Hypergraph Partitioning With Embeddings

Justin Sybrandt, *Member, IEEE,* Ruslan Shaydulin, *Member, IEEE,* Ilya Safro, *Member, IEEE*

Abstract—The problem of placing circuits on a chip or distributing sparse matrix operations can be modeled as the hypergraph partitioning problem. A hypergraph is a generalization of the traditional graph wherein each "hyperedge" may connect any number of nodes. Hypergraph partitioning, therefore, is the NP-Hard problem of dividing nodes into k similarly sized disjoint sets while minimizing the number of hyperedges that span multiple partitions. Due to this problem's complexity, many partitioners leverage the multilevel heuristic of iteratively "coarsening" their input to a smaller approximation until an inefficient algorithm becomes feasible. The initial solution is then propagated back to the original hypergraph, which produces a reasonably accurate result provided the coarse representation preserves structural properties of the original. The multilevel hypergraph partitioners are considered today as state-of-the-art solvers that achieve an excellent quality/running time trade-off on practical large-scale instances of different types. In order to improve the quality of multilevel hypergraph partitioners, we propose leveraging graph embeddings to better capture structural properties during the coarsening process. Our approach prioritizes dense subspaces found at the embedding, and contracts nodes according to both traditional and embedding-based similarity measures.
Reproducibility: All source code, plots and experimental data are available at https://sybrandt.com/2019/partition.

1 Introduction

In order to model problems that contain interconnected groups of items, such as the various data dependencies between processes found in large scientific applications, many leverage the formalism of hypergraphs. A hypergraph is similar to a traditional graph, with the added generalization that the "hyperedges" may connect any number of nodes. Hypergraphs have been used in VLSI design [1], machine learning [2], [3], [4], parallel algorithms [5], combinatorial scientific computing [6], and social network analysis [7], [8].

The *hypergraph partitioning* problem is that of dividing the nodes of a hypergraph among k similarly-sized disjoint sets. A good partitioning is one that minimizes the number of hyperedges spanning multiple partitions. In the context of combinatorial scientific computing and load balancing, this is the problem of dividing logical threads (nodes) across the various available machines (partitions) in order to reduce the amount of communication necessary between machines (cut hyperedges). Unfortunately, it both is NP-Hard to solve [9] or accurately approximate [10] a solution to this problem.

To mange the complexity of hypergraph partitioning, practitioners turn to heuristical algorithms [11], such as the multilevel paradigm [12], [13], [14], [15], [16], [17]. The multilevel approach consists of a V-Cycle containing three phases, depicted in Figure 1. The V-cycle starts by iteratively *coarsening* the input hypergraph. Each iteration of the coarsening creates new coarse nodes by contracting groups of nodes in the the current set. These contractions are determined through a matching process that is informed by some similarity measure so that the resulting approximation retains the structural features of the original problem. This allows the coarse level partition to be interpolated to the next-finer level without applying too many refinement steps that may substantially slow down the entire multilevel framework. Coarsening continues until the approximate hypergraph is small enough to partition directly, forming the *initial solution*. Multilevel partitioners then expand this approximate solution by iteratively *uncoarsening* to the original input. At each stage of the uncoarsening process, solvers interpolate the coarse solution and perform a local search or other methods to refine it. The resulting solution, once gradually refined to the highest level, becomes the final partitioning.

Because the initial solution to a multilevel algorithm propagates through the entire uncoarsening process, it is important to create a coarsened representation that shares structural properties with the original hypergraph. In order to improve coarsening, other solvers have exploited clustering and community detection techniques [18], algebraic distance [13], and others. However, recent advances in graph embedding [19] indicate that the latent spaces found by unsupervised machine learning algorithms can better identify structural similarities between nodes.

1.1 Our Contribution

In this work we propose exploiting latent node representations gained through embeddings to better coarsen large hypergraphs for partitioning. First, we apply star-expansion [20] to gain a bipartite representation of the input hypergraph. Then we learn latent structural features of this graph using a graph embedding method. Note that our algorithm is agnostic to the particular embedding. These dense real-valued embeddings inform our coarsening algorithm to prioritize more similar nodes at each level of coarsening. Then, we identify coarsening partners by comparing latent features in conjunction with traditional edge-wise features. After each iteration, we assign newly coarsened nodes an embedding equal to the centroid of their primal embeddings.

We implement our coarsening algorithm in both the n-level solver KaHyPar [11], as well as the $(\log n)$-level

• *Authors are with the School of Computing, Clemson University, Clemson SC 29634 USA. Emails: {jsybran, rshaydu, isafro}@clemson.edu.*

Manuscript received 9 Sept. 2019.

solver Zoltan [16]. In the case of KaHyPar, we evaluate our coarsening under its original uncoarsening strategy, as well as its recent flow-based refinement [21]. We also compare our solution quality when using six different graph embeddings: Node2Vec [22], Metapath2Vec++ [23], Boolean and Algebraic Heterogeneous Bipartite Graph Embeddings [19], as well as two combination embeddings, also proposed in [19].

We evaluate our implementations against five state-of-the-art partitioners: hMetis [24], Zoltan [16], PaToH [25], KaHyPar (with community-based coarsening [18]), and KaHyPar Flow (with both community-based coarsening and flow-based refinement [21]). For each method we additionally compare both the cut and $k-1$ optimization objectives[1]. Our evaluation spans a range of the number of partitions from 2 to 128, and 96 graphs from the SuiteSparse Matrix Collection [26]. For each combination of proposed implementation, baseline method, optimization metric, partition count, and hypergraph we perform twenty trials each with a different random seed and relabeling of the input graph. This analysis consists of over half-a-million individual experiments.

We report summary statistics for the improvement of each proposed implementation when compared to each baseline method. Specifically, we consider the improvement relative to the minimum, maximum, and average observed objective value as well as the standard deviation of trials. We additionally supply more detailed plots in the online appendix for the improvements of all graphs across all method comparisons in order to highlight graph-wise difference. These plots display the averages and standard deviations of each graph per comparison, and include statistical significant values. Additionally, all experimental data for each individual trial, including parameter settings, is available as a publicly downloadable MongoDB database dump[2].

Using our proposed coarsening, we observe a significant improvement between each implementation and its directly comparable baseline (e.g., our modified Zoltan against the baseline Zoltan). We observe, however, that the improvement gradually vanishes as the number of parts is increasing, indicating a promising future research direction. In some specific cases, such as hypergraphs representing social networks, our coarsening can find partitioning solutions that are over **400%** better than the existing solutions. Our coarsening also improves the standard deviation of results. Typical multilevel solvers visit nodes in a random order for each level of coarsening. Our approach replaces this with a prioritized visit order derived from embeddings. This change decreases the standard deviation for almost all scenarios by over 100% and often as high as 500%. All experimental code, data, visualization scripts, and results are publicly available at https://sybrandt.com/2019/partition.

2 Background

A hypergraph H is an ordered pair $H = (V, E)$, where V is the set of nodes and E is the set of hyperedges. Each hyperedge $e \in E$ is a non-empty subset of V. In hypergraph

1. Note that hMetis does not optimize for $k-1$. For this objective we only compare against the remaining four.
2. https://sybrandt.com/2019/partition

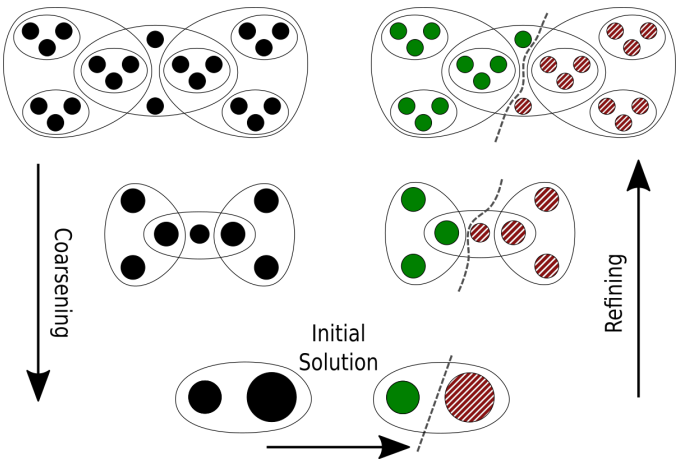

Fig. 1: A standard V-cycle, consisting of coarsening, and initial partition, and uncoarsening. Node size corresponds to the weight of hypothetical coarse nodes. The dashed line demonstrates the initial partition and iterative local searches at each uncoarsening level. In this example, the multilevel hierarchy consists of three levels.

k-partitioning the goal is to split the set of nodes V into k disjoint subsets or parts $(V_1, \ldots, V_k)$ such that $V_1 \cup \ldots \cup V_k = V$ while minimizing an objective function over cut hyperedges subject to an imbalance constraint factor ϵ. A hyperedge belongs to the cut E_{cut} if it contains nodes from at least two parts: $e \in E_{cut}$ iff $\exists u, v \in e : u \in V_i, v \in V_j, i \neq j$. Both nodes and hyperedges can have weights, namely, $w_v \in \mathbb{R}_{\geq 0}$, and $w_e \in \mathbb{R}_{\geq 0}$, for each $v \in V$, and $e \in E$, respectively. In this paper we consider two objective functions: "cut" and "$k-1$". The cut is the sum of weights of cut hyperedges: $\sum_{e \in E_{cut}} w_e$. Connectivity of an edge $\lambda(e)$ is defined as the number of parts an edge spans. The $k-1$ metric is then defined as $\sum_{e \in E_{cut}} (\lambda(e) - 1) w_e$. Note that for $k = 2$ these two metrics are equivalent. The imbalance factor ϵ ensures that for each part V_i the following holds $\sum_{u \in V_i} w_u \leq (1+\epsilon) \lceil \frac{1}{k} \sum_{u \in V} w_u \rceil$.

Many partitioning algorithms assign weights to both nodes and hyperedges. Initially, weights are all set equally to 1. Once coarsened, the weight of a newly coarsened node is set equal to the sum of the weights of the contracted fine nodes. Coarse hyperedges are similarly weighted whenever two hyperedges are merged.

2.1 Multilevel Hypergraph Partitioning

Multilevel algorithms solve problems by constructing a hierarchy of sub-problems that approximate the original. These "coarsened" sub-problems contain fewer degrees of freedom and are therefore easier to solve. The multilevel approach captures the global structure of the problem by combining local information at different levels of coarseness. Originally introduced to speed up existing algorithms [27] and inspired by multigrid and multiscale optimization strategies [28], the multilevel method was quickly recognized to be a good way to improve the quality of partitioning [29] and is currently considered to be one of the state-of-the-art methods [30] for this problem. In the context of hypergraphs, one constructs a multilevel hierarchy by merging nodes — multiple nodes at the finer level become a single node in the coarser level. Once reduced to

a sufficiently small problem, a multilevel partitioner can solve the coarse parititoning problem using an algorithm that would be infeasible on large-scale instances. This initial solution is iteratively uncoarsened by first interpolating it onto a finer level and then refining it. The refinement is typically performed using local search or other methods. The coarsening-uncoarsening pipeline is commonly referred to as V-cycle (see Fig. 1). Traditionally, at each level of the coarsening process all or almost all nodes have at least one merging partner, resulting in $\log n$ levels. This is the approach used by Mondriaan [31], hMetis2 [1], Zoltan [16], and PaToH [25]. However, KaHyPar [11] implements an n-level approach where at each level only one pair of nodes is contracted. Over the years the multilevel method has become the gold standard in hypergraph partitioning to achieve an excellent time/quality trade-off in many practical cases and is used by most state-of-the-art solvers, including all of the ones discussed in this paper. For an extensive review of (hyper)graph partitioning methods, the reader is referred to [30], [32].

When constructing coarser hypergraphs, state-of-the-art partitioners contract nodes according to some heuristic such that during the uncoarsening the solution can be interpolated from coarser levels without the loss of quality. These methods typically make coarsening decisions based on some similarly measure that can be computed on node-pairs. Most multilevel hypergraph partitioners, including Mondriaan [31], hMetis2 [1] and Zoltan [16], measure inner product or its variations, such as *absorption* (PaToH [25]) and *heavy edge* (hMetis2 [1], Parkway [33], PaToH [25] and KaHyPar [18]).

The inner product of two nodes is defined as the Euclidean inner product of the weighted hyperedge incidence vectors [16]. This similarity measure and its variations are simple and computationally inexpensive, but are limited due to only using local information. Recently, a number of advanced coarsening schemes were introduced to address this limitation. Shaydulin et al. introduce a relaxation-based similarity metric *algebraic distance* [13], extending a similar approach from graphs [34]. In [35] this approach is extended and incorporated within an aggregative coarsening scheme, inspired by algebraic multigrid and stable matching approaches. An unfinished but promising attempt to generalize hypergraph coarsening using algebraic multigrid (AMG) on graphs [36] was published in Sandia Labs Summer Reports [37]. In AMG coarsening [38], [39], [40], instead of being contracted, the nodes are split into fractions which form coarse aggregates. Heuer and Schlag introduce a community-aware coarsening that uses global clustering information to restrict matching between communities [18].

During uncoarsening, nodes are uncontracted and the coarser-level partition is interpolated to the finer-level node set. Then the solution is iteratively refined using a local node-moving heuristic. A majority of hypergraph partitioners use a variation of Fiduccia-Mattheyses [41] or Kernighan-Lin [42] to perform these local searches [1], [16], [21], [25], [31], [33]. Using a local search heuristic at the uncoarsening stage allows these partitioners to locally improve the global solution interpolated from coarser levels. Recently, Heuer et al. introduced a flow-based refinement scheme for k-way hypergraph partitioning [21], extending similar approaches from graph partitioning [43].

2.2 Hypergraph Embeddings

The Skip-Gram text embedding model presented by Mikolov et al. learns embeddings by discovering the relationship between each word and its typical context [44], [45]. This model underpins many graph embedding models [22], [23], [46]. In order to efficiently handle large volumes of text the Skip-Gram model samples "windows." Each window is centered around a target word, and includes local context both leading and trailing the target. The Skip-Gram model learns to predict each window's contents given the target word. The underlying assumption behind this approach is that "similar words share similar company," and has shown to result in semantically rich latent features [47], [48].

Deepwalk, a pioneer graph embedding technique presented by Perozzi et al., reduces the graph structure to an analogous textual problem in order to also leverage the Skip-Gram approach [46]. Here, the underlying assumption is "similar nodes share similar company," however graphs present additional challenges not found in text. Firstly, representing a node's "company" is nontrivial. To simply take all first-order neighbors of a target node may not be sufficient, or may contain more neighbors than fit in memory. To reconcile this, Perozzi et al. proposes random-walk sampling. Pseudo-sentences form when traversing a graph, wherein each node is analogous to a word and each random walk is analogous to a sentence. These random walks serve as input to the hierarchical Skip-Gram model, similar to that proposed by Mikolov et al.

Extending this approach, Grover and Leskovec modify random-walk sampling to add a "return probability" parameter [22]. They observe that typical depth-first random walks capture structural similarities, while breadth-first approaches (such as the LINE embedding method [49]) capture homophilic relationships. In order to improve overall embedding quality, Node2Vec random walks strike a balance between the two traversal strategies by probabilistically doubling-back on old neighborhoods, or forging onward to unseen areas.

The above-mentioned graph embedding techniques assume nothing is known about the considered graph's structure. However, recent methods address particular graph techniques that are applicable to hypergraphs. Metapath2Vec++, proposed by Dong et al., assumes each node is of a particular type, and that certain "metapaths," path descriptions containing only type information, are known to be meaningful [23]. In the case of hypergraphs, we can perform a star-expansion to convert each hyperedge to a new layer of nodes, which converts our input hypergraph into a traditional bipartite graph [20]. This representation has two node types, original nodes, and those derived from hyperedges. Due to its bipartite structure the only metapath is that of alternating types. However, due to the model architecture of Metapath2Vec++, we can learn some type-specific latent features for each.

FOBE + HOBE Details: Further recent work addresses the bipartite case specifically. Sybrandt and Safro present multiple methods for embedding bipartite graphs [19]. These include First- and High-Order Bipartite Embeddings,

as well as a combination approach to learn joint embeddings on bipartite graphs. These approaches are applicable to hypergraphs as represented through star-expansion. These methods model the two distinct types of nodes present in a bipartite network separately in order to better capture same-typed features. In the context of hypergraphs this is analogous to modeling nodes and hyperedges separately. For the purpose of this work however, we only consider the embedding of nodes present in the original hypergraph.

The first-order approach, FOBE, presented by Sybrandt and Safro samples observed node similarities and then learns embeddings to encode those similarities via dot product. Nodes are deemed "similar" in this context if they share an edge, or a neighbor. Formally, if $B = (V, E)$ is an undirected bipartite graph with nodes V and edges E, an edge from nodes $x, y \in V$ is indicated as $xy, yx \in E$, and $\Gamma(x) = \{y | xy \in E\}$ indicates the neighborhood of x, then the similarity measured by FOBE is:

$$\mathbb{S}_F(x, y) = \begin{cases} 1 & xy \in E \text{ or } \Gamma(x) \cap \Gamma(y) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Note that for two nodes to be measured as similar by the above equation, they must either be of different bipartite types and share an edges, or of the same type and share a neighbor of the opposite type. FOBE then encodes the above similarities into node embeddings. However, the objective used to learn these embeddings is constructed to only explicitly compare nodes of the same type. If A and B are disjoint subsets of V indicating the two types present in the bipartite graph, and $\epsilon(x)$ is a function that related node x to its embedding in $\mathbb{R}^k$, then the various encoded similarities are represented as the following:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2}$$

$$\widetilde{\mathbb{S}}_A(x, y) = \widetilde{\mathbb{S}}_B(x, y) = \sigma\left(\epsilon(x)^\intercal \epsilon(y)\right) \tag{3}$$

$$\widetilde{\mathbb{S}}_V(x, y) = \mathop{\mathbb{E}}_{z \in \Gamma(y)} \left[\widetilde{\mathbb{S}}_A(x, z)\right] \mathop{\mathbb{E}}_{z \in \Gamma(x)} \left[\widetilde{\mathbb{S}}_B(y, z)\right] \tag{4}$$

Here, $\widetilde{\mathbb{S}}_A$ and $\widetilde{\mathbb{S}}_B$ indicate the similarity shared between the nodes of the same type. Then, $\widetilde{\mathbb{S}}_V$ decomposes the similarity of cross-typed nodes into sets of same-typed comparisons. These predicted similarity measures derived from embeddings are fit to the observed samples above simultaneously.

The high-order embedding method (HOBE) presented by Sybrandt and Safro extends FOBE by sampling neighbors-of-neighbors, and prioritizes these similarities through the local heuristic signal provided by algebraic distance on graphs [13], [34], [40], [50]. This approach begins with a fast iterative relaxation technique that places all bipartite nodes on the $[0, 1]$ interval such that *locally similar* nodes are more likely to have similar values. Multiple trials with random initializations boost this signal by reducing the effect of incidental proximity observed between distant nodes in a single trial. Formally, this algebraic similarity measure is determined by first calculating algebraic coordinates for each node $\mathbf{a}(x)$. These coordinates are randomly initialled $\mathbf{a}^{(0)}$, and are refined over $t = 1, \ldots, K$ iterations via the following:

$$\mathbf{a}_r^{(t+1)}(v_i) = \lambda \mathbf{a}_r^{(t)}(v_i) + (1 - \lambda) \frac{\sum_{v_j \in \Gamma(v_i)} \mathbf{a}_r^{(t)}(v_j) |\Gamma(v_j)|^{-1}}{\sum_{v_j \in \Gamma(v_i)} |\Gamma(v_j)|^{-1}} \tag{5}$$

Here, r indicates the trail ($r \in \{1, \ldots, R\}$), t indicates the iteration, and λ is the damping factor (suggested $\lambda = 0.5$ in [13]). These coordinates per-trial are then combined into a more robust similarity measure through the following:

$$s(x, y) = \frac{\sqrt{R} - \sqrt{\sum_{r=1}^{R} \left(\mathbf{a}_r^{(K)}(x) - \mathbf{a}_r^{(K)}(y)\right)^2}}{\sqrt{R}} \tag{6}$$

Building from this heuristic signal, the HOBE similarity measures the presence of highly-similar shared neighbors through the following:

$$\mathbb{S}_A^{'}(x, y) = \mathbb{S}_B^{'}(x, y) = \max_{z \in \Gamma(x) \cap \Gamma(y)} \min\left(s(x, z), s(y, z)\right) \tag{7}$$

$$\mathbb{S}_V^{'}(x, y) = \max\left(\max_{z \in \Gamma(y)} \mathbb{S}_A^{'}(x, z), \max_{z \in \Gamma(x)} \mathbb{S}_B^{'}(y, z)\right) \tag{8}$$

In a manner similar to FOBE, these three similarity measures are encoded into the dot product of embeddings ϵ through a combined objective function.

In this work we explore the solution quality of our coarsening algorithm using a number of different embedding methods. We select Node2Vec and Metapath2Vec++ as well as both FOBE and HOBE to explore. In addition, we train two combination embeddings, one that merges all four methods, and another that combines only FOBE and HOBE. We do not attempt to demonstrate that any individual embedding is superior for hypergraph partitioning, on the contrary we demonstrate in Section 5 that all embeddings improve the partitioning quality, showing that such embeddings are an excellent tool for advanced coarsening schemes potentially not only for the partitioning problem. Node2Vec allows us to evaluate a generic embedding technique not designed with hypergraphs in mind, Metapath2Vec++ evaluates a method shown to transfer well to hypergraphs [19], while the Heterogeneous Bipartite approaches are designed to facilitate hypergraph learning.

3 Method

In order to improve the quality of multilevel hypergraph partitioning solvers, such as Zoltan [16] and KaHyPar [11], we take advantage of graph embedding techniques. These methods learn dense, real-valued representations in a fixed-sized vector space for each node. In the case of traditional graphs, Grover et al. demonstrate that these embeddings can capture both structural and homophilic latent relationships [22]. Additional work from Sybrandt and Safro demonstrates that these methods extend to hypergraphs [19] through star expansion [20].

Graph embedding methods typically encode observed similarities through some similarity measure. In the case of Algebraic and Boolean Heterogeneous Bipartite Embeddings, these similarities are explicitly modeled using the dot product [19]. The same similarity measure is also found in more traditional methods such as LINE [49]. Semantically, dot product implies that two nodes are similar if they share common prominent features. Unlike cosine similarity, the

dot product is not normalized, and therefore does not significantly penalize nodes for being dissimilar, provided their dissimilar values are near zero. We observe that dot product also applies to other graph embedding techniques, such as the Skip-Gram-based methods used in Node2Vec [22], Deepwalk [46], and by extension, MetaPath2Vec++ [23]. While the specifics of each method are beyond the scope of this work, we note that dot product is a robust measure of similarity across embeddings.

We exploit graph embeddings to better match nodes during coarsening. The typical matching process, in both n-level and $(\log n)$-level coarsening, identifies pairs of similar nodes, called coarsening partners, u, v to merge in the next-coarsest representation. The resulting coarsened node x becomes a member of all hyperedges incident to both u and v. As a result, the overall partitioning solution can be drastically altered by the quality of these node-pairs, as demonstrated below in Section 5.

One common node similarity measure for finding coarsening partners is an inner product of edge features. In KaHyPar [11], this measure is a ratio between edge weight and size, as reproduced in (9). Here, w_e corresponds to the weight of a coarsened hyperedge, which indicates the number of original hyperedges containing the same coarsened node set.

$$S_E(u,v) = \sum_{e \in E | u,v \in e} \frac{w_e}{|\{n \in e\}| - 1} \tag{9}$$

This measure prioritizes nodes sharing many "tight" hyperedges, those with fewer members, as these tend to be more meaningful in real-world applications. For instance, members of a selective club or shoppers buying a niche ingredient are likely more self-similar than those buying bread or belonging to a massive organization. However, this model equally prioritizes all hyperedges of similar size, even if they contain a random assortment of nodes. To improve this coarsening measure, we introduce a term derived from a pretrained graph embedding.

Hypergraph embeddings, typically derived from the bipartite representation, project nodes into a fixed-dimensionality vector space [19]. While the dimensionality of this space is a hyperparameter to an embedding model, typical values range from 100 to 1,000 and are robust to small changes. As a result, many methods capture similarities mathematically through the inner product of embeddings [19], [22], [23]. Formally, we represent the pretrained embedding as a function $\epsilon(v) : V \to \mathbb{R}^d$ mapping each node to a d-dimensional vector. We represent the embedding similarity between two nodes as

$$S_\epsilon(u,v) = \epsilon(u)^\intercal \epsilon(v). \tag{10}$$

These embeddings can capture both structural and homophilic latent properties [22]. Structural properties include hubs, bridges, and leafs, while homophilic properties include clusters and common neighbors. Different embedding techniques prioritize different latent features, and we explore six different embedding schemes to underpin our coarsening. These methods are outlined in detail in Section 2.2. However, we observe that all six considered embedding improve overall coarsening results (see Figure 4 as well as the online appendix).

We combine both hyperedge-wise and embedding-wise similarities into a single measure for each node pair. As a result, two nodes will be selected as coarsening partners if they share both many hyperedges as well as many latent features. This formulation provides a mechanism to lessen the impact of hyperedges without self-similar content, because the similarity conveyed by a tight hyperedge will be lessened by the dissimilarity conveyed in the embedding. In addition, we add in a regularization term to maintain balance between node weights. The weight of a coarsened node w_v is simply the number of original nodes that have been merged together in the coarsened representation. Without this penalty, dense subregions of the hypergraph could be coarsened entirely before anything else (in the n-level case), resulting in an imbalanced solution. Our resulting score is formally put

$$S(u,v) = \frac{S_E S_\epsilon}{w_u w_v}. \tag{11}$$

Note that to receive a high score given our proposed method, two nodes must share hyperedges, have similar latent features, and be of reasonably small weights. By including the edge-wise inner product, our method cannot coarsen disparate regions of the network that happen to share similar latent features, which can arise from some embedding techniques. For instance, disconnected subgraphs may be embedded in overlapping subspaces, and a simpler embedding-only similarity measure would then conjoin the disconnected components.

We additionally apply the latent information present in embeddings to order nodes when identifying coarsening partners. Our goal is to match the pairs with the highest similarity first, so that the resulting coarsened nodes more likely to share the same higher-order structural feature, such as a cluster or role. We sort nodes by their nearest neighbor in the embedding space, and penalize this similarity again by weights. We restrict the nearest-neighbor search to those nodes actually sharing hyperedges, as to match the scores calculated above. Formally, the sorting criteria we propose is as follows

$$S'(v) = \max_{u \in \Gamma(v), u \neq v} \frac{\epsilon(u)^\intercal \epsilon(v)}{w_u w_v}, \tag{12}$$

where $\Gamma(u)$ represents the neighborhood of node u, namely, $\Gamma(u) = \{v : \exists\, e \in E \mid u, v \in e\}$.

We present our overall matching algorithm in Procedure 1. All nodes begin unmatched, as indicated by M_v, a Boolean characteristic vector of (un)matched nodes. We then visit each node in sorted order, according to the above criteria. Provided a visited node is unmatched, we iterate its neighborhood and consider any unmatched neighbor that would not result in a coarse node above the weight tolerance. Out of these considered nodes, we select whichever has the highest score according to Eq. 11.

After coarsening, newly contracted nodes are assigned an embedding equal to the centroid of its primal nodes. In this context, a primal node is a fully uncoarsened node specified at the finest level of the problem. For instance, if at a given level of coarsening we match u and v, the resulting coarse node x would have the following properties. Here x represents the newly coarsened node, E'

represents the modified edge set, and P_v represents the set of primal nodes corresponding to node v. At the finest level, $P_v = \{v\}\ \forall v \in V$.

$$E_u = \{e \in E : u \in e\} \tag{13}$$
$$E_v = \{e \in E : v \in e\} \tag{14}$$
$$E' = \{e - \{u, v\} + \{x\} : e \in E_u \cup E_v\} \tag{15}$$
$$+ (E - E_u - E_v) \tag{16}$$
$$w_x = w_u + w_v \tag{17}$$
$$P_x = P_u \cup P_v \tag{18}$$
$$\epsilon(x) = \frac{1}{|P_x|} \sum_{v \in P_x} \epsilon(v) \tag{19}$$
$$\tag{20}$$

Procedure 1 Match nodes for coarsening.

Input: Hypergraph $H = (V, E)$ and corresponding weights $w_v\ \forall v \in V$ and $w_e\ \forall e \in E$. Node weight tolerance T.
Output: Set of matches (u, v) to be further coarsened.

$M_v \leftarrow 0\ \forall v \in V$
Sort $v \in V$ with respect to $S'(v)$ (Equation 12) in decreasing order.
for $v \in V$ in sorted order **do**
  if $M_v = 0$ **then**
    $p \leftarrow \emptyset$, $S_p \leftarrow -\infty$
    for $u \in \Gamma(v)$ **do**
      if $u \neq v$ **and** $w_u + w_v < T$ **and** $M_u = 0$ **then**
        $S_E \leftarrow \sum_{e \in E | u, v \in e} \frac{w_e}{|\{n \in e\}| - 1}$
        $S_\epsilon \leftarrow \epsilon(u)^\intercal \epsilon(v)$
        $S_{uv} \leftarrow \frac{S_E S_\epsilon}{w_u w_v}$
        if $S_{uv} > S_p$ **then**
          $p \leftarrow n$, $S_p \leftarrow S_{uv}$
    if $p \neq \emptyset$ **then**
      Match v with p for coarsening
      $M_v, M_p \leftarrow 1$

4 Experimental Design

In order to evaluate the partitioning quality of our proposed coarsening method, we implement our matching algorithm in both KaHyPar [11] and Zoltan [16]. Our KaHyPar implementation adds a new coarsening class to replace the existing community-based structure, and maintains other KaHyPar features such as its direct k-way initial solution. We evaluate this implementation with both traditional vertex-swapping refinement as well as more recent flow-based refinement [21]. In the case of Zoltan we introduce a new function to evaluate nodes during matching. Our implementation also requires minor modifications elsewhere in the software package in order to address re-indexing during recursive bisection. These changes do not effect the actual coarsening algorithm, as each call to recursive bisection begins with a subset of nodes and hyperedges from the original hypergraph.

In order to quantify the improvement in quality gained by embedding-based coarsening, we compute a number of partitions under a variety of scenarios. This begins with a set of embeddings. Due to resource constraints, we only embed each graph once for each considered technique and reuse this embedding in different runs. This compromise is necessary because graph embedding can more expensive than the considered multi-level hypergraph partitioners by orders of magnitude, determined often by the efficiency of the embedding software. Furthermore we note that the problem of embedding coarsened hypergraphs is nontrivial. We observe a significant decrease in overall solution quality when attempting to recompute embeddings at intermediate coarse levels, as the considered methods were not indented to capture to small weighted structures. Ultimately we find that this challenge lies outside the scope of this work.

The set of embedding techniques we explore consists of Node2Vec [22], Metapath2Vec++ [23], FOBE, and HOBE [19], as well as two combination embeddings (also presented in [19]). The first combination merges only FOBE and HOBE, while the second combination merges all four previously stated embeddings. All considered embeddings are in $\mathbb{R}^{100}$. While higher-dimensional embeddings have the ability to capture more complex latent structure, this complexity can also lead to poorer convergence while training. We performed an initial experiment comparing 100- to 500-dimensional embeddings of our hypergraph set, and observed no significant difference in solution quality. In addition, we do not claim to extensively test our coarsening against all state of the art embeddings, only that our proposed technique is robust to different embedding algorithms.

Each of the six input embeddings combines with each of the three proposed implementations, KaHyPar, KaHyPar Flow, and Zoltan, to create a set of eighteen proposed partitioners with embedding-based coarsening. We add to this five baseline methods: hMetis [24], Zoltan [16], PaToH [25], KaHyPar (with community-based coarsening [18]), and KaHyPar Flow (with both community-based coarsening and flow-based refinement [21]). This results in 23 considered partitioners. For each of the partitioners, we run separate trials optimizing for cut and $k-1$ respectively. The differences between these objectives is defined in detail in Section 2.

For all combination of partitioner and objective we additionally compare across a range of k-values. Many solvers identify a larger number of partitions through recursive-bisection (all considered except KaHyPar), which iteratively partitions the input hypergraph into two parts until reaching the desired number of partitions. For this reason we compare different numbers of partitions corresponding to the powers of two from 2 to 128. For each of these scenarios, we apply an overall imbalance tolerance of 3%. Then, for each combination of partitioner, objective, and k-value, we compare across a benchmark of hypergraphs.

Our benchmark consists of 86 sparse matrices selected from the SuiteSparse Matrix Collection [26]. These matrices span a range of domains including social networks, power grids, and linear systems. We interpret each matrix M as the incidence matrix of a hypergraph. In doing so, we consider each row to represent a node, each column to be a hyperedge, and a nonzero value in M_{ij} to indicate node j

participates in hyperedge i.

We additionally include ten synthetic hypergraphs that were designed to test the robustness of the coarsening process, extending a similar approach from graphs [38]. These graphs are a mixture of graphs that are weakly connected between each other, with less than 1% of edges connecting different graphs in the mixture. In multilevel setting, this can cause the coarsening process to incorrectly contract edges between different graphs in the mixture, resulting in uneven coarsening, overloaded refinement and worse quality of the final solution. This structure can be found in many real-world graphs, including multi-mode networks [51] and logistics multi-stage system networks [52]. We introduce additional complexity by adding additional $< 1\%$ random edges (denoted in the online appendix as "W/ Noise"). Full graphs, as well as scripts used to generate them are available in the online appendix.

Our overall benchmark suite of 96 graphs is explored in detail in the online appendix, wherein we present node and hyperedge distributions for all graphs. All names provided, except for our newly generated synthetic graphs, correspond to those found in the Sparse Matrix Collection.

For each combination of partitioner, objective, k-value, and graph, we compute twenty trials, with a total of over half-a-million trials. For each trial we generate a new random seed and randomly relabel the node and hyperedge indices.

In order to quantify the difference in quality between the two compared methods, we rely on summary statistics such as the macro-average of improvement. We define the "improvement" as a ratio between the partitioning quality of some baseline method against some comparison method. Note that if the comparison method achieves a cut or $k-1$ value that is lower than the baseline, the improvement will be greater than 1. We compute four different improvement statistics between two methods: average, minimum, maximum, and standard deviation. In this way we compare the expected, worst-case, and best-case observed partitioning quality, as well as the variance of results. In the following equations, $I_{(\cdot)}(X, Y, G, k, M)$ indicates the improvement between a baseline X and comparison Y algorithm for the k-partition problem on hypergraph G with respect to metric M. Let $T^{(i)}_{X,G,k,M}$ represent the i^{th} trial of algorithm X. For our experiments we run $\tau = 20$ of these trails.

$$I_{\text{avg}}(X, Y, G, k, M) = \frac{\mathbb{E}_{i=1,\ldots,\tau}\, T^{(i)}_{X,G,k,M}}{\mathbb{E}_{i=1,\ldots,\tau}\, T^{(i)}_{Y,G,k,M}} \tag{21}$$

$$I_{\text{min}}(X, Y, G, k, M) = \frac{\min_{i=1,\ldots,\tau} T^{(i)}_{X,G,k,M}}{\min_{i=1,\ldots,\tau} T^{(i)}_{Y,G,k,M}} \tag{22}$$

$$I_{\text{max}}(X, Y, G, k, M) = \frac{\max_{i=1,\ldots,\tau} T^{(i)}_{X,G,k,M}}{\max_{i=1,\ldots,\tau} T^{(i)}_{Y,G,k,M}} \tag{23}$$

$$\sigma(X, G, k, M) = \sqrt{\mathbb{E}[(T^{(i)}_{X,G,k,M})^2] - \mathbb{E}[T^{(i)}_{X,G,k,M}]^2} \tag{24}$$

$$I_{\sigma}(X, Y, G, k, M) = \frac{\sigma(X, G, k, M)}{\sigma(Y, G, k, M)} \tag{25}$$

We can then reduce the overall comparison of two methods with respect to a particular optimization metric into the macro-average of improvement across all graphs in the benchmark. Here, G' represents the set of considered benchmark.

$$I^{(M)}_{(\cdot)}(X, Y, k, M) = \frac{1}{|G'|}\left(\sum_{G \in G'} I_{(\cdot)}(X, Y, G, k, M)\right) \tag{26}$$

5 Results

Following the experimental procedure described in Section 4, we evaluate the partitioning quality of our proposed coarsening algorithm. Due to the volume of experimental trials performed for our evaluation, we can only present summary statistics for representative experiments in the body of this work. Full results are available online.

We present the macro-average improvement ($I^{(M)}_{\text{avg}}$) gained by the embedding-based coarsening for each implementation across all partitions counts in Table 1 (a and b) for the $k-1$ and cut metrics. These tables compare each implementation against its corresponding baseline without embedding-based coarsening. For instance, KaHyPar without flow-based refinement [18] is compared to the same KaHyPar without flow-based refinement but with the embedding-based coarsening.

Figure 2 depicts improvement of representative methods to KaHyPar with flow-based refinement, the top performing baseline [21]. Note that in these plots we use "EC" to refer to embedding-based coarsening. Additionally, in these tables and figures we select the FOBE embedding for both the KaHyPar and Zoltan implementations to represent overall embedding quality, as all considered embeddings perform similarly. Furthermore, Figure 3 depicts the relative improvement gained by embedding-based coarsening per-graph for the 2-partition problem. Our analysis focuses on insights that can be observed from these representative results. Overall we observe that embedding-based coarsening increases average quality across almost all considered comparisons. This improvement is typically greater for low partition counts.

In the case of KaHyPar, an improvement is the result of replacing the existing community-based coarsening [18]. The community-based coarsening, which is further discussed in Section 6, restricts coarsening partners to only nodes that share a community in the finest level of the input hypergraph. Our embedding-based coarsening is similar in the sense that node communities are likely to share similar embeddings. However, by introducing node embeddings we relax this constraint. As a result our approach gains in solution quality by occasionally merging across communities, which is particularly important when merging hubs or bridges that may border multiple communities.

When comparing the partitioning quality across all KaHyPar trials, we observe that KaHyPar with embedding-based coarsening but without flow-based refinement can find better partitioning solutions than KaHyPar with flow-based refinement. Specifically we observe an average improvement of 10% for the $k-1$ metric for $k = 2$ and $k = 4$, demonstrated in Figure 2. Furthermore, some graphs partitioned with embedding-based coarsening in Zoltan outperform even the latest KaHyPar version. Applications attempting to partition particularly large hypergraphs may benefit from this result as embedding-based coarsening and

# Parts(k):	2	4	8	16	32	64	128
KaHyPar	8%	13%	10%	6%	4%	3%	1%
KaHyPar(flow)	9%	11%	4%	2%	3%	2%	0%
Zoltan	48%	28%	15%	14%	9%	5%	3%

(a) Marginal avg. $k-1$ improvement.

# Parts(k):	2	4	8	16	32	64	128
KaHyPar	8%	16%	9%	1%	3%	1%	0%
KaHyPar(flow)	10%	11%	3%	1%	1%	1%	-1%
Zoltan	51%	45%	51%	41%	31%	14%	8%

(b) Marginal avg. cut improvement.

TABLE 1: The above tables summarize the average increase in quality that can be gained per-metric and per-method when utilizing embedding-based coarsening. Each method is compared against its corresponding baseline, such as comparing KaHyPar (flow) with and without embedding-based coarsening. All quality numbers come from an average of all trials using the FOBE embedding and $I^{(M)}_{\text{avg}}$ metric.

($\log n$)-level partitioners expose more parallelism than the n-level KaHyPar design.

For small k this flexibility appears to be the most valuable, as these latent embedding spaces may only detect a handful of relevant ground-truth clusters. For higher k, the number of partitions appears to exceed the number of natural divisions in our embedding space. Our KaHyPar implementation particularly struggles here, which we observe is likely to result from the direct k-way initial solution this method identifies [53]. In contrast, Zoltan, which uses recursive-bisection to recursively 2-partition the input hypergraph, is more resilient for larger k. Recursive bisection has the effect of splitting the input embedding into two subspaces. When combined with embedding-based coarsening, these subspaces divide the key axes of variance within the embedding spaced. Then, the next iteration need only consider locally relevant differences within each respective subspace, which retains more locally-relevant information. This effect is what keeps Zoltan's partitioning quality competitive with KaHyPar flow for larger k, as seen in Figure 2.

Examining the standard deviation results shown in Figure 2, we observe that embedding-based coarsening decreases the standard deviation of possible results for a given hypergraph. These figures corresponding to the standard deviation of both the $k-1$ and cut metrics demonstrate that the macro-average improvement of standard deviation is often substantial, and occasionally over an order of magnitude. This result comes from replacing the typically random node-visit order with a sorted ordering dependent on each node's nearest-neighbor. If addition, the figures corresponding to the minimum and maximum $k-1$ and cut observed pet-trial all demonstrate that embedding-based coarsening consistently improves the expected worst-case ($I^{(M)}_{\text{max}}$) and average best-case ($I^{(M)}_{\text{min}}$) quality. Many applications run multiple partitioning trials and select the top-performing result [54], however by reducing the variance of results to an improved range, our coarsening approach could improve overall application efficiency.

Looking into the graph-wise results, shown in Figure 3, we observe that there is a class of hypergraphs that are best aided by embedding-based coarsening. We observe that embedding-based coarsening can identify partitions with 200-400% improvement in graphs with rich latent structure, such as the communication networks corresponding to the Enron or European Union email networks (as found in [26]). Additionally, some synthetic graphs, those constructed through a star-shaped merge of multiple real-world networks and are designed to complicate the coarsening process, are similarly improved. These improvements on this class of graphs is also highly statistically significant ($p < 0.01$). These highly-improved graphs have rich latent global structure that may not be accurately captured through hyperedge-wise features. For example, the departmental structure within Enron is lost when individually considering emails between particular employees. While not every graph can be exploited to that magnitude, we do observe that a significant portion of our benchmark is significantly improved, and a further portion is merely unchanged. We do however observe a subset of graphs that are partitioned worse with embedding-based clustering. Nemsemm2, a sparse matrix corresponding to a linear program, is partitioned almost three-times worse using embedding-based coarsening. The incidence matrix of this hypergraph is nearly block-diagonal, which results in significant hyperedge-wise features that are not translated into an embedding, as disjoint graph regions are often embedded in overlapping spaces. In contrast, Nemswrld is another linear-program sparse matrix published by the same group, but is less block-diagonal and receives an statistically significant average improvement of about 33%. Each of these above results refer to a 2-partition performed by KaHyPar (Figure 4.

6 Related Work

Our proposed embedding-based coarsening is similar to the community-based coarsening proposed by Heuer et al. and used in our KaHyPar baseline [18]. Their approach begins with a "community detection phase" wherein traditional community detection algorithms cluster the nodes contained in the bipartite star-expansion of the original graph. From there, the coarsening process is restricted to only contract nodes within a community. This approach is intended to maintain global community structure from the original hypergraph in the final coarsest representation. While both methods leverage the bipartite representation to find initial node features, embedding-based coarsening improves upon community-based coarsening by relaxing the requirement that nodes can only be coarsened within a community. Nodes within a modularity maximizing community are internally dense and externally sparse [55]. As a result nodes sharing a community are more likely to co-occur in any local sampling strategy employed by a graph embedding algorithm. Therefore, it is likely that the natural clusters within our considered graph embeddings are similar to the communities found by KaHyPar. However, these embeddings inform additional global relationships between clusters that are lost when each community is coarsened

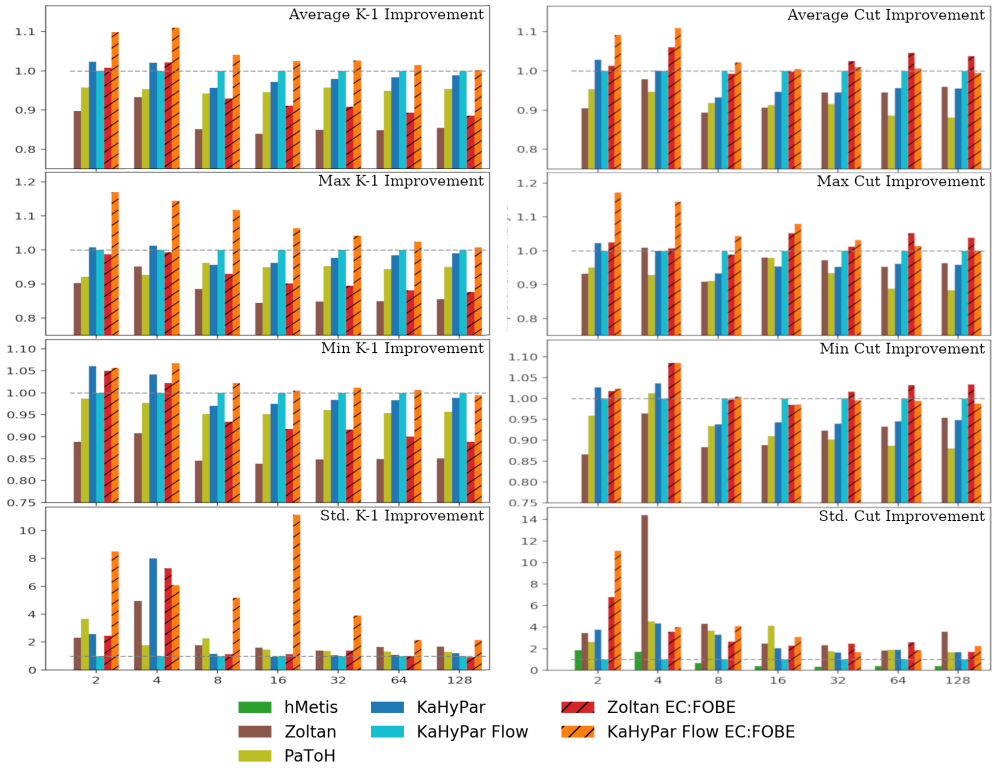

Fig. 2: Above depicts $I_{\text{avg}}^{(M)}$ (26) using KaHyPar Flow as the baseline (M) for representative considered metrics.

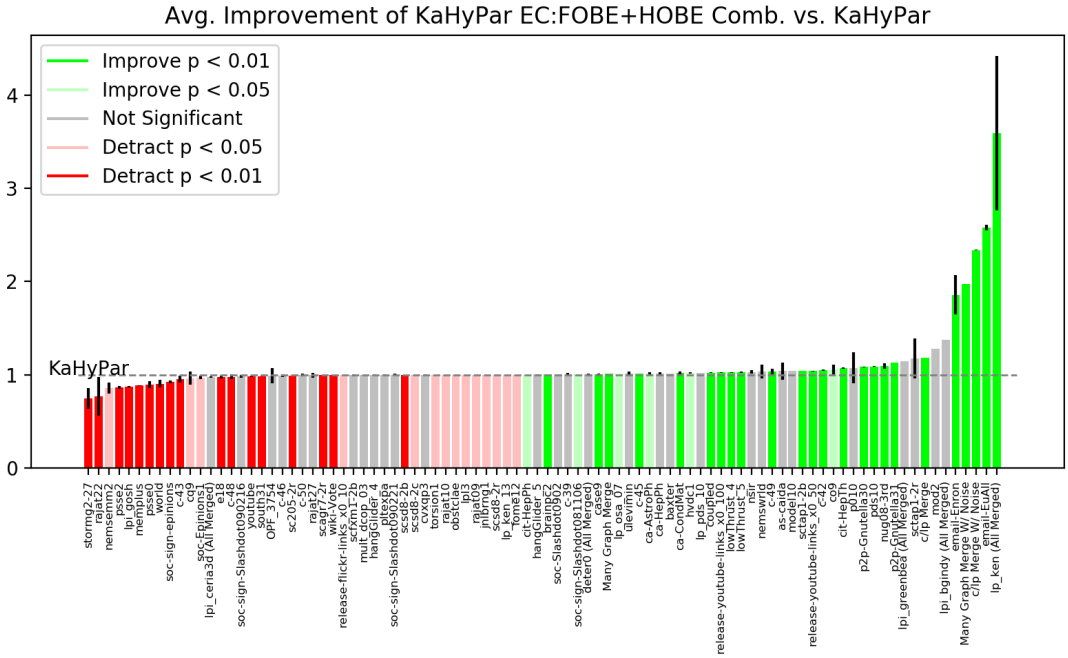

Fig. 3: Above depicts the improvement of the $k-1$ metric in KaHyPar when neural coarsening is applied to the 2-partition problem. Each bar represents a comparison of 20 baseline and 20 embedding-based trails for a single graph. The color of each bar represents the statistical significance between the sets of trial results. The small black lines represent the standard deviation of the embedding-based method, and the absence of a bar indicates a standard deviation near zero. Note that the graphs with the most improvement primarily social networks. Bar heights correspond to the I_{avg} statistic.

independently. For instance, nodes on the boundary of two communities will likely receive embeddings spatially located between two clusters. This distinction is able to remain in the coarsest representation of the hypergraph, and may be lost in community-based coarsening when nodes are initially split due to community assignments.

Memetic partitioning, also proposed for KaHyPar, uses the principles of genetic algorithms to discover improved partitioning solutions [12]. This approach creates high quality partitions by iterating through different "generations" of solutions, starting with an initial generation produced by KaHyPar run multiple times with different seeds. From the initial set, multiple combination operators "breed" new solutions by combining some number of "parents" to form new solutions. Each iteration is designed to improve the population's average $k-1$ metric. Combination operators are specifically posed such that offspring solutions perform at least as good as its corresponding parents. While this approach is demonstrated to improve overall hypergraph partitioning quality, it does so by adding a meta process to the set of initial hypergraph solutions. We anticipate that adding embedding-based coarsening as a method for generating a high quality initial solution population may be a complimentary way to improve the overall process.

The proposed embedding-based coarsening extends the relaxation-based coarsening developed by Shaydulin et al. [13] in Zoltan. This work introduces *algebraic distance for hypergraphs*, which in turn extends a similar measure designed for traditional graphs [34]. Algebraic distance is a similarity measure that takes into account distant neighborhoods of vertices, enabling the coarsening process to exploit the global structure of highly irregular hypegraphs. Algebraic distance is computed by an iterative process that is shown to stabilize quickly [13], requiring only tens of iterations to obtain rich latent features. As such, this method is additionally found within the Algebraic Heterogeneous Bipartite Embeddings we consider (AHBE) [19]. However, as uncovered in that work, neural graph embeddings can learn additional latent features not often captured by algebraic distance alone.

Aggregative coarsening [35] uses ideas from algebraic multigrid, extending an unfinished attempt published in Sandia Summer Reports [37]. At each step of the coarsening process a set of seed vertices is selected. Each seed then becomes a center of an aggregate, with non-seeds assigned to seeds using different aggregation rules. An aggregate at finer level forms a vertex at coarser level. Two aggregation rules, based on inner product matching and stable matching were explored. Our embedding-based coarsening can be used within the aggregative coarsening to inform the aggregation rules.

7 Conclusion

In this work we propose embedding-based coarsening, an approach that uses latent features present in a pretrained hypergraph embedding to better solve the hypergraph partitioning problem. We do so by prioritizing nodes that share many latent features during the coarsening process, and then leveraging a combination of traditional and embedding-derived features when determining coarsening partners. We evaluate this approach over multiple trials per combination of 96 graphs, 7 partition counts, 6 pretrained embedding methods, 5 baseline partitioners, 3 implementations, and 2 objective functions. We observe a significant increase in quality for small values of k (from 2 until about 16) gained from embedding-based coarsening. For higher values of k we observe overall quality that returns to the state-of-the-art baseline. All experiments, plots and code are available in our online appendix at sybrandt.com/2019/partitioning.

An important future research direction is related to the embedding-based coarsening for large k as the improvement we observe is less significant. One potential explanation is that our fixed sized embeddings only contain a relatively small number of latent clusters. This would imply that beyond certain small k, most coarsening comparisons will occur within a single cluster, wherein all nodes are similar. However, we demonstrate that using the proposed embedding-based coarsening one can improve the solution quality of existing hypergraph paritioners by about 10% for small k, and up to 400% on particular graphs with rich latent structure. For example, this method increases the quality of Zoltan above that of KaHyPar with flow-based refinement in some cases, which is particularly important as then $\log n$-level paradigm implemented in Zoltan exposes substantially more parallelism than the n-level counterpart. We also note that our algorithm is embedding-agnostic and is ready to incorporate other types of embeddings that can potentially work better for specific types of instances.

8 Acknowledgements

We would like to thank Sebastian Schlag from the Karlsruhe Institute of Technology for helping us to understand KaHyPar. This work was supported by NSF awards MRI #1725573, DMS #1522751, and NRT #1633608.

References

[1] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in vlsi domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, 1999.

[2] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in neural information processing systems*, 2007, pp. 1601–1608.

[3] M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram, "The total variation on hypergraphs-learning on hypergraphs revisited," in *Advances in Neural Information Processing Systems*, 2013, pp. 2427–2435.

[4] C. Zhang, S. Hu, Z. G. Tang, and T. Chan, "Re-revisiting learning on hypergraphs: confidence interval and subgradient method," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 4026–4034.

[5] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on parallel and distributed systems*, vol. 10, no. 7, pp. 673–693, 1999.

[6] U. Naumann and O. Schenk, *Combinatorial scientific computing*. CRC Press, 2012.

[7] M. A. Shepherd, C. R. Watters, and Y. Cai, "Transient hypergraphs for citation networks," *Information Processing & Management*, vol. 26, no. 3, pp. 395–412, 1990.

[8] Z.-K. Zhang and C. Liu, "A hypergraph model of social tagging networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2010, no. 10, p. P10005, 2010.

[9] T. Lengauer, *Combinatorial algorithms for integrated circuit layout*. Springer Science & Business Media, 2012.

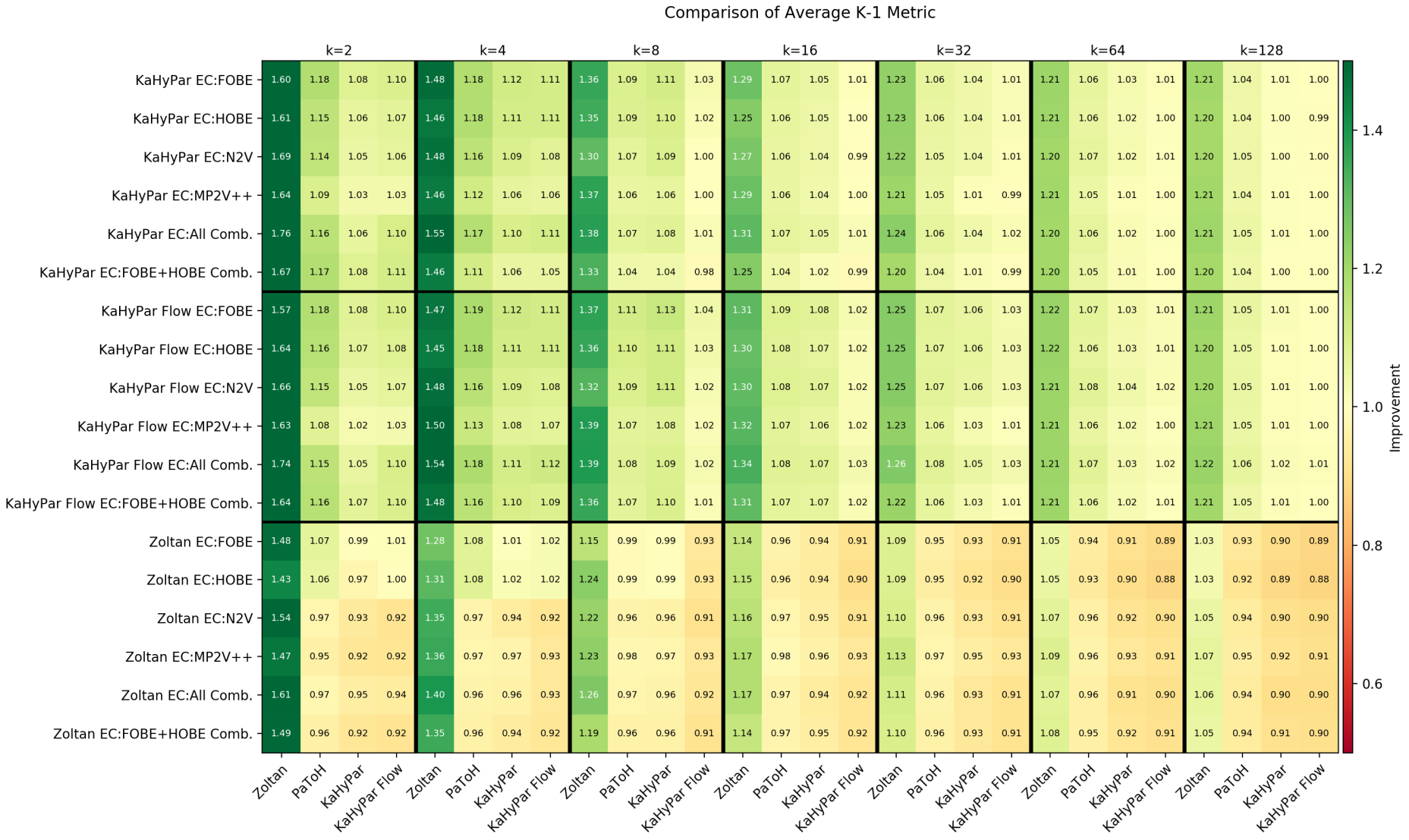

Fig. 4: Macro-average improvement of the $k-1$ metric across all considered graphs and methods. We performed 20 partitions per-graph per-method using different seeds. Additional result matrices available online.

[10] T. N. Bui and C. Jones, "Finding good approximate vertex and edge partitions is np-hard," *Information Processing Letters*, vol. 42, no. 3, pp. 153–159, 1992.

[11] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz, "k-way hypergraph partitioning via *n*-level recursive bisection," in *18th Workshop on Algorithm Engineering and Experiments, (ALENEX 2016)*, 2016, pp. 53–67.

[12] R. Andre, S. Schlag, and C. Schulz, "Memetic multilevel hypergraph partitioning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '18, 2018, pp. 347–354.

[13] R. Shaydulin, J. Chen, and I. Safro, "Relaxation-based coarsening for multilevel hypergraph partitioning," *Multiscale Modeling & Simulation*, vol. 17, no. 1, pp. 482–506, 2019.

[14] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," *VLSI design*, vol. 11, no. 3, pp. 285–300, 2000.

[15] E. G. Boman, U. V. Catalyurek, C. Chevalier, K. D. Devine, I. Safro, and M. M. Wolf, "Advances in parallel partitioning, load balancing and matrix ordering for scientific computing," in *Journal of Physics: Conference Series*, vol. 180, no. 1. Institute of Physics Publishing, 2009, pp. 12 008–12 013.

[16] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 10–pp.

[17] C. Chevalier and I. Safro, "Comparison of coarsening schemes for multilevel graph partitioning," *Learning and Intelligent Optimization*, pp. 191–205, 2009.

[18] T. Heuer and S. Schlag, "Improving coarsening schemes for hypergraph partitioning by exploiting community structure," in *16th International Symposium on Experimental Algorithms, (SEA 2017)*, 2017, pp. 21:1–21:19.

[19] J. Sybrandt and I. Safro, "First-and High-Order Bipartite Embeddings," *submitted, arXiv preprint arXiv:1905.10953*, 2019.

[20] S. Agarwal, K. Branson, and S. Belongie, "Higher order learning with graphs," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 17–24.

[21] T. Heuer, P. Sanders, and S. Schlag, "Network Flow-Based Refinement for Multilevel Hypergraph Partitioning," in *17th International Symposium on Experimental Algorithms (SEA 2018)*, 2018, pp. 1:1–1:19.

[22] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[23] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 135–144.

[24] G. Karypis, "hmetis 1.5: A hypergraph partitioning package," *http://www. cs. umn. edu/~ metis*, 1998.

[25] Ü. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," *Encyclopedia of Parallel Computing*, pp. 1479–1487, 2011.

[26] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[27] S. T. Barnard and H. D. Simon, "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency and computation: Practice and Experience*, vol. 6, no. 2, pp. 101–117, 1994.

[28] A. Brandt and D. Ron, "Chapter 1 : Multigrid solvers and multilevel optimization strategies," in *Multilevel Optimization and VLSICAD*, J. Cong and J. R. Shinnerl, Eds. Kluwer, 2003.

[29] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[30] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Engineering: Selected Results and Surveys. LNCS 9220, Springer-Verlag*. Springer, 2016, pp. 117–158.

[31] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM review*, vol. 47, no. 1, pp. 67–95, 2005.

[32] C.-E. Bichot and P. Siarry, *Graph partitioning*. Wiley Online Library, 2011.

[33] A. Trifunović and W. J. Knottenbelt, "Parallel multilevel algorithms for hypergraph partitioning," *Journal of Parallel and Distributed Computing*, vol. 68, no. 5, pp. 563–581, 2008.

[34] J. Chen and I. Safro, "Algebraic distance on graphs," *SIAM Journal on Scientific Computing*, vol. 33, no. 6, pp. 3468–3490, 2011.

[35] R. Shaydulin and I. Safro, "Aggregative coarsening for multilevel hypergraph partitioning," *17th International Symposium on Experimental Algorithms (SEA 2018)*, 2018.

[36] I. Safro, D. Ron, and A. Brandt, "Multilevel algorithms for linear ordering problems," *Journal of Experimental Algorithmics (JEA)*, vol. 13, p. 4, 2009.

[37] A. Buluc and E. G. Boman, "Towards scalable parallel hypergraph partitioning," in *CSRI Summer Proceedings 2008*. Sandia National Labs, 2008, pp. 109–119.

[38] I. Safro, P. Sanders, and C. Schulz, "Advanced coarsening schemes for graph partitioning," *ACM Journal of Experimental Algorithmics (JEA)*, vol. 19, pp. 2–2, 2015.

[39] I. Safro, D. Ron, and A. Brandt, "Graph minimum linear arrangement by multilevel weighted edge contractions," *J. Algorithms*, vol. 60, no. 1, pp. 24–41, 2006.

[40] D. Ron, I. Safro, and A. Brandt, "Relaxation-based coarsening and multiscale graph organization," *Multiscale Modeling & Simulation*, vol. 9, no. 1, pp. 407–423, 2011.

[41] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Papers on Twenty-five years of electronic design automation*. ACM, 1988, pp. 241–247.

[42] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.

[43] P. Sanders and C. Schulz, "Engineering multilevel graph partitioning algorithms," in *European Symposium on Algorithms*. Springer, 2011, pp. 469–480.

[44] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[45] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[46] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[47] Y. Tsvetkov, M. Faruqui, W. Ling, G. Lample, and C. Dyer, "Evaluation of word vector representations by subspace alignment," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 2049–2054.

[48] A. Gladkova, A. Drozd, and S. Matsuoka, "Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't." in *Proceedings of the NAACL Student Research Workshop*, 2016, pp. 8–15.

[49] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[50] E. John and I. Safro, "Single-and multi-level network sparsification by algebraic distance," *Journal of Complex Networks*, vol. 5, no. 3, pp. 352–388, 2016.

[51] L. Tang, H. Liu, J. Zhang, and Z. Nazeri, "Community evolution in dynamic multi-mode networks," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 677–685.

[52] L. Stock, *Strategic Logistics Management*, ser. Cram101 Textbook Outlines. Lightning Source Inc, 2006. [Online]. Available: http://books.google.com/books?id=1LyCAQAACAAJ

[53] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, "Engineering a direct *k*-way hypergraph partitioning algorithm," in *19th Workshop on Algorithm Engineering and Experiments, (ALENEX 2017)*, 2017, pp. 28–42.

[54] A. Trifunovic, "Parallel algorithms for hypergraph partitioning," Ph.D. dissertation, University of London, 2006.

[55] M. Newman, *Networks: an introduction*. Oxford university press, 2010.