Hypergraph Partitioning With Embeddings

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

Abstract—Problems in scientific computing, such as distributing large sparse matrix operations, have analogous formulations as hypergraph partitioning problems. A hypergraph is a generalization of a traditional graph wherein "hyperedges" may connect any number of nodes. As a result, hypergraph partitioning is an NP-Hard problem to both solve or approximate. State-of-the-art algorithms that solve this problem follow the multilevel paradigm, which begins by iteratively "coarsening" the input hypergraph to smaller problem instances that share key structural features. Once identifying an approximate problem that is small enough to be solved directly, that solution can be interpolated and refined to the original problem. While this strategy represents an excellent trade off between quality and running time, it is sensitive to coarsening strategy. In this work we propose using graph embeddings of the initial hypergraph in order to ensure that coarsened problem instances retrain key structural features. Our approach prioritizes coarsening within self-similar regions within the input graph, and leads to significantly improved solution quality across a range of considered hypergraphs.
Reproducibility: All source code, plots and experimental data are available at https://sybrandt.com/2019/partition.

1 Introduction

Hypergraphs provide the formalism needed to solve problems consisting of interconnected item sets. Similar to a traditional graph, the hypergraph has the added generalization that "hyperedges" may connect any number of nodes. Domains such as very-large-scale integration for creating integrated circuits [1], machine learning [2], [3], [4], parallel algorithms [5], combinatorial scientific computing [6], and social network analysis [7], [8] all contain significant and challenging instances of hypergraph problems. One important problem, *Hypergraph partitioning*, involves dividing the nodes of a hypergraph among k similarly-sized disjoint sets while reducing the number of hyperedges that span multiple partitions. In the context of load balancing, this is the problem of dividing logical threads (nodes) that share data dependencies (hyperedges) among available machines (partitions) in order to balance the number of threads per machine and minimize communication overhead. However, hypergraph partitioning is both NP-Hard to solve [9] and approximate [10].

Therefore, state-of-the-art partitioners apply heuristically-backed algorithms to overcome these inherent computational limitations [11]. The most common and effective technique is the *multilevel paradigm* [1], [12], [13], [14], [15]. The multilevel paradigm is well known to be successful beyond the (hyper)graph partitioning in such areas as the cut-based problems on graphs [16] and machine learning [17]. Multilevel partitioners consist of three phases, referred to collectively as the *V-Cycle*: coarsening, the initial solution, and uncoarsening. We depict these phases in Figure 1. The overarching idea behind this technique is to find a problem instance that shares key structural features with the input hypergraph, but is small enough to be partitioned directly. The initial solution to this small analogous problem can then be gradually interpolated and refined to apply to the input hypergraph.

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

Manuscript received 9 Sept. 2019.

The small analogous problem is identified through an iterative *coarsening* process consisting of many levels. At each level, groups of similar nodes are identified, and each is "contracted" into a single merged node at the next more-coarse level. While grouping nodes, the goal is to identify self-similar regions of the current hypergraph so that the more coarse problem instances retain key structural features. Most commonly, these coarsening groups are formed by pairing nodes due to a similarity measure [13], [15] that heuristically or rigorously suggests to place both nodes in the same partition. An n-level algorithm is one that identifies only one pair of coarsening partners at each level [18], while a $\log n$-level algorithm pairs many nodes each time [15]. Coarsening stops once identifying a sufficiently small hypergraph based on some criterion that indicates that this problem is possible to (almost) optimally solve on given computational resources. The *initial solution* can then be identified directly using the best available algorithm. Now, the solution is *uncoarsened* back through the levels in order to identify a solution to the original problem. Uncoarsening consists of three sub-phases: expansion, interpolation, and refinement. Expansion undoes the coarsening at a given level by "expanding" the current level's coarsened nodes with those contracted in the prior. Next, interpolation assigns each expanded node the partition label assigned to their corresponding coarse representation. Then, local refinement cheaply updates the partition labels among the expanded nodes in order to improve the overall solution quality for the next level. This process is repeated from the initial solution through all coarsening levels and back to the original hypergraph, which final refinement solution is accepted as the solution to the partitioning problem.

Because the strategy used to contract nodes determines the coarsening at each level, the quality of the initial solution, and the behavior of interpolation and refinement during uncoarsening, we find that this single factor can dramatically effect partitioning quality. Other works exploring coarsening strategies, such as relaxation-based [13] or community-aware [19] coarsening, arrive with a similar conclusion.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3017120, IEEE Transactions on Knowledge and Data Engineering

1.1 Our Contribution

We propose *embedding-based coarsening*, a novel coarsening strategy that leverages graph embeddings to prioritize the contraction of self-similar regions of the input hypergraph in order to retain global structural features. This approach augments the existing strategy that contracts nodes based on their co-participation in small hyperedges by adding an embedding-based term that can break ties among similarly ranked coarsening pairs. A toy example of this phenomena is depicted in Figure 2, wherein three potential coarsening pairs are equally ranked by the traditional scheme, but embedding-based signals favor the pair that retains both key clusters.

The field of graph embedding is evolving rapidly, and the proposed embedding-based coarsening is designed to be agnostic with respect to any particular technique, provided that similarities between nodes are encoded via the dot product of embedding vectors. Specifically, our proposed technique accepts a precomputed embedding as an auxiliary input per-hypergraph, and we demonstrate that a wide range of existing embedding techniques improve partitioning performance similarly. Decoupling partitioning from embedding enables embedding-based coarsening to more easily benefit from future advances in machine learning techniques. In order to apply embedding techniques designed for classical graphs, we need a classical representation of each input hypergraph. The star-expansion [20] represents a hypergraph as an undirected bipartite graph wherein hyperedges from the original structure form a new layer of nodes. An edge between two nodes i and j in the bipartite structure indicates that node i participated in hyperedge j within the original structure. As opposed to other classical representations like the clique-expansion, the star-expansion retains all relevant hypergraph information, and is scalable for large graphs [20]. Furthermore, existing embedding techniques specifically designed for bipartite graphs [21] apply to star-expanded graphs directly.

Given an input hypergraph and an embedding for each node of the input structure, embedding-based coarsening follows this outline at each coarsening level. First, each node is assigned a score equal to the highest dot product between its embedding and each of its neighbors. Nodes are visited in decreasing order by score. A visited node is matched from among its neighbors based on the product of their classical edge-wise score, and the dot product of each node's embedding. After matching nodes, based on whether we are performing n- or $\log n$-level coarsening, mated nodes are contracted. Newly coarsened nodes are assigned an embedding equal to the average embedding of all initial embeddings contained within the coarse representation.

We implement our proposed coarsening strategy in both KaHyPar [18], which is a n-level partitioner with state-of-the-art solution quality, as well as Zoltan [15], which is a parallel $\log n$-level partitioner with high quality and state-of-the-art speed. Furthermore, we compare the effect of various different embedding techniques, including Node2Vec [22], Metapath2Vec++ [23], and FOBE/HOBE [21], which were designed specifically for bipartite graphs. We additionally compare the effect of each embedding-based coarsening strategy with hMetis [24], Zoltan [15], PaToH [25], KaHyPar (with community-based coarsening [19]), and KaHyPar Flow (with both community-based coarsening and flow-based refinement [26])[1]. We compare performance of each partitioner across 96 hypergraph from the SuiteSparse Matrix Collection [27]. For each graph, we compute one embedding using each of the proposed techniques to serve as an auxiliary input across all trial. We compare quality across both the "cut" and "connectivity" objectives as well as for partition counts from 2 to 128 for each partitioner. For each combination of experimental parameters we run 20 trials in order to compare the variance and establish significance with respect to different random seeds. Overall, we produce over 500,000 individual trials.

We find that embedding-based coarsening has a significant improvement over the state of the art that is especially pronounced for smaller partition counts (< 32). In some cases, this leads to a solution quality that is improved by as much as 400%. Because embedding-based coarsening replaces the traditionally random visit order with one that prioritizes self-similar regions of the hypergraph, we also observe an improvement in the standard deviation of quality. All experimental code, data, visualization scripts, and a database of all experimental results, including all hyperparameters per-trial, can be found at: https://sybrandt.com/2019/partition. A longer-form version of this work can be found online[2].

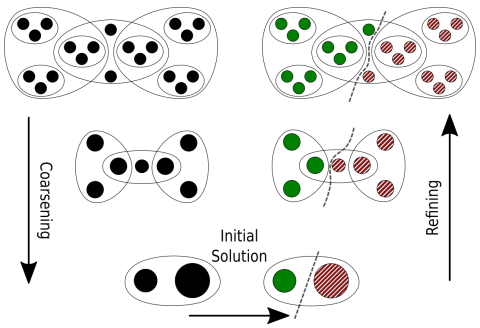


Fig. 1: A standard V-cycle, consisting of coarsening, and initial partition, and uncoarsening.

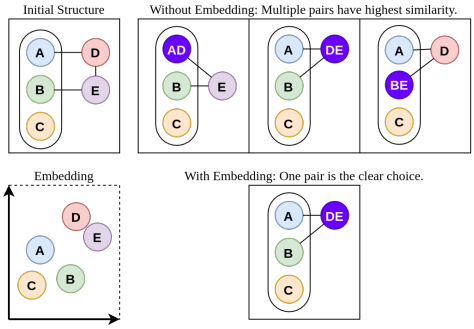


Fig. 2: Typical heavy-edge coarsening results in three possible contractions of equal weight. Embedding-based coarsening breaks the tie for the pair that preserves global structure — a cluster of weight 3 and of weight 2.

1. Neither hMetis nor PaToH provide source code. Instead, we can only use pre-compiled binaries for comparison purposes.
2. https://arxiv.org/abs/1909.04016

2 Notation and Preliminary Concepts

A hypergraph $H = (V, E)$ consists of nodes $v \in V$ and hyperedges $e \in E$. As opposed to a traditional graph, each hyperedge may contain any non-empty subset of V. The hypergraph partitioning problem is to divide V into k disjoint subsets of similar size while minimizing a given objective function. Two common objectives considered here are "cut" and "connectivity." Cut measures the number of hyperedges spanning more than one partition. If $\lambda(e)$ is the number of partitions spanned by edge e, then the cut objective is defined as: $|e \in E, \text{ such that } \lambda(e) > 1|$.

The "connectivity" objective, also commonly referred to as "$k-1$," penalizes each edge by the number of spanned partitions, namely, $\sum_{e \in E}(\lambda(e) - 1)$. In the case of $k = 2$, this is equivalent to cut.

Weights. Although we consider unweighted input hypergraphs (all nodes and hyperedges count the same towards their corresponding objectives and constraints), the multilevel paradigm introduces weights to intermediate level hypergraphs. Each node and hyperedge has a corresponding weight (w_v and w_e) equal to one for the input hypergraph. During coarsening, if two nodes v_i and v_j are contracted into a new coarse node v', then $w_{v'} = w_{v_i} + w_{v_j}$. This new node v' will also be added to all edges originally containing either v_i or v_j, before removing those original nodes from the resulting coarse hypergraph. If during the coarsening two edges will contain the same subset of nodes ($e_1 = e_2$), they will be replaced with a new edge e' containing the same nodes but with added weights: $w_{e'} = w_{e_1} + w_{e_2}$. When solving for cut and connectivity for intermediate subproblems during the multilevel strategy, we introduce these weights into the objective:

$$\text{weighted cut} = \sum_{e \in E, \lambda(e) > 1} w_e \tag{1}$$

$$\text{weighted connectivity} = \sum_{e \in E} (\lambda(e) - 1) w_e \tag{2}$$

One issue during coarsening is the potential for individual nodes to accumulate a disproportionate amount of weight. When this occurs, balanced partitioning can become impossible at the coarsest level, especially if one node's weight exceeds $|V|/k$. In order to avoid this negative effect, multilevel partitioners enforce a weight tolerance $w^{(T)}$, which is parameterized by the user. No coarsening partners may be contracted if their resulting coarse node would exceed this limit.

Imbalance Constraint. It is important to balance the number of nodes in the resulting partitions. Therefore, partitioners include an optimization constraint to determine how uneven the resulting partitions are allowed to be. For each partition $V_i \subset V$, given a predefined imbalance tolerance α, this constraint is defined as:

$$\sum_{v' \in V_i} w_{v'} \leq (1 + \alpha) \left\lceil \frac{1}{k} \sum_{v \in V} w_v \right\rceil \tag{3}$$

Embeddings. We use the function $\epsilon : V \rightarrow \mathbb{R}^n$ to denote a pre-trained embedding. Conceptually, this is a lookup table that assigns a node in the input hypergraph to a real-valued n-dimensional vector. In our experiments we select $n = 100$. Note, higher-dimensional embeddings show no statistically significant performance difference in our experiments.

3 Background and Related Work

Multilevel Partitioning.

The multilevel paradigm is the state-of-the-art solution strategy for hypergraph partitioning [11]—both improving result quality [28] and runtime [29]. Multilevel algorithms follow the V-cycle pattern, consisting of coarsening, an initial solution, and uncoarsening. This approach is effective because coarse hypergraphs are easier to solve yet they retain global structural features of the original. Coarse hypergraphs are created by iteratively merging multiple nodes at the current "finer" level into single nodes at the "coarser" level. Once sufficiently small, a partitioner can directly solve the coarsest problem instance using an algorithm that would normally be infeasible for large problems. Uncoarsening then interpolates the initial solution through each coarse level, performing local search at each level to refine the solution, until a solution is presented for the initial problem instance.

Usually, at each level of the coarsening process almost all nodes have at least one merging partner, resulting in $\log n$ levels. This is the approach used by Mondriaan [30], hMetis2 [1], Zoltan [15], and PaToH [25]. However, KaHyPar [18] implements an n-level approach where at each level only one pair of nodes is contracted.

Coarsening Strategies. A good coarsening strategy is one that groups together nodes that will ultimately share the same partition label, meaning that the coarser solution can be interpolated to the finer solution without a loss of quality. In practice, this loss of quality is to be expected, which is why local refinement is common during uncoarsening. However, if global structural features are not preserved, the loss of quality during interpolation cannot be rectified through the fast local refinement process. Therefore, the choice of coarsening heuristic is paramount.

Most heuristics used to identify nodes for contraction do so by scoring node pairs, and most partitioners, including Mondriaan [30], hMetis2 [1] and Zoltan [15], measure the edge-wise inner product, or some variation. The edge-wise inner-product is the Euclidean inner product of the weighted hyperedge incidence vectors [15]. Edge weights are defined formally in Section 2. Specifically, if w_e is the weight of hyperedge e, then the edge-wise inner product of nodes u and v is defined as:

$$\sum_{u,v \in e \in E} w_e$$

Although this approach is simplistic, it is also very computationally inexpensive and has provided a firm baseline. As mentioned, many variations exist, such as *absorption*, implemented in PaToH [25], and *heavy edge*, implemented in hMetis2 [1], Parkway [31], and KaHyPar [19], as well as a number of other normalization techniques, often based on node or hyperedge degree. Heavy edge, which is of particular interest due to its simple formulation and high performance, simply normalizes hyperedge weight by the expected degree of the resulting hyperedge following con-

traction. If $|e|$ is the number of nodes present in hyperedge e, then this score is:

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

One key limitation to the edge-wise score heuristics is that each only considers local information around each node. Therefore, global structural features can be collapsed during coarsening. This work seeks to use graph embeddings to provide this global information, however prior work has attempted to provide similar signals in alternate ways. Shaydulin et al. introduce *algebraic distance* for hypergraphs, a relaxation-based similarity measure that extends a similar approach from traditional graphs [32], [33]. This measure treats nodes as entities in a mutually-reinforcing environment, which enables this technique to apply a fast relaxation-based approach to supply a coordinate per node. Conceptually this acts as a one-dimensional embedding, wherein two nodes receive a similar coordinate if their neighborhoods are similar. This similarity measure is used to quantify node similarities and assign weights to hyperedges.

Another approach to incorporate global information is *community-aware* coarsening, which uses clustering information to restrict matching between communities. This approach, which is implemented in KaHyPar, makes the assumption that nodes belonging to different clusters of the input hypergraph should never be contracted. The proposed clustering is performed by a fast global modularity-maximizing algorithm, leveraging the connection between partitioning and clustering. This modularity-based clustering, which groups star-expanded nodes within a bipartite representation of a hypergraph, identifies communities are internally dense and externally sparse [34], which is desired for a good partitioning. We note, and discuss further in Section 7, that the clusters found by this modularity-maximizing approach are similar to the self-similar regions within a graph embedding. However, in some scenarios the hard restriction to never merge nodes across communities appears to be too strong. Instead, embedding-based coarsening simply penalizes the contraction of nodes across clusters, allowing more flexible decisions for nodes along the periphery.

Refinement. While this work proposes a new coarsening strategy, important work also explores the refinement stage of uncoarsening, wherein each partitioner performs local search in order to improve the interpolated coarser solution on the finer level. The typical strategy is the *node-moving heuristic*, wherein each expanded node at the newly refined level is given the option of switching partition label. A majority of hypergraph partitioners use a variation of Fiduccia-Mattheyses [35] or Kernighan-Lin [36] to perform these local searches [1], [15], [25], [26], [30], [31]. Recently, Heuer et al. introduced a flow-based refinement scheme for k-way hypergraph partitioning [26], extending similar approaches from graph partitioning [37]. This flow-based refinement, which is implemented in KaHyPar, is considered as a temperate case within our benchmark. As a result, we can compare the performance of embedding-based coarsening without flow-based regiment, and vice-versa.

Additional Partitioning Strategies. There are a few coarsening and partitioning strategies that are not included in our benchmark, but are worth additional discussion. *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 connectivity 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. *Aggregative coarsening* [38] uses ideas from algebraic multigrid. 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.

3.1 Graph Embeddings

Our embedding-based coarsening accepts embeddings for each node of the input hypergraph as an auxiliary input. Rather than depend on a particular graph embedding technique, this work simply assumes that some measure of global graph structure is encoded via the dot product of embeddings, meaning that two nodes with a higher dot product of embeddings will be more similar. In this manner, new advances in graph embedding, or fine-tuned versions of existing algorithms for particular graphs, can be introduced into our proposed strategy. Importantly, this work does not seek to establish any graph embedding technique as inherently better for hypergraph partitioning. *Instead, we find that all considered embeddings greatly improve solution quality*.

Neural Graph Embedding. The Deepwalk graph embedding [39], which applies the skip-gram model [40] to random walks of nodes, marks the beginning of neural network graph embeddings. The node2vec approach [22] modifies Deepwalk to parameterize random walk behavior, allowing walks to explore local regions or broad swaths of a graph. In doing so, Grover et al. identify that node2vec graph embeddings can encode both homophilic and structural latent features. Tsitsulin et al. generalize the formalism across a range of random-walk based graph embedding techniques, noting that community-based, role-based, and structural features of nodes can all be encoded in a single unified framework [41].

Bipartite Embeddings. Sybrandt et al. [21] explore a number of bipartite graph embedding techniques when presenting First- and Higher-Order Bipartite Embedding (FOBE

and HOBE), including BiNE [42], and Metapath2Vec++ [23]. Due to the star expansion, bipartite embedding techniques are likely among the best suited for hypergraphs. Therefore, we select a subset of bipartite embedding methods that performed the best in this prior work to explore here.

FOBE and HOBE. Because most readers will not be familiar with FOBE and HOBE, and because we find these techniques are often the most useful for embedding-based coarsening, we summarize these techniques here. Both methods create a set of observations, which are used to train embeddings.

Formally, if $G = (V, E)$ is a bipartite graph, $\Gamma(x)$ is the neighborhood of node x, and $u, v \in V$ are nodes, then FOBE assigns a sampled score for the u, v pair as follows:

$$\mathbb{S}^{(\text{FOBE})}(u, v) = \begin{cases} 1 & \Gamma(u) \cap \Gamma(v) \neq \emptyset \\ 1 & uv \in E \\ 0 & \text{otherwise} \end{cases}$$

These observations are fit by minimizing the following for each observed u, v pair, where $\epsilon(x)$ indicates the learned embedding of node x:

$$\mathcal{L}^{(\text{FOBE})}(u, v) = \sigma(\epsilon(u), \epsilon(v)) \log \left(\frac{\mathbb{S}^{(\text{FOBE})}(u, v)}{\sigma(\epsilon(u), \epsilon(v))} \right),$$
$$\text{where } \sigma(x) = \frac{1}{1 + e^{-x}}.$$

HOBE, in contrast, learns higher-ordered relationships that are weighted using algebraic distance, the same underlying technique used within relaxation-based coarsening [13], which places nodes on the unit interval such that similar nodes receive similar coordinates. Formally, the algebraic coordinate of node u is determined by this iterative process:

$$\mathbf{a}_{i+1}(u) = \frac{1}{2} \left(\mathbf{a}_i(u) + \frac{\sum_{v \in \Gamma(u)} \mathbf{a}_i(v) |\Gamma(v)|^{-1}}{\sum_{v \in \Gamma(u)} |\Gamma(v)|^{-1}} \right)$$

where $\mathbf{a}_0$ is randomly initialized, and the algebraic coordinate for u is determined after a fixed number of steps t. HOBE runs $R = 10$ random restarts and summarizes the algebraid similarity between two nodes u and v in the following way:

$$s(u, v) = \frac{\sqrt{R} - d(u, v)}{\sqrt{R}}$$
$$\text{where } d(u, v) = \sqrt{\sum_{r=1}^{R} \left(\mathbf{a}_t^{(r)}(u) - \mathbf{a}_t^{(r)}(v) \right)^2}$$

HOVE uses algebraic similarities to assign weights to node pairs by identifying highly similar shared neighbors: following manner:

$$\mathbb{S}^{(\text{HOBE})}(u, v) = \begin{cases} \alpha(u, v) & \Gamma(u) \cap \Gamma(v) \neq \emptyset \\ \max \left(\begin{array}{c} \max_{x \in \Gamma(v)} \alpha(u, x), \\ \max_{x \in \Gamma(u)} \alpha(x, v) \end{array} \right) & uv \in E \\ 0 & \text{otherwise} \end{cases}$$
$$\text{where } \alpha(u, v) = \max_{x \in \Gamma(u) \cap \Gamma(v)} \min \left(s(u, x), s(v, x) \right)$$

Embeddings are fit by minimizing the following loss for each u, v pair:

$$\mathcal{L}^{(\text{HOBE})}(u, v) = \left(\mathbb{S}^{(\text{HOBE})}(u, v) - \max(0, \epsilon(u)^\intercal \epsilon(v)) \right)^2$$

Combination Embeddings. To demonstrate the ability for embedding-based coarsening to apply to any given embeddings, we explore the combination approach also presented by Sybrandt et al. in [21]. This method learns a joint representation for each node given multiple pre-trained embeddings. This technique does not rely on any random walk strategy, and instead learns a unified embedding per-node given the edge list as a set of embeddings per node. The particular model combines a link-prediction objective with an auto-encoding objective, and in doing so ensures that the resulting joint embedding captures relevant structural signals that are needed to reproduce both the edge list as well as the input embeddings. This technique is very similar to that presented by Wang et al. [43] in that it consists of two connected auto encoders. The result of this method is an embedding that merges the structural features present in a range of embeddings while preserving any useful distinct features from across the set. We direct the reader to [21] to find the specifics of this approach.

Deep Learning Graph Embedding. In addition to the above techniques, which are generally fast, scalable, and parallel sizable, there are another set of deep-learning embedding techniques that apply larger models to the problem of graph embedding. One popular technique, the graph convolutional network [44], constructs a neural network in the same structure as the input graph, and embeddings are derived by a "message-passing" function that distributes node features among neighborhoods. Another technique by Cao et al. learns deep representation by first constructing a large co-occurrence matrix from a process of "random-surfing" following by deep auto encoders [45]. A similar auto-encoder-based approach is presented by Wang et al. [43], wherein a pair of deep auto-encoders both encode nodes independently, as well as ensure that similar nodes are assigned similar embedding. While these deep-learning techniques do achieve high quality results for relatively small graphs, these techniques are less scalable than the previously discussed class of algorithms, due to their larger model structure and the accompanying need for more graph samples. While these techniques could certainly improve the quality of embedding-based coarsening for some hypergraphs, we designed our proposed technique to be independent of any particular embedding, and evaluated our technique over a large collection of hypergraphs and scenarios. As a result, the analysis of deep-learning graph embedding techniques was infeasible for this work.

4 Embedding-Based Coarsening

Embedding-based coarsening begins with a user-supplied hypergraph as well as an embedding of each node. For instance, we use the star-expansion [20] of the hypergraph in order to apply a range of embedding techniques designed for classical graphs. During coarsening, nodes are visited in an order determined by the embeddings of each node's neighborhood. When visited, an unmatched node

is paired with whichever neighbor maximizes a combined measure of edge-wise inner product as well as embedding dot product. After identifying matches, paired nodes are contracted into new coarse nodes, which are assigned an embedding equal the average of all the initial embeddings it represents. Because embedding-based coarsening preserves more global structural features than other methods, the initial partitioning solution is more applicable to the large-scale graph, resulting in higher partitioning quality. We implement embedding-based coarsening in both Zoltan [15] and KaHyPar [18], and explore a range of embedding techniques, including node2vec [22], MetaPath2Vec++ [23], FOBE and HOBE [21], as well as merged embeddings from among this set.

Node Visit Order. We begin matching nodes in an order that tries to prioritize self-similar regions of the input hypergraph. Specifically, a node is a good candidate for being contracted at the current level if it shares a hyperedge with a partner that has a very similar embedding. This indicates that both nodes share many global structural features that would be preserved in their coarsened replacement. However, it is also important to reduce the weight of the resulting coarse nodes. While we also apply more explicit weight-based limitations below, maintaining the balance of coarse node weights begins with adding a weight normalization to this embedding-based similarity score. Otherwise, very dense regions of the network will be contracted into extremely imbalanced and heavy nodes before the rest of the hypergraph, which can eventually invalidate the imbalance constraint. Note that Section 2 contains more thorough definitions for the embedding function ϵ and node weight w, as well as the rest of the notation used in this section. Using these concepts, we can order nodes based on how similar each is to its closest neighbor. Specifically, we order each node u with respect to the following:

$$S_O(u) = \max_{v \in \Gamma(u), u \neq v} \frac{\epsilon(u)^\intercal \epsilon(v)}{w_u w_v} \tag{4}$$

Scoring Contraction Partners. When visiting node u at a given level of coarsening, we must select a neighbor v with which it will contract into a new coarse node in the following level. To do so, we assign a score to each neighbor of u, and select the node with the highest score to match with. We assign scores based on a combination of the KaHyPar "heavy edge" scoring function [19], as summarized in Section 3, as well as the dot product of embeddings. The heavy edge scoring function increases the score of hyperedges with fewer nodes. In real-world applications, this can correspond to "niche" communities that tend to carry more meaning for those involved. We additionally penalize this score by the node's weights in order to reduce the imbalance of the resulting coarse nodes. Specifically, we assign a score to neighboring nodes u and v during the matching process equal to:

$$S_\epsilon(u, v) = \left(\frac{\epsilon(u)^\intercal \epsilon(v)}{w_u w_v} \right) \left(\sum_{e \in \Gamma(u) \cap \Gamma(v)} \frac{w_e}{|e| - 1} \right) \tag{5}$$

Note that in order for a node pair to receive a high S_ϵ score, they must both share low-participation hyperedges as well as global structural embedding-based features. This way, embedding-based coarsening allows us to break ties between multiple nodes that all co-occur in similar intersections of similarly weighted hyperedges, which has the effect of breaking ties, as depicted in Figure 2. Additionally, this measure provides a sorting criteria who's relative values is more important than its absolute value. For this reason we observe a significant benefit by *not* normalizing the dot product value. While some embedding techniques encode node similarity through cosine similarity, which normalizes the dot products between nodes, others do not. In these cases, the relative magnitudes of embedding dot products is a valuable signal for determining coarsening partners.

Imbalance Constraint. As previously stated, it is also important to ensure that the weight of coarsened nodes remains reasonably balanced so that no coarse node becomes so "heavy" that the overall partitioning becomes imbalanced. To address this, we only match nodes that will produce coarse nodes below a given weight tolerance. We accept the weight tolerance $w^{(T)}$ to be a hyperparameter determined by the partitioner. Then, when matching nodes u and v, we disqualify any pair such that $w_u + w_v > w^{(T)}$.

Embeddings. Embedding-based coarsening accepts any pretrained embedding (ϵ) to score potential coarsening partners. We assume that this function places each node of the initial hypergraph into a fixed-dimensional space. Because graph embedding is computationally expensive, we interpolate coarse node embeddings from the initial set. This interpolation consists of the average of all initial node embeddings present in the coarsened node. For instance, if the coarse node u has weight w_u, then that number of nodes from the input hypergraph have been accumulated into u. These initial nodes, $v_1, \ldots, v_{w_u}$ each have embeddings that were supplied in the initial hypergraph embedding. Therefore, we define $\epsilon(u)$ to be the following in the case where u is a coarse node that *does not appear* in the initial embedding:

$$\epsilon(u) = \frac{1}{w_u} \sum_{i=0}^{w_u} \epsilon(v_i) \tag{6}$$

Runtime Impacts. Embedding-based coarsening comes with two runtime increases that are not present in the fast edge-wise coarsening that is typically used by KaHyPar and Zoltan. Firstly, one must perform a graph embedding to learn ϵ. Secondly, at each level of coarsening, we sort V in accordance to embedding-based signals. Graph embedding, in general, is an expensive machine learning operation, requiring significant time and memory to sample a graph and learn embeddings for each node. However, because the proposed embedding-based coarsening algorithm is independent of any particular embedding technique, the specific resources and time needed to produce a graph embedding are subject to change. There are a few broad patterns that most embedding methods follow. Graph embeddings are learned from a set of samples. These samples can be the edges of the graph itself [46], observations determined based on first- or second-order relationships [21], [47], or random-walks of the graph [22], [23], [39]. In each case, the observation capturing process is linear with respect to the size of the graph. Additionally, these observations can often be collected in parallel. Next, the observations are formulated into batches for a neural network to learn

embeddings. Each observation is viewed once-per-epoch, and effects the learned weights of a gain model. Therefore, the complexity of training is equal to the size of the graph times the complexity of performing back-propagation of a particular model. Embeddings can also be paralleled, both by GPU acceleration, as well as through multi-node computation [48]. While the embedding process overall is certainly expensive, the coarsening algorithm proposed in this work only requires one embedding of the input graph as a prepossessing step. This may not be feasible for applications that must partition thousands of midsize hypergraphs daily, but is likely worth it for any application the relies more on the quality of the resulting partition.

The second difference in runtime comes from the sorting used to prioritize coarsening partners during each step of the proposed algorithm. At each iteration, we visit each node to find its most-similar neighbor in terms of node embedding, and then order nodes by this measure. In contrast, classical coarsening randomly orders nodes before identifying partners. While this process introduces the overhead of sorting, we find that removing randomness to prioritize self-similar hypergraph regions can significantly improve partitioning quality while decreasing the quality variance. This may save time overall, as practitioners could reduce the number of random restarts needed to find a high-quality partition. The entire embedding-based coarsening algorithm is outlined in Procedure 1.

Procedure 1 Embedding-based Coarsening.

Output: Produces a set of (u, v) pairs to be contracted in the next level of coarsening.

$M_u \leftarrow \emptyset \; \forall u \in V$ {M is the matching array.}
Sort $u \in V$ in decreasing order by $S_O(u)$. {Eq. (4)}
for $u \in V$ **do**
  if $M_u = \emptyset$ **then**
    $p \leftarrow \emptyset$ {p will be matched with u.}
    $s \leftarrow -\infty$ {s is the score associated with p.}
    for $v \in \Gamma(u)$ **do**
      if $v \neq u$ **and** $M_v = \emptyset$ **and** $w_u + w_v < w^{(T)}$ **then**
        $t \leftarrow S_\epsilon(u, v)$ {Eq. (5)}
        if $t > s$ **then**
          $s \leftarrow t, p \leftarrow v$
    if $p \neq \emptyset$ **then**
      $M_p \leftarrow u, M_u \leftarrow p$ {Match u and p.}
Contract nodes according to M.

5 Considered Implementations

We implement our algorithm in both KaHyPar [18], the n-level partitioner, as well as Zoltan [15], the $\log n$-level partitioner. These two partitioners are considered as other alternatives such as PaToH [25] and hMetis [24] do not provide open source implementations. In each, embedding-based coarsening consists of only a few hundred lines of code, demonstrating that both partitioners are easily expandable for new coarsening algorithms. For ease of development, we use singletons to manage the state of the embedding, and overwrite functions related to scoring neighboring nodes during the coarsening process. We additionally implement a partitioner independent prepossessing step to convert a hypergraph into a star-expanded classical graph in order to apply existing graph embedding techniques. The output of these graph embeddings is supplied as an auxiliary input to the singleton embedding manager. Overall, this separation of embedding and partitioning allows easy experimentation and adaptation with respect to new and constantly changing embedding techniques.

An additional implementation note is necessary for managing embedding behavior during the recursive bisection process of Zoltan, which identifies larger numbers of partitions by iteratively solving the 2-partition problem on iterative halves of the input hypergraph. Simply put, node indices during the recursive bisection process are remapped to start at zero for each recursive partitioning call, which requires the embedding singleton to access the logic of this routine. KaHyPar, in contrast, solves the k-partitioning problem directly at the point of initial solution, and does not perform recursive bisection, and therefore does not require extra engineering. While important for any wishing to implement embedding-based coarsening in the context of recursive bisection, this technical detail does not modify the overall behavior of the proposed algorithm.

6 Experimental Design

We implement embedding-based coarsening in both KaHyPar [18] and Zoltan [15], and compare the result quality against KaHyPar with community-based coarsening [19], KaHyPar with community-based coarsening and flow-based refinement [26], Zoltan with standard coarsening [15], PaToH [25], and hMetis [24]. For both KaHyPar and Zoltan with embedding-based coarsening, we compare embeddings produced by Node2Vec [22], Metapath2Vec++ [23], FOBE and HOBE [21], as well as a combined FOBE+HOBE embedding, and a combined Node2Vec, Metapath2Vec++, FOBE, and HOBE embedding. The combinations are trained using the semi-supervised joint embedding technique also presented in [21], which merges retrained embeddings through a combination of auto-encoding and link-predictive objectives. For the sake of comparison, we choose 100-dimensional embeddings for all cases and for all hypergraphs. We selected the FOBE and HOBE combination as this produces a high quality embedding in prior work [21]. We then wanted to explore a new combination with the whole range of considered embeddings. Additionally, when comparing performance of embedding-based coarsening within KaHyPar, we compare both with and without flow-based refinement. Overall, we explore 18 different partitioning settings with embedding-based coarsening, and five different partitioners with traditional coarsening strategies.

For each of the 23 total partitioner configurations, we explore 96 total hypergraphs. Eighty-six of these are supplied by the SuiteSparse Matrix Collection [27]. These matrices span a range of domains including social networks, power grids, and linear systems. We interpret each matrix $\mathcal{H}$ 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 $\mathcal{H}_{ij}$ 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 to

generate potentially hard instances from graphs [49]. 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 [50] and logistics multi-stage system networks [51]. 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. Summary statistics for each graph are supplied in the appendix.

For each partitioner and hypergraph combination, we explore both the "cut" and the "connectivity" objective, which can influence the initial solution, as well as some decisions during refinement across the considered benchmark[3]. Additionally, we explore a number of partitions (k) for powers of 2 from 2 to 128. For each partitioner, objective, and k-value combination, we run at least twenty trials with different random seeds in order to explore the stability of each scenario. Overall, we compute over 500,000 different experimental trials across our wide benchmark, and for each trial we record all relevant hyperparameters and quality results in a database download supplied in our online appendix.

Metrics. In order to understand aggregate system performance, we report a range of summary statistics for each proposed method. We are primarily concerned with partitioning performance, as quantified by the value of the considered objective value at the end of the multilevel paradigm. However, different hypergraphs have substantially different optimal objective values. Therefore, we report *improvement* statistics between two considered partitioners, with one acting as a baseline for the consideration of the other. A value greater than 1 indicates a *reduction* in the considered partitioner when compared to the baseline across the same hypergraphs.

Formally, if P is a partitioner configuration, including algorithm, embedding method (if applicable), k, and objective function, and H is a hypergraph then let $P(H)$ be the resulting value of the objective function given H and a new random seed. Then, let G be a summary statistic, such as mean, min, max, or standard deviation. We apply G over τ trials of a given partitioner with the same input and different random seeds. The improvement of P with respect to baseline method P_B for a single hypergraph is determined to be:

$$I(P, P_B, G, H) = \frac{G(P_B(H)_1, \ldots, P_B(H)_\tau)}{G(P(H)_1, \ldots, P(H)_\tau)} \tag{7}$$

Note that the formulation above places the baseline partitioner in the numerator because an "improvement" is quantified as a *decrease* in objective value. Therefore, if the proposed partitioner P produces consistently *lower* objective values than P_B, then I will be a number greater than 1.

3. hMetis cannot optimize the "connectivity" objective, and is therefore omitted from that portion of the analysis.

When comparing two partitioners across the entire benchmark of hypergraphs D, we compute the macro-summary. This means that we first apply the summary statistic G to each hypergraph's trials separately, before averaging the results together. Formally, the macro-summary is defined as:

$$\mathcal{I}(P, P_B, G) = \frac{1}{|D|} \sum_{H \in D} I(P, P_B, G, H) \tag{8}$$

When making these comparisons, we select P and P_B pairs such that both partitioners are optimizing the same number of partitions using the same objective. Additionally, we explore summary functions G including mean, min, max, and standard deviation. While mean indicates average quality, min and max indicate worse- and best-case performance, while the standard deviation explores the variance in the resulting partitioners with respect to the random seed.

7 Results

We present a range of result summaries following the experimental design discussed above. For a more in-depth look at our results, we present additional data regarding each hypergraph, and each trial in our online appendix.

To begin our analysis, we summarize the performance improvement of each embedding-based coarsening implementation compared to its respective baseline. For instance, we compare KaHyPar with embedding-based coarsening, using FOBE embeddings, against KaHyPar using community-aware coarsening. We also compare KaHyPar with flow-based refinement, as well as Zoltan with and without embedding-based coarsening. These results are summarized using the macro-improvement statistic $\mathcal{I}$ (Eq. 8), and with the trial summary statistic $G = mean$. Improvements as a percentage for both "cut" and "connectivity" objectives for all considered numbers of partitions (k) in Table 1.

The most striking result in this small collection of summaries is the inverse relationship between improvement and k. As the number of partitions increases, the advantage of embedding-based coarsening decreases. This is due to the manner that we create interpolated embeddings for coarse nodes. As detailed in Section 4, when a newly coarsened node is introduced at a new level of the coarsening process, it is assigned an embedding equal to the average of the initial embeddings it contains. This has the effect of "smoothing" the embedding space at the coarse level. As a result of this smoothing, only major variances between nodes will be captured at the point of initial solution. For instance, if a hypergraph structure has a set number of key clusters, it is hard for embedding-based coarsening to identify anything else at the coarsest level. Higher values of k, larger than the number of identified clusters, therefore do not benefit from this technique.

Future work looking creating more useful coarse node representations is likely to address this problem. However, simple solutions such as embedding coarse graph instances has significant challenges. For instance, we find that small problem instances result in poor embedding convergence across all considered embedding techniques. Therefore we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3017120, IEEE Transactions on Knowledge and Data Engineering

# 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) Average connectivity 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) Average cut improvement.

TABLE 1: Improvement for each implementation of embedding-based coarsening when compared to its corresponding baseline for both the cut and connectivity objectives. Results each use the FOBE embedding instance of embedding-based coarsening. Performance numbers correspond to $\mathcal{I}$ macro-summaries (Eq. 8) where $G = \text{mean}$.

observed in initial trials, re-embedding coarse graphs dramatically *decreased* result quality. Additionally, graph embeddings are computationally expensive, and performing any non-constant number of embeddings is likely to be infeasible for any real-world problem instance.

We continue our comparison of embedding-based coarsening across a range of baselines in Figure 3a. Here we compare Zoltan with embedding-based coarsening, KaHyPar with embedding-based coarsening and flow-based refinement (the better performing KaHyPar implementation), against all baseline methods. We additionally explore a range of summary statistics G including mean, best-case (min), worse-case (max), and standard deviation. To easily compare all partitioners, we use KaHyPar with flow-based refinement as the baseline (P_B) for all methods. Therefore, KaHyPar with flow always scores a one, denoted by the dashed line in each plot, and an improvement over this baseline is indicated by a macro-improvement $\mathcal{I}$ greater than one.

We observe a similar negative relationship between k and improvement across the benchmark in Figure 3a as was seen in Table 1. When considering the connectivity objective for k values of 2 and 4, we interestingly observe that both the KaHyPar and Zoltan implementations with embedding-based coarsening outperform the baseline. This is especially important for Zoltan, which greatly under performs the baseline without our proposed coarsening. When looking at best-base performance ($G = \max$) we observe that the negative trends with respect with k is less pronounced for KaHyPar and the connectivity objective. This trend demonstrates the consistent ability of embedding-based coarsening to identify solutions that are of a higher quality than any found by any considered baseline, and suggests that practitioners willing to accept a quality-for-speed trade-off can find substantial performance gains with our proposed technique.

Examining the standard deviation results shown in Figure 3a, we observe that embedding-based coarsening greatly improves the standard deviation of possible results for a given hypergraph (shown where $G = \text{std}$). This decrease in variance comes from the deterministic node-visit order, which replaces a typically random ordering. As a result, the standard deviation of KaHyPar with embedding-based coarsening can be reduced by over an order-of-magnitude in some cases. Because many applications run multiple partitioning trials with various random seeds in order to find a top-performing result [52], we find that this decrease in variance enables these applications to run fewer trials while retaining the same confidence in their performance.

Comparison with Community-aware Coarsening. Embedding-based coarsening attempts to merge together self-similar regions of the input hypergraph with respect to the structural signals provided by node embeddings. In contrast, community-aware coarsening restricts the contraction of nodes that do not share a cluster assignment in the original hypergraph. While these two approaches are very similar, they both promote contractions within self-similar regions of the original hypergraph, we find that embedding-based coarsening is a more flexible constraint. Embedding-based coarsening simply penalizes nodes that do not share structural features, but may still merge seemingly dissimilar neighbors if no better options are found. Because of this relaxation, we find that embedding-based coarsening outperforms community-aware coarsening in a range of scenarios. This behavior, which we first report in aggregate in Table 1, is depicted in depth in the appendix. In this example, each considered hypergraph is listed, and graph-wise performance summaries I (Eq. 7) are depicted for each. The specific properties of each graph are briefly summarized in the appendix, with more information available online.

In the summary table, we demonstrate that for low k-values, that embedding-based coarsening can improve result quality over community-aware coarsening by around 10% for $k = 2, 4, 8$. When viewing the per-hypergraph results, we see a more detailed picture. Some hypergraphs with particularly useful structural features, such as then hypergraph constructed from the enron email dataset, the eu email dataset, or the difficult and noisy merged hypergraphs, *can find partitioning solutions with a connectivity objective that is between one half and one fourth of the community-aware baseline.* For many other graphs this improvement is a modest few percentage points, while other graphs are relatively unchanged. For these graphs, we find that the community-detection solution found by KaHyPar provides nearly the same information as the selected graph embedding, leading to no improvement. Only a small handful of graphs are substantially worsened by this proposed technique when compared to the community-aware baseline. For instance, 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%.

Comparison Across All Partitioners. We supply a large

table in the appendix that depicts the average improvement of each proposed embedding-based coarsening partitioner configuration against each baseline for the connectivity objective. The numbers in each cell correspond to the macro-summary $\mathcal{I}$ using $G = \text{mean}$ to summarize trials. For space limitations, we only show this one large table, but online we present similar tables for the cut objective, as well for the min, max, and standard-deviation summarizes. In both the in-print and online appendix we supply improvement numbers per-point of comparison per-hypergraph. When examining the included table, however, we see clear trends that are replicated in each online table. All considered embeddings improve performance similarly, with FOBE and HOBE performing marginally above the other methods in some cases. We observe that Zoltan is the "easiest" baseline partitioner, while KaHyPar with flow-based refinement is the most challenging. Because KaHyPar with flow-based refinement produces higher quality partitions than Zoltan in prior work [26], it is notable that some instances of Zoltan with embedding-based coarsening can achieve similar quality.

When looking across the KaHyPar trials, we see that embedding-based coarsening without flow-based refinement can outperform community-aware coarsening with the most expensive flow-based refinement. This result confirms the intuition, initially discussed in Section 1, that the coarsening process one of the most fundamental operations in multilevel partitioning.

Across all considered implementations of embedding-based coarsening we still observe a decrease in performance for larger values of k. As previously discussed, this derives from the smoothed embedding space produced by iterative averages of coarse nodes. It is worth noting that this smoothing effect produces results that are most similar to KaHyPar with its broad community-aware coarsening. In contrast, embedding-based coarsening still outperforms Zoltan and PatoH, partitioners that do not account for global structural properties in a similar way.

Runtime Our proposed coarsening introduces two sources of overhead into the typical partitioning process, as described in Section 4. We compare this runtime effect across our benchmark by focusing on the considered implementations of KaHyPar. In Figure 3b we present average runtimes for each implementation, as well as isolated runtimes for the coarsening and uncoarsening phases. Note that when reporting runtime results for the embedding-based implementations of KaHyPar, these performance numbers represent an average across all considered embeddings. We find that embedding-based coarsening multiplies the time needed to coarsen an input hypergraph, which is due to the additional node-wise comparisons and sorting overhead. However, we also find that coarsening amounts to only a fraction of the overall petitioner runtime, which is dominated by the runtime of the uncoarsening phase, which is relatively unchanged.

We present a plot of the distributions in runtime for FOBE and HOBE across the benchmark in the supplemental information. These methods are the slower of those considered, and are not as easily distributed as the other random-walk embedding techniques. We again note that all considered embeddings are treated as an offline component that is subject to its own field of research, are highly subject to hyperparameters like batch size and number of training epochs, and that the considered graph embedding implementations have not been optimized. However, we find that the median considered FOBE embedding requires 1832 seconds, while the median HOBE embedding requires 8682 seconds.

These runtimes are multiples of the average partitioning time, which may disqualify our proposed algorithm from "high performance" scenarios, such as hypergraph partitioning to accelerate scientific computing workloads at runtime [6]. However, many applications, such as placing circuits on a chip [1], recommending documents [53], machine learning on hypergraphs [2], or deep learning on hypergraphs [54], could all significantly benefit from the slow but higher-quality partitioning brought by embedding-based coarsening.

8 Conclusion

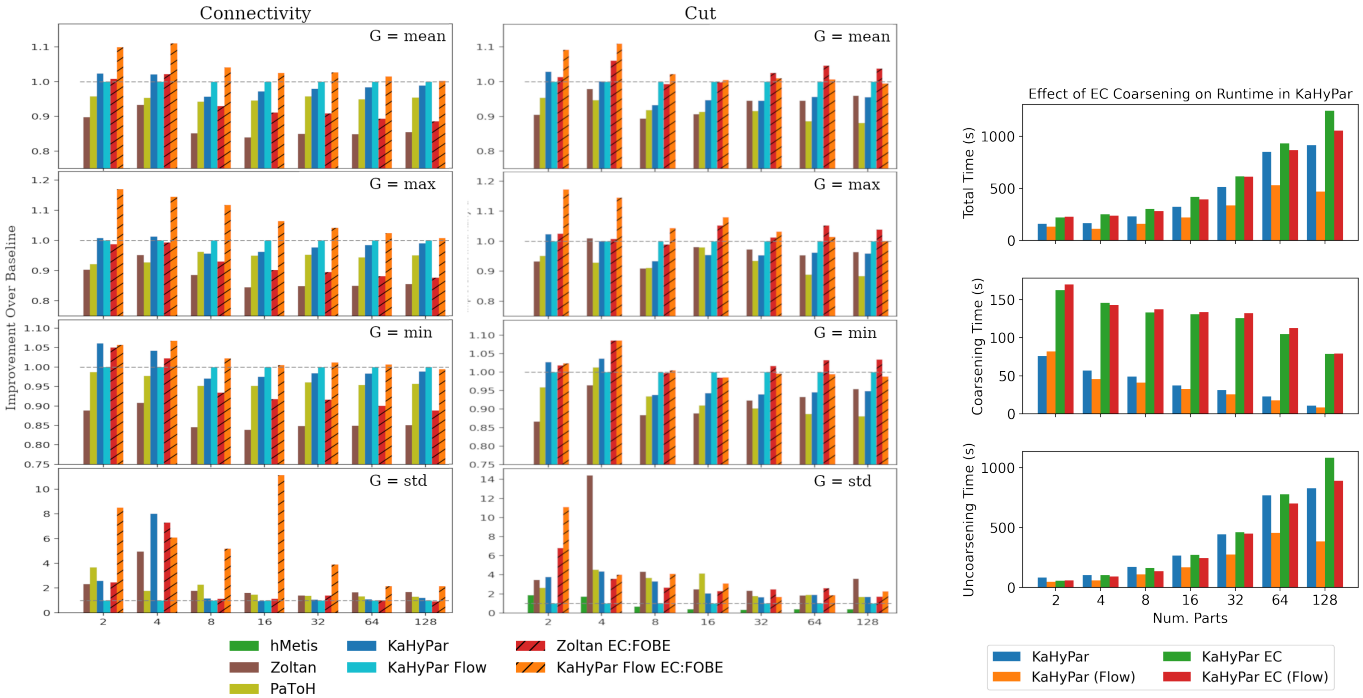
We propose embedding-based coarsening, an approach that leverages global structural features present in a pretrained hypergraph embedding in order improve the solution quality of multilevel hypergraph partitioning. This approach prioritizes self-similar regions of the hypergraph by visiting nodes in a deterministic order based on the embedding properties of each node's neighborhood. From there, embedding-based coarsening matches nodes by a score that combines a more traditional edge-wise inner-product with the dot product of node embeddings. We observe that the introduction of embedding-based features provides a "tie-breaking" mechanism that ultimately preserves global structural features at the coarsest level in the V-cycle. We implement our proposed coarsening strategy in both KaHyPar [18] and Zoltan [15].

We observe a significant increase in quality for small values of k (from 2 to 16) gained from embedding-based coarsening. For higher values of k we observe overall quality that returns to the state-of-the-art baseline. Furthermore, we find that embedding-based coarsening improves partitioning quality significantly across a range of scenarios in both the KaHyPar and Zoltan frameworks. *Specifically, KaHyPar with flow-based refinement [26] and embedding-based coarsening, using either FOBE or HOBE [21] to produce node embedding, scores consistently higher on average than all considered baselines.* Furthermore, we find that by replacing the random node visit order in many coarsening algorithms with a deterministic strategy that prioritizes self-similar node pairs, we both improve solution quality while drastically reducing solution variance, often by an order of magnitude. Large scale results for all benchmarks and considered metrics is also available in the online appendix: sybrandt.com/2019/partition.

9 Acknowledgements

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3017120, IEEE Transactions on Knowledge and Data Engineering



(a) The above depicts the relative performance of various partitioners, each using KaHyPar with flow-based refinement as a baseline. The results correspond to macro-summaries $\mathcal{I}$ (Eq. 8), where a value of 1, indicated by the horizontal dashed line, is baseline performance of P_B. We explore different summary statistics G, including mean, max, min, and standard deviation.

(b) Average Runtime of KaHyPar Across Benchmark.

Fig. 3: Effect of Quality and Runtime

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] B. Hendrickson and A. Pothen, "Combinatorial scientific computing: The enabling power of discrete algorithms in computational science," in *International Conference on High Performance Computing for Computational Science*. Springer, 2006, pp. 260–280.

[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.

[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] 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.

[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] 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.

[15] 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.

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

[17] E. Sadrfaridpour, T. Razzaghi, and I. Safro, "Engineering fast multilevel support vector machines," *Machine Learning*, vol. 108, no. 11, pp. 1879–1917, 2019.

[18] 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.

[19] 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.

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

[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. 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

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

[35] 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.

[36] 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.

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

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

[39] 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.

[40] 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.

[41] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "Verse: Versatile graph embeddings from similarity measures," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 539–548.

[42] M. Gao, L. Chen, X. He, and A. Zhou, "BiNE: Bipartite Network Embedding," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, ser. SIGIR '18. New York, NY, USA: ACM, 2018, pp. 715–724. [Online]. Available: http://doi.acm.org/10.1145/3209978.3209987

[43] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[44] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[45] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[46] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, "PyTorch-BigGraph: A Large-scale Graph Embedding System," in *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019.

[47] 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.

[48] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[49] 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.

[50] 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.

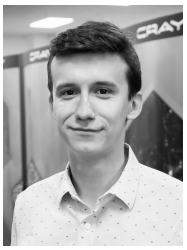
[51] L. Stock, *Strategic Logistics Management*, ser. Cram101 Textbook Outlines. Lightning Source Inc, 2006. [Online]. Available: http://books.google.com/books?id=1LyCAQAACAAJ

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

[53] Y. Zhu, Z. Guan, S. Tan, H. Liu, D. Cai, and X. He, "Heterogeneous hypergraph embedding for document recommendation," *Neurocomputing*, vol. 216, pp. 150–162, 2016.

[54] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3558–3565.

Justin Sybrandt received his B.Sc. in Computer Science from Grove City College and will graduate May 2020 with a Ph.D. in Computer Science from Clemson University. His work is focused in latent representations of text and graphs, with an emphasis on understanding scientific literature. He has since begun work at Google Brain. Contact him at jsybrandt@google.com.



Ruslan Shaydulin received his B.Sc. in Applied Mathematics and Physics from Moscow Institute of Physics and Technology and is currently pursuing a Ph.D. in Computer Science at Clemson University. His work is focused on quantum and classical algorithms for combinatorial optimization and machine learning. Contact him at rshaydu@clemson.edu.

Ilya Safro is an associate professor in the School of Computing and faculty scholar in the School of Health Research, Clemson University. His research interests include graph algorithms, machine learning, combinatorial scientific computing, and quantum-classical methods. Safro received a Ph.D. in applied mathematics and computer science from the Weizmann Institute of Science. Contact him at isafro@clemson.edu.