns corresponding to vertices from one part into blue and another into red. On the right side of the "V" the hypergraph is uncoarsened and the partitioning is refined.

and the average sum of weights of vertices over all partitions. We define the imbalance as

$$imbal = \frac{\sum_{v \in V_{\max}} w(v)}{\frac{1}{k} \sum_{v \in V} w(v)}, \tag{2}$$

where V_{max} is the largest partition by weight (i.e., $\sum_{v \in V_{\max}} w(v) = \max_i \left(\sum_{v \in V_i} w(v) \right)$). Imbalance constraint imposes a limit on the value of $imbal$, e.g., imbalance of 5% means $imbal < 1.05$. The cut metric used in this paper is simply total weight of the cut edges, namely, $\sum_{e \in E_{\text{cut}}} w(e)$.

2.2 Multilevel method

The main objective of multilevel methods is to construct a hierarchy of problems, each approximating the original problem but with fewer degrees of freedom. This is achieved by introducing a chain of successive restrictions of the problem domain into low-dimensional or smaller-size domains (coarsening) and solving the coarse problems in them using local processing (uncoarsening) [41]. The coarsening-uncoarsening pipeline is often referred to as V-cycle. The multilevel frameworks combine solutions obtained by the local processing at different levels of coarseness into one global solution. Typically, for combinatorial optimization problems, the multilevel algorithms are suboptimal metaheuristics [53] that incorporate other methods as refinement at all levels of coarseness. Except partitioning, examples can be found in linear ordering [27, 43, 44, 46], clustering and community detection [42, 5], and traveling

salesman problems [54]. In (H)GP, these algorithms were initially introduced to speed up existing algorithms [4] but later proved to improve the quality of the solution [24, 31]. A multilevel hypergraph partitioning of a hypergraph constructed from `LPnetlib/lp_scfxm2` matrix is presented in Figure 1.

During the coarsening stage, for a hypergraph $H = (V, E)$ a hierarchy of decreasing in size hypergraphs $H^0 = (V^0, E^0), \ldots, H^l = (V^l, E^l)$ is constructed. Here l denotes the number of levels in multilevel hierarchy. During the initial partitioning stage, the coarsest hypergraph $H^l = (V^l, E^l)$ is partitioned. Finally, during the refinement stage the solution from coarser levels is projected onto finer ones and refined, typically using a local search heuristic.

2.3 Algebraic distance

Algebraic distance for hypergraphs is a relaxation-based vertex similarity measure [50]. It extends the algebraic distance for graphs [12, 41, 29] by taking into account the non-pairwise nature of the connections between vertices in a hypergraph. Algebraic distance improves on simpler similarity metrics, such as hyperedge weights, by incorporating information about more distant vertex neighborhood, thus better capturing vertex's place in the global structure of the hypergraph. Algebraic distance is inspired by iterative techniques for solving linear systems. An iterative method can be represented in a standard form:

$$x^{(i)} = Hx^{(i-1)} \qquad i = 1, 2, 3 \ldots \tag{3}$$

where H is the iteration matrix.

Similarly, algebraic distances are computed at each coarsening level using the following stationary iterative relaxation. Let $A \in R^{|E|\times|V|}$ be hypergraph incidence matrix, i.e., $A_{ij} = 1$ if the hyperedge i contains the vertex j. Let $S^v \in \mathbb{R}^{|V|\times|V|}$ and $S^h \in \mathbb{R}^{|E|\times|E|}$ be diagonal matrices such that

$$S^v_{jj} = w(v_j) \quad \text{and} \quad S^h_{ii} = \frac{w(h_i)}{|h_i|}, \tag{4}$$

where $|h_i|$ denotes the cardinality of the ith hyperedge. Denote

$$W = \begin{bmatrix} 0 & A^T S^h \\ AS^v & 0 \end{bmatrix} \tag{5}$$

and let D be the diagonal matrix with elements $D_{jj} = \sum_i W_{ij}$. Then the iterative step is defined as

$$x^{(i)} = \frac{1}{r-l}\Big[\underbrace{\omega D^{-1}Wx^{(i-1)} + (1-\omega)x^{(i-1)}}_{x^{*(i-1)}}\Big] - \frac{r+l}{2(r-l)}\mathbf{1} \tag{6}$$

where $\mathbf{1}$ is the vector of all ones, and r and l are the maximum and the minimum of the elements in $x^{*(i-1)}$, respectively. We can simplify the update formula as

$$x^{(i)} = \alpha^{(i-1)}Hx^{(i-1)} + \beta^{(i-1)}\mathbf{1}, \tag{7}$$

where

$$H = \omega D^{-1}W + (1-\omega)I, \quad \alpha^{(i-1)} = \frac{1}{r-l}, \quad \text{and} \quad \beta^{(i-1)} = -\frac{r+l}{2(r-l)}. \tag{8}$$

The iterative scheme is performed multiple times for different random initial values $x_0^{(0)}, \ldots x_R^{(0)}$ (called test vectors in algebraic multigrid [34]) . Then, the algebraic distance between vertices i and j is set to be the maximum over R random initializationsm namely, $\text{algdist}_{ij} = \max_R |x_i - x_j|$. For the detailed discussion of algebraic distances on hypergraphs and for convergence analysis of the described iterative scheme the reader is referred to [50].

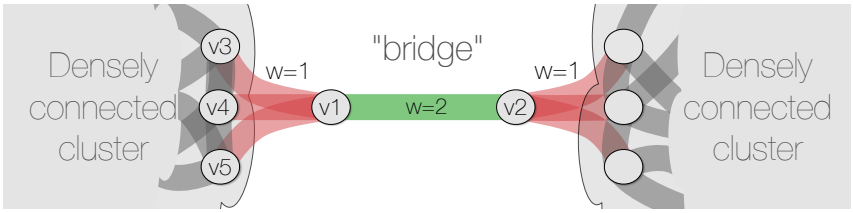

Figure 2 An example demonstrating the limitations of schemes based on edge weights. Here the "bridge" edge (green) connecting v1 and v2 has weight two and all other edges (red) have unitary weights. The best cut is achieved by cutting the green "bridge" between v1 and v2 and therefore matching v1 with one of the vertices on the left (v3,v4 or v5). However, matching schemes based on edge weights can match v1 with v2 instead and increase the cut.

3 Related work

Practical approaches to solving HGP typically rely on heuristics. Many have been developed over the years, but the most common approach is multilevel. It is implemented in many state-of-the-art hypergraph partitioners including but not limited to Zoltan [15], hMetis [30], KaHyPar [49] and PaToH [10]. In this section we will briefly describe the multilevel approach used by those state-of-the-art partitioners and discuss existing advanced coarsening schemes for hypergraphs. For a more detailed review of hypergraph partitioning, the reader is referred to [2, 3, 39, 52].

3.1 Brief overview of multilevel hypergraph partitioning

The HGP multilevel frameworks consist of three stages: coarsening, initial partitioning and uncoarsening with refinement. During the coarsening, the hypergraph is approximated via a series of decreasing in size hypergraphs. At each coarsening step, the next hypergraph is formed by matching a group of vertices into one, such that a set of vertices at level k becomes one vertex at level $k+1$. The decision as to which vertices to match is made based on similarity metrics such as inner product (i.e., the total weight of hyperedges connecting two vertices). However, simple metrics often result in a decision that ignores the structure of the hypergraph. Consider the example in Figure 2. It shows two densely connected clusters, separated by a "bridge" between vertices v1 and v2. Schemes that use naive similarity metrics like hyperedge weights might match v1 with v2, whereas an algorithm that considers larger neighborhoods to minimize a cut would prefer to match v1 with either v3, v4 or v5 instead. This example demonstrates the challenges of capturing the hypergraph structure by using only local information. In the refinement stage all the aforementioned state-of-the-art partitioners use a combination of Fiduccia-Mattheyses [18] or Kernighan-Lin [32] with the exception of KaHyPar, which uses a novel localized adaptive local search heuristic [1].

3.2 Aggregative coarsening

The standard approach to coarsening used in most state-of-the-art hypergraph partitioners is matching-based. Originally, this meant that at each level pairs of adjacent vertices are selected to become one vertex at the next level. This technique has later been extended to include non-pairwise matchings (i.e., more than two fine vertices can form a coarse vertex). One of the alternative approaches is aggregative coarsening inspired by algebraic multigrid. In aggregative coarsening, the set of vertices V is separated into disjoint sets of seeds and non-seeds, namely, C and F such that $F \cup C = V$. The non-seed vertices aggregate themselves

around the seeds (hence the name aggregative coarsening). The aggregation can be strict (F-vertices are not split) and weighted (F-vertices can be split between multiple seeds with vertex weight conservation). At the refinement stage, the partitioning decision (i.e., partition assignment) is interpolated from each seed to the non-seeds in its aggregate. This separation between seed and non-seeds helps to introduce additional guarantees. For example, on graphs Safro et al. [45] introduce the notion of strong connection and guarantee that each vertex in the graph is strongly connected to at least one seed. The weighted aggregation was initially introduced for several cut problems on graphs [48, 41, 46] including GP [47]. There was an unfinished attempt to extend this approach to hypergraphs. Buluç and Boman [7] describe several challenges in applying aggregative coarsening to hypergraphs, as well as propose two very similar coarsening schemes, strict and weighted. In this paper, we limit our discussion to strict aggregation.

In aggregative coarsening, two main questions have to be addressed: seed selection and aggregation of non-seeds around seeds. In the process of seed selection, Buluç and Boman [7] follow Safro et al. [45] in using the concept of *future volumes*. Future volume is a measure of how many vertices a seed can incorporate into itself (in other words, how large a vertex can grow). They propose computing future volumes on the star expansion of the hypergraph (thus limiting the complexity), then iteratively adding vertices with high future volumes to the set of seeds C until $|C|$ reaches a certain threshold. Aggregation rules are established on the star expansion of the hypergraph. Seeds and non-seeds select a constant number of adjacent hyperedges to "invade" based on the exclusive coarseness (a metric indicating how many seeds an hyperedge contains).

4 Two Aggregation Algorithms

Our algorithm combines the ideas of aggregative coarsening described in [45] and [7] with the algebraic distance [41, 50]. Aggregative coarsening is a two-step process, so we have to address both the seed selection and the rules of aggregation. At each coarsening level, a set of seeds is selected and each seed is assigned a set of non-seeds to form a cluster. The cluster at a given coarsening level becomes one vertex at the next level.

Both introduced schemes utilize algebraic distances by augmenting hyperedge weights with algebraic weights. We define the algebraic weight of hyperedge e as an inverse of the algebraic distance between two farthest apart vertices in e, i.e.,

$$\rho(e) = 1 \;/\; \max_{i,j \in e} \text{algdist}_{ij}. \tag{9}$$

4.1 Seed selection

For the seed selection we utilize two core concepts: *future volumes* and *strong connection*. The main goal is to construct a set of seeds C such that every vertex in the graph is *strongly connected* to C. We define *strong connection* as follows: the vertex $i \in F$ is *strongly connected* to C if the sum of algebraic weights of the edges connecting it to C is more than a certain fraction of the total algebraic weight of incident edges:

$$i \text{ is strongly connected to } C \quad \Longleftrightarrow \quad \frac{\Sigma_{j \in C} \rho(e_{ij})}{\Sigma_j \rho(e_{ij})} > Q, \tag{10}$$

where Q is a parameter (in our experiments $Q = 0.5$). The *future volume* of a vertex is a measure of how large an aggregate seeded by it can grow. Intuitively, we want to add the vertices with very high volume (or the ones that might become centers of the aggregates of

Listing 1 Seed selection.

```
for i in V:
        fv[i] = w[i] + Σ_j w[j](w[e_ij] / Σ_k w[e_jk])
for i in V:
        if fv[i] > mean(fv) + 2 * stdev(fv):
        C.insert(i)
    else:
        F.remove(i)
for i in F:
        fv[i] = w[i] + Σ_{j∈F} w[j](w[e_ij] / Σ_{k∈F} w[e_jk])

for i in sort_indices(fv):
        if Σ_{j∈C} w[e_ij] / Σ_j w[e_ij] < Q:
                C.insert(i)
                F.remove(i)
```

very high volume) to the set of seeds. *Future volume* of a vertex is defined as follows (note that here we use the hyperedge weights w and not the algebraic weights ρ):

$$fv(i) = w(i) + \Sigma_j w(j) \frac{w(e_{ij})}{\Sigma_k w(e_{jk})}. \tag{11}$$

We begin the construction of set C by computing future volumes for all vertices. Then, we initialize C with vertices with large future volumes (if mean future volume is m_{fv} and standard deviation of the distribution of future volumes is σ_{fv}, then $i \in C \iff fv(i) > m_{fv} + 2\sigma_{fv}$) and initialize F with all other vertices, such that $F \cup C = V$. After that the future volumes of vertices in F are recomputed, only taking into account connections with other vertices in F (i.e., in Equation (11) assume $w(e_{ij}) = 0$ if $j \in C$ or $i \in C$). Finally, vertices in F are visited in order of decreasing future volume and added to the set C if they are not strongly connected to C. Note that at the end of this process each vertex in V is strongly connected to the set C and $F \cup C = V$. Pseudocode for this procedure is presented in Listing 1.

4.2 Aggregation

We investigate two approaches to establishing the rules of aggregation. First approach is a scheme similar to inner-product matching used in Zoltan[11] and PaToH[10] but applied in algebraic multigrid setting. Second approach consists of computing a stable assignment [23] between vertices of C and F. Both approaches take advantage of algebraic distances as a similarity measure when establishing aggregation rules.

Inner-product aggregation proceeds by visiting the non-seed vertices in the random order. For each unmatched vertex $v \in F$, a neighboring seed $u \in C$ with the highest inner product is selected and v is added to the cluster C_u seeded by it. The inner product is defined as the total algebraic weight of the edges connecting v with the seed u. Concretely, $\text{ipm}(v, u) = \Sigma_{e|v,u \in e}\ \rho(e)$. See Listing 2 for pseudocode. We experimented with visiting the non-seeds in order of decreasing future volume and with using connectivity to make decisions when establishing aggregation rules. These approaches are more computationally intensive and do not produce better results (see Appendix B of full version [51] for the comparison of different parameters).

Stable assignment aggregation begins by constructing preference lists. Each seed orders adjacent non-seeds in the order of decreasing total algebraic weight of the hyperedges

Listing 2 Inner-product aggregation.

```
for i in F:
        j = argmax_{u∈C} ipm(v,u)
        C_j.insert(i)
```

Listing 3 Stable matching aggregation.

```
def propose(i):
        for j in pref_list[i]:
                if waitlist[i].size > threshold:
                        return
                if propos[j] == -1: // j holds no proposal
                        propos[j] = i
                        waitlist[i].push_back(j)
                        continue
                if i is preferable to propos[j]:
                        rejected = propos[j]
                        propos[j] = i
                        waitlist[i].push_back(j)
                        propose(rejected)

// Step 1: compute preference lists
for i in F:
        for j in seed_neighbors[i]:
                pref[j] = Σρ(e_ij) // pref is a hashtable
        for j in sort_by_value(pref).keys():
                pref_list[i].push_back(j)
for i in C:
        for j in non_seed_neighbors[i]:
                pref[j] = Σρ(e_ij)
        for j in sort_by_value(pref).keys():
                pref_list[i].push_back(j)

// Step 2: compute stable assignment
for i in C:
        propose(i)
```

connecting them (and vice versa): $\text{pref}_i(j) = \Sigma\rho(e_{ij})$. Then the stable assignment is computed using an algorithm similar to the classical one described in [19]. Each seed in C proposes to non-seeds in its preference list. If the non-seed does not have a better offer, it tentatively accepts the proposal and is put on the waitlist. If that non-seed later receives a better offer (i.e., an offer from a seed that ranks higher on its preference list), it rejects the current offer and the rejected seed proposes to the next candidate on its preference list. To discourage the creation of very large clusters, we limit the size of waitlist for a seed to the maximal vertex weight on a given coarsening level times three plus ten: $\text{len(waitlist)} = 3 \times \text{max\_vtx\_wgt} + 10$. Procedure terminates when each non-seed has been assigned to a waitlist or a seed has been rejected by every non-seed. At this point each seed forms a cluster with all vertices on its waitlist, subject to size constraint (we guarantee that no cluster can be larger than total vertex weight over the number of parts). The fact that we use a classical problem as a subproblem in our heuristic allows us to potentially leverage the previous work in optimizing and parallelizing stable assignment, such as [35],[36] and [22]. The pseudocode is presented in Listing 3.

5 Results

We implemented all algorithms described in this paper within Zoltan [15] package of the Trilinos Project [26]. Zoltan is an open-source toolkit of parallel combinatorial scientific computing algorithms [15]. It includes a hypergraph partitioning algorithm PHG (Parallel HyperGraph partitioner) and interfaces to PaToH and hMetis2. We added our new coarsening schemes and left other phases of the multilevel framework unchanged. Our implementation, data, and full results are available at `http://bit.ly/aggregative2018code`.

The hypergraphs in our benchmark are generated from a selection of matrices using the row-net model. In the row-net model, each column of the matrix represents a vertex, each row represents an edge and a vertex j belongs to the hyperedge i if there is a non-zero element at the intersection of j-th column and i-th row, i.e., $A_{ij} \neq 0$. All matrices (more than 300) were obtained from SuiteSparse Matrix Collection [14] that includes other collections. For each combination of hypergraph/algorithm/set of parameters, we executed 20 experiments.

We compare our algorithm with four state-of-the-art partitioners: hMetis2 [30], PaToH v3.2 [9], Zoltan PHG [15] and Zoltan-AlgD [50]. PaToH is used as a plug-in for Zoltan with default parameters described in Zoltan's User Guide [17]. hMetis2 is used in k-way mode with all parameters set to default: greedy first-choice scheme for coarsening, random k-way refinement, and min-cut objective function. The reason we run hMetis in k-way mode is the way hMetis specifies imbalance constraint. In recursive bisection mode, the imbalance constraint is applied *at each bisection step*, therefore relaxing the constraint as the number of parts increases. We found it almost impossible to compare hMetis in recursive bisection mode fairly (i.e., with the same imbalance) with other partitioners. Both Zoltan and PaToH are used in serial mode.

Optimizing the constants in the running time of the proposed algorithms is beyond the scope of this paper. Currently, for the existing unoptimized implementation the running time of other state-of-the-art hypergraph partitioners is not improved except for those that generate less levels in the hierarchy. In the experiments, the runtime of unoptimized implementation of our algorithms is up to an order of magnitude larger than the runtime of other state-of-the-art partitioners in worst cases. However, we must point out that our algorithm utilizes the building blocks and ideas of algebraic multigrid, which makes it possible to improve the runtime drastically by leveraging a plethora of existing research in optimizing and parallelizing algebraic multigrid solvers(e.g. [25], [40]). Similarly, there exists extensive research into optimizing the performance of stable matching solvers. Manne et al. [36] demonstrate the connection between graph matchings and stable marriage and show the scalability of Gale-Shapely type algorithms. Munera et al. [38] present an adaptive search formulation of stable marriage problem and take advantage of a Cooperative Parallel Local Search framework [37], achieving superlinear speedup. Gelain et al. [21] demonstrate a different efficient local search method for stable marriage problem.

In Figures 3 and 4 the results are presented graphically. In the main body of the paper, we only plot the results for 10% imbalance. For results for other imbalance factors please refer to Appendix A of the full version [51]. In Figure 3, we show the results of inner-product algebraic multirgid aggregation coarsening. In Figure 4, the stable matching aggregation is demonstrated. We use frequency histograms to present the distribution of cut differences between our methods and other state-of-the-art hypergraph partitioners. The value being represented (see horizontal axes) is the ratio

$$\zeta = \frac{\text{cut obtained using another partitioner}}{\text{cut obtained using our method}}. \tag{12}$$

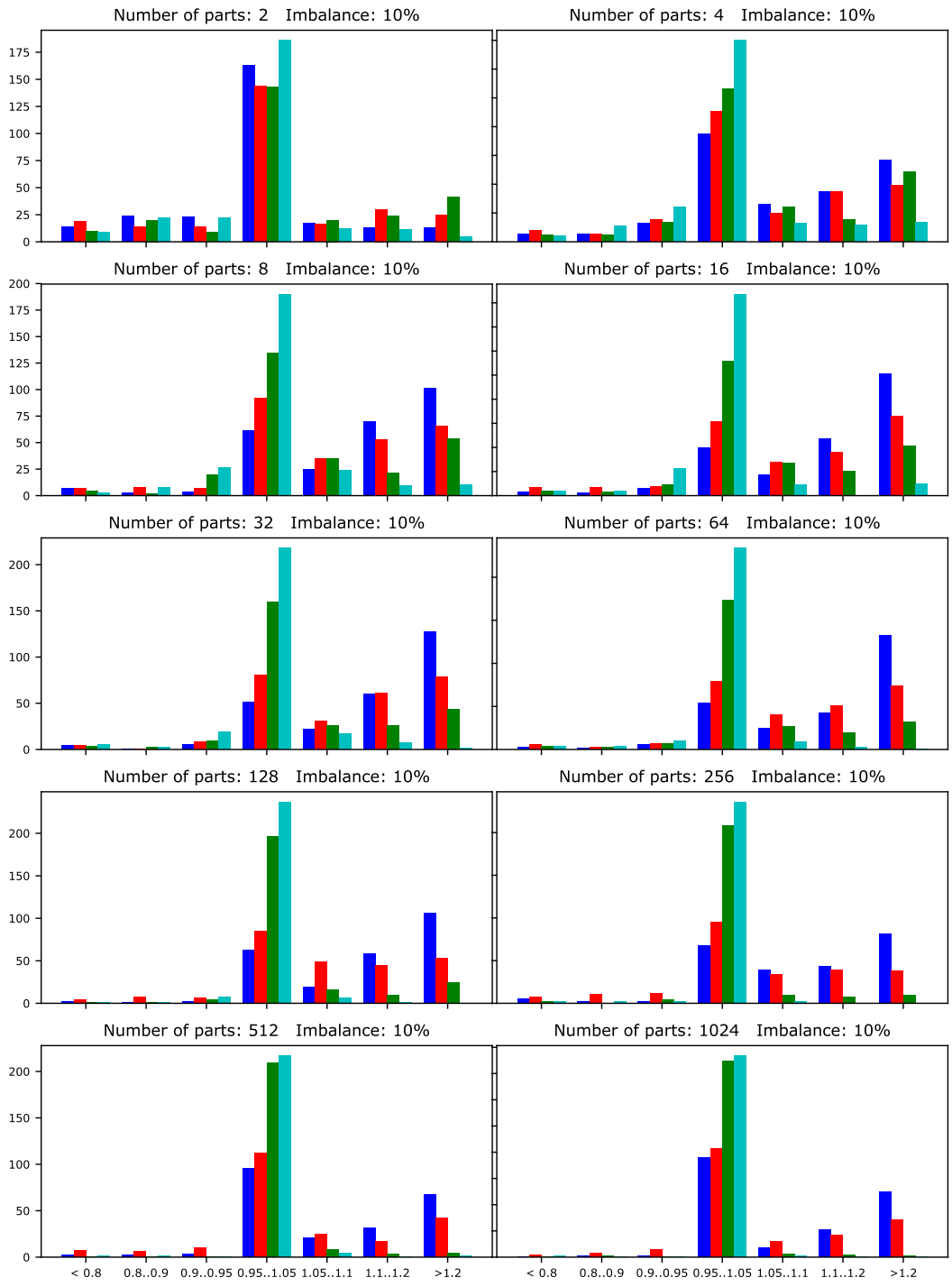

Figure 3 Histogram of ζ for coarsening using algebraic multigrid inner-product aggregation. Blue rectangle corresponds to PaToH, red to hMetis2, green to Zoltan PHG and cyan to Zoltan-AlgD.

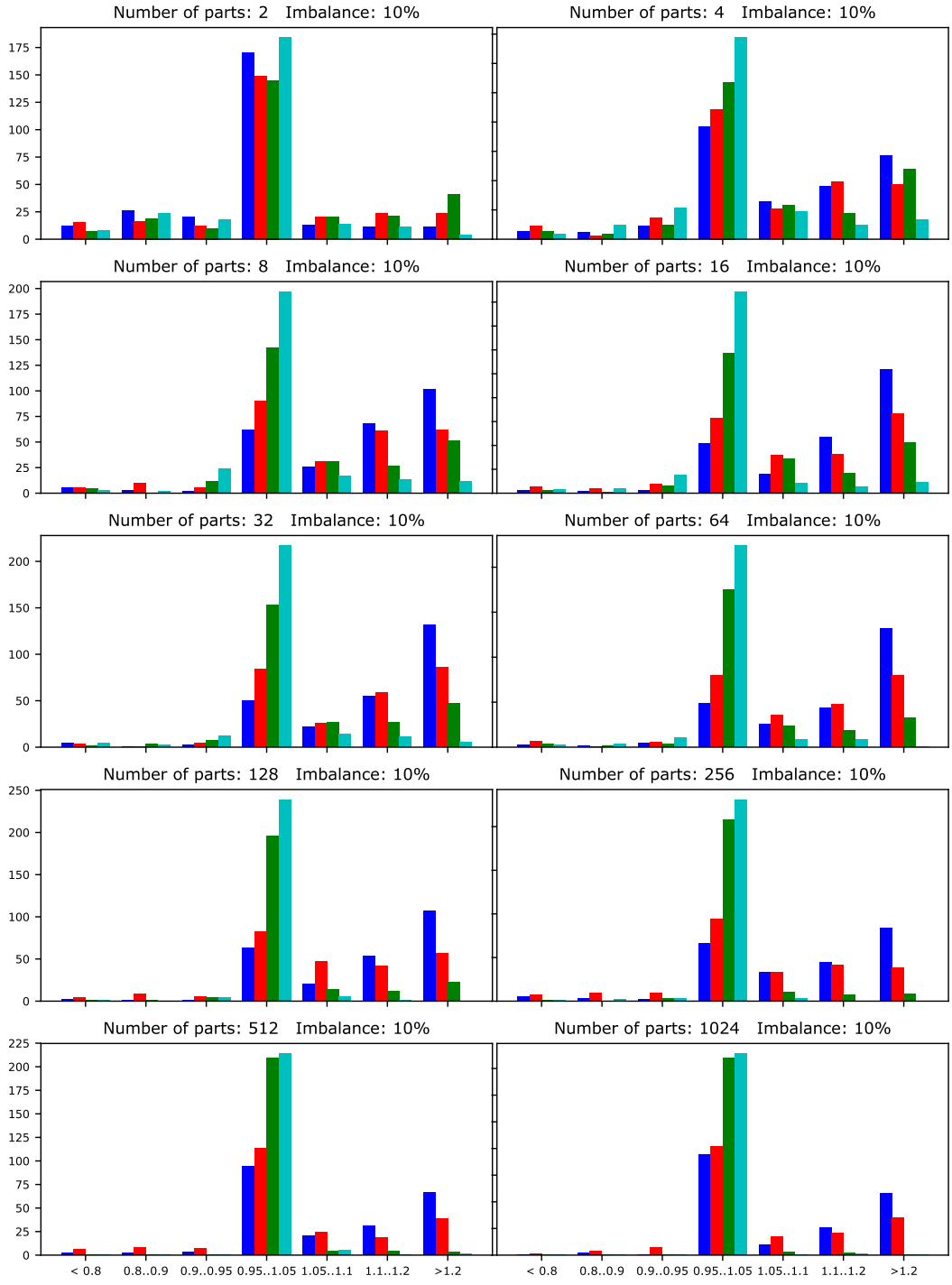

Figure 4 Histogram of ζ for coarsening using stable matching aggregation. Blue rectangle corresponds to PaToH, red to hMetis2, green to Zoltan PHG and cyan to Zoltan-AlgD.

Each bin corresponds to a range of the ratios (for example, the middle bin corresponds to the differences of less than ±5% and the rightmost to the improvements of $> 20\%$). Each rectangle corresponds to a partitioner: blue corresponds to PaToH, red corresponds to hMetis2, green corresponds to Zoltan PHG and cyan corresponds to Zoltan-AlgD. For the full results, please refer to `http://bit.ly/aggregative2018results`

The results demonstrate that given the same refinement, the proposed schemes are at least as effective as traditional matching-based schemes, while outperforming them on many instances. Both proposed coarsening schemes almost equally succeed in improving the quality of solvers (see Appendix A of the full version [51] for comparison of the performance of two algorithms). Further investigation of the difference between them is a very interesting future research direction, because, in fact, they represent very different algorithms. Since Zoltan utilizes recursive bisectioning scheme, we can see that improvements decrease as number of parts increases. This can be attributed to refinement becoming more and more important as number of parts increases.

6 Conclusion

We have presented two novel aggregative coarsening schemes for hypergraphs. The introduced schemes incorporate ideas of algebraic multigrid and stable matching into multilevel hypergraph partitioning framework. We have implemented the described algorithms within state-of-the-art hypergraph partitioner Zoltan and compared their performance against a number of other state-of-the-art partitioners on a large benchmark.

The experimental results demonstrate that given the same uncoarsening, the proposed coarsening schemes perform at least as well as traditional matching-based schemes, while outperforming them on many instances. This suggests that algebraic-multigrid-inspired coarsening schemes have great potential when combined with appropriate refinement.

References

1 Yaroslav Akhremtsev, Tobias Heuer, Peter Sanders, and Sebastian Schlag. Engineering a direct k-way hypergraph partitioning algorithm. In *19th Workshop on Algorithm Engineering and Experiments, (ALENEX 2017)*, pages 28–42, 2017.

2 Charles J Alpert and Andrew B Kahng. Recent directions in netlist partitioning: a survey. *Integration, the VLSI journal*, 19(1-2):1–81, 1995.

3 David A Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. Graph partitioning and graph clustering: 10th dimacs implementation challenge, vol. 588. *American Mathematical Society*, 7:210–223, 2013.

4 Stephen T Barnard and Horst D Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency and computation: Practice and Experience*, 6(2):101–117, 1994.

5 V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and R. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 10:P10008, 2008.

6 Achi Brandt and Dorit Ron. Multigrid solvers and multilevel optimization strategies. In *Multilevel optimization in VLSICAD*, pages 1–69. Springer, 2003.

7 Aydin Buluç and Erik G Boman. Towards scalable parallel hypergraph partitioning. *CSRI Summer Proceedings 2008*, page 109, 2008.

8 Aydin Buluc, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering: Selected Results and Surveys*, volume 9220, pages 117–158. Springer, 2016.

9 Ümit Çatalyürek and Cevdet Aykanat. Patoh (partitioning tool for hypergraphs). In *Encyclopedia of Parallel Computing*, pages 1479–1487. Springer, 2011.

10 Umit V Catalyurek and Cevdet Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, 1999.

11 Umit V Catalyurek, Erik G Boman, Karen D Devine, Doruk Bozdağ, Robert T Heaphy, and Lee Ann Riesen. A repartitioning hypergraph model for dynamic load balancing. *J. Parallel Distrib. Comput.*, 69(8):711–724, 2009.

12 Jie Chen and Ilya Safro. Algebraic distance on graphs. *SIAM J. Sci. Comput.*, 33(6):3468–3490, 2011.

13 Carlo Curino, Evan Jones, Yang Zhang, and Sam Madden. Schism: a workload-driven approach to database replication and partitioning. *Proceedings of the VLDB Endowment*, 3(1-2):48–57, 2010.

14 Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

15 Karen D Devine, Erik G Boman, Robert T Heaphy, Rob H Bisseling, and Umit V Catalyurek. Parallel hypergraph partitioning for scientific computing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

16 Karen D Devine, Erik G Boman, Robert T Heaphy, Bruce A Hendrickson, James D Teresco, Jamal Faik, Joseph E Flaherty, and Luis G Gervasio. New challenges in dynamic load balancing. *Applied Numerical Mathematics*, 52(2-3):133–152, 2005.

17 Karen D Devine, Vitus Leung, Erik G Boman, Sivasankaran Rajamanickam, Lee Ann Riesen, and Umit Catalyurek. Zoltan user's guide, version 3.8.(2014), 2014. URL: `http://www.cs.sandia.gov/zoltan/ug_html/ug_alg_patoh.html`.

18 Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *Papers on Twenty-five years of electronic design automation*, pages 241–247. ACM, 1988.

19 David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

20 Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. New York, 2002.

21 Mirco Gelain, Maria Silvia Pini, Francesca Rossi, K Brent Venable, and Toby Walsh. Local search approaches in stable matching problems. *Algorithms*, 6(4):591–617, 2013.

22 Giorgos Georgiadis and Marina Papatriantafilou. Overlays with preferences: Distributed, adaptive approximation algorithms for matching with preference lists. *Algorithms*, 6(4):824–856, 2013.

23 Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.

24 Bruce Hendrickson and Robert W Leland. A multi-level algorithm for partitioning graphs. *SC*, 95(28), 1995.

25 Van Emden Henson and Ulrike Meier Yang. Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, 2002.

26 Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P Pawlowski, Eric T Phipps, et al. An overview of the trilinos project. *ACM Trans. Math. Software*, 31(3):397–423, 2005.

27 Y. F. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM J. Sci. Comput.*, 23(4):1352–1375, 2001. `doi:10.1137/S1064827500377733`.

28 Ruoming Jin, Yang Xiang, David Fuhry, and Feodor F. Dragan. Overlapping matrix pattern visualization: A hypergraph approach. *Data Mining, IEEE International Conference on*, 0:313–322, 2008. doi:10.1109/ICDM.2008.102.

29 Emmanuel John and Ilya Safro. Single-and multi-level network sparsification by algebraic distance. *Journal of Complex Networks*, 5(3):352–388, 2016.

30 George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 7(1):69–79, 1999.

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

32 Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49(2):291–307, 1970.

33 K Ashwin Kumar, Abdul Quamar, Amol Deshpande, and Samir Khuller. Sword: workload-aware data placement and replica selection for cloud data management systems. *The VLDB Journal*, 23(6):845–870, 2014.

34 Oren E Livne and Achi Brandt. Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

35 Enyue Lu and SQ Zheng. A parallel iterative improvement stable matching algorithm. In *International Conference on High-Performance Computing*, pages 55–65. Springer, 2003.

36 Fredrik Manne, Md. Naim, Håkon Lerring, and Mahantesh Halappanavar. *On Stable Marriages and Greedy Matchings*, pages 92–101. SIAM, 2016. doi:10.1137/1.9781611974690.ch10.

37 Danny Munera, Daniel Diaz, Salvador Abreu, and Philippe Codognet. A parametric framework for cooperative parallel local search. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 13–24. Springer, 2014.

38 Danny Munera, Daniel Diaz, Salvador Abreu, Francesca Rossi, Vijay A Saraswat, and Philippe Codognet. Solving hard stable matching problems via local search and cooperative parallelization. In *AAAI*, pages 1212–1218, 2015.

39 David A Papa and Igor L Markov. Hypergraph partitioning and clustering., 2007.

40 Jongsoo Park, Mikhail Smelyanskiy, Ulrike Meier Yang, Dheevatsa Mudigere, and Pradeep Dubey. High-performance algebraic multigrid solver optimized for multi-core based distributed parallel systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 54. ACM, 2015.

41 Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423, 2011.

42 Randolf Rotta and Andreas Noack. Multilevel local search algorithms for modularity clustering. *Journal of Experimental Algorithmics (JEA)*, 16:2–3, 2011.

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

44 I. Safro, D. Ron, and A. Brandt. Multilevel algorithm for the minimum 2-sum problem. *Journal of Graph Algorithms and Applications*, 10(2):237–258, 2006.

45 Ilya Safro, Dorit Ron, and Achi Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

46 Ilya Safro, Dorit Ron, and Achi Brandt. Multilevel algorithms for linear ordering problems. *ACM Journal of Experimental Algorithmics*, 13, 2008. doi:10.1145/1412228.1412232.

47 Ilya Safro, Peter Sanders, and Christian Schulz. Advanced coarsening schemes for graph partitioning. *ACM Journal of Experimental Algorithmics (JEA)*, 19:2–2, 2015.

48 Ilya Safro and Boris Temkin. Multiscale approach for the network compression-friendly ordering. *J. Discrete Algorithms*, 9(2):190–202, 2011. doi:10.1016/j.jda.2010.09.007.

49 Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. k-way hypergraph partitioning via n-level recursive bisection. In *18th Workshop on Algorithm Engineering and Experiments, (ALENEX 2016)*, pages 53–67, 2016.

50 Ruslan Shaydulin, Jie Chen, and Ilya Safro. Relaxation-based coarsening for multilevel hypergraph partitioning. *arXiv preprint arXiv:1710.06552*, 2017.

51 Ruslan Shaydulin and Ilya Safro. Aggregative coarsening for multilevel hypergraph partitioning. *CoRR*, abs/1802.09610, 2018. `arXiv:1802.09610`.

52 Aleksandar Trifunovic. *Parallel algorithms for hypergraph partitioning*. PhD thesis, University of London, 2006.

53 C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals Oper. Res.*, 131:325–372, 2004.

54 Chris Walshaw. A multilevel approach to the travelling salesman problem. *Operations Research*, 50(5):862–877, 2002.

55 Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *NIPS*, volume 19, pages 1633–1640, 2006.