HGP TIG Challenge Description

The Innovation Game | Aoibheann Murray

December 1, 2025

1 Problem Description and Formulation

The Hypergraph Partitioning challenge at TIG can be formulated as follows:

Goal: Divide a hypergraph into a specified number of balanced partitions while minimizing the number of cut hyperedges.
Consider a hypergraph $H = (V, N)$, with:

- A set of nodes (vertices) V, where each node $v \in V$ has a weight $w[v]$.
- A set of hyperedges (nets) N, where each hyperedge $n \in N$ is a subset of V and has a cost $c[n]$.

Constraints:

- Each node belongs to exactly one part, i.e., $V_k \cap V_l = \emptyset$ for all $1 \leq k < l \leq K$.
- Every part must contain at least one node, i.e., $V_k \neq \emptyset$.
- The total weight of each part, W_k, must not exceed the average part weight W_{avg} by more than an allowed imbalance tolerance:

$$W_k \leq W_{\text{avg}}(1 + \epsilon),$$

where $W_k = \sum_{v \in V_k} w[v]$, $W_{\text{avg}} = \frac{\sum_{k=1}^{K} W_k}{K}$, and ϵ is the allowed imbalance tolerance.

Objective:

$$\text{Minimize:} \quad \sum_{n \in N} c[n]\big(\lambda_n - 1\big),$$

where the connectivity, λ_n, is the number of parts that hyperedge n spans.

2 Baseline Calculation

The baseline algorithm provides a reference performance metric for each instance. It is crucial that this baseline is both stable (e.g., consistently within 20% of the best solution) and efficient.

The algorithm recursively partitions the node set into two subsets until the desired number of partitions is reached. Before applying any greedy refinements, the algorithm begins with an initial bipartition seeded by the two level-1 groups produced by the hypergraph generation method. Utilizing this initial partition capitalizes on the inherent structure of the node set, providing an effective starting point that results in a more stable baseline partition. Each subsequent bipartitioning step proceeds as follows:

1. **Determine target sizes.** Given a current subset of nodes, calculate how many nodes should go to the left and right parts (e.g., if we aim for 2^d total parts, each subdivision targets two parts of prescribed sizes).
2. **Sort nodes by degree.** For the nodes in the current subset, compute their degrees (the number of hyperedges to which each node belongs). Sort them in descending order so that higher-degree nodes are placed first.

3. **Place nodes greedily.** Initialize two arrays (one per part) to track hyperedges already "activated" (those with at least one node assigned) within each part. For each node in sorted order:

   (a) Count how many of its hyperedges are activated in the left part and how many in the right.

   (b) If one side has a strictly higher overlap, assign the node to that side (provided it has not reached its target size). If overlaps are equal, assign to the part with fewer nodes.

   (c) If one part has already reached capacity, assign the node to the other part by default.

   Continue assigning nodes until one part reaches its target size; then assign any remaining nodes to the other part.

4. **Recursive subdivision.** After producing each bipartition, recursively apply the same procedure to each newly formed part until the desired number of parts (e.g., 64) is reached.

Finally, the connectivity metric of the complete multiway partition is computed, giving the `baseline_value`. Although this local greedy strategy does not capture global interactions perfectly, it remains computationally efficient, intuitive, and serves as a solid performance benchmark for more sophisticated methods.

3 Random Instance Generation

3.1 Motivation

To ensure that the challenge drives progress on practically relevant problems, our synthetic instances must resemble the hypergraphs most frequently encountered in the literature and in production workloads. We therefore aim to match key strucutres found in *SuiteSparse Matrix Collection* [1], which is a standard data source for evaluating hypergraph partitioners (e.g., [2]).

3.2 Methodology

We generate each instance with HyperLap [3], a GPU-friendly, hierarchical extension of the HyperCL null model that reproduces the heavy-tailed, community-centric overlap patterns observed in real data.
Given

- an array of *node weights*;
- an array of desired *hyperedge sizes*; and
- a *level-weight* vector $\mathbf{p} = (p_1, \ldots, p_L)$ with $\sum_\ell p_\ell = 1$,

the generator proceeds in two stages.

1. Hierarchical layout. For $|N|$ hyperedges we create $L = \lfloor \log_2 |N| \rfloor$ levels. Level ℓ consists of $2^{\ell-1}$ equally sized, nested groups: if two nodes share a group at level ℓ, they also share a group at every lower level.

2. Edge construction. For every desired hyperedge size s we

1. sample a level ℓ with probability proportional to p_ℓ;
2. choose one of the $2^{\ell-1}$ groups at that level uniformly at random; and
3. repeatedly sample nodes within the chosen group proportional to their node degree until s distinct nodes are selected.

This scheme preserves both the input degree sequence and the hyperedge-size distribution while injecting realistic overlap patterns.

3.3 Interpreting the Level-Weight Vector

The level-weight vector $\mathbf{p}$ dictates the *resolution* at which hyperedges form: low levels span large node sets, high levels cover tiny groups. Shifting probability mass therefore tunes how strongly hyperedges overlap and how "visible" the community structure is:

Low-level weight (p_1+$p_2 \approx 1$). Edges sample from the coarsest groups, so overlaps are largely accidental and community structure is weak—hard instances for partitioners.

High-level weight ($p_L \gg p_1$). Edges stay inside very small groups, yielding tight micro-communities; coarse partitioning becomes easy.

Multi-dominant low levels (p_2+p_3+$p_4 \approx 1$). Weight spread over the first few refinements produces a realistic, multiscale hierarchy with heavy-tailed overlaps.

One-level dominant. Nearly all mass on one level k makes block-diagonal structure at a single, known scale - easy to partition if k is exploited.

Using *HyperLap+* (which automatically fits optimal level-weight vectors to observed hypergraph data), we observed two dominant regimes in SuiteSparse hypergraphs: a single-level dominant pattern and a multi-dominant low-level pattern. We adopt the latter due to its richer overlap structure, posing greater challenges for partitioning algorithms.

Varying the Level-Weight Vector p

Instead of using a fixed multi-peak vector $\mathbf{p}$, we sample it from a narrow distribution derived from multiple HyperLap runs on our reference real-world hypergraph. This increases challenge difficulty and more effectively conceals the underlying hierarchy.

Due to variations from random sampling and normalization effects, the net effect of $\xi = 0.2$ in our chosen implementation results in noise between 17% and 25%.

3.4 Fixed Parameters

- **nodes vs. hyperedges.** We set $|N| = |V|$ to match the sparsity pattern of SuiteSparse matrices.
- **Uniform weights and costs.** All node weights and hyperedge costs are 1, so the objective reduces to minimising the connectivity λ_n.
- **CUDA implementation.** HyperLap maps naturally onto GPUs, which was a major motivation for this hypergraph generator.

3.5 Addition of Noise via a Random Hypergraph

We integrate noise by combining our structured hypergraph with a purely random hypergraph, following the approach of Kaminski et al. [4]. The parameter ξ controls the proportion of each node's degree attributed to random (background) hyperedges.
To smoothly incorporate noise within our generation process, we adjust the level-weight vector (see random instance generation details in our previous update), achieving the same effect as directly splitting each node's degree.
Specifically, let

$$\mathbf{p}_s = (p_1, p_2, \ldots, p_L), \qquad \text{where } \sum_{\ell=1}^{L} p_\ell = 1$$

represent the structured hypergraph's level-weight vector, and let

$$\mathbf{p}_r = (1, 0, \ldots, 0)$$

represent a purely random hypergraph, where hyperedges are uniformly chosen over all nodes.
We define the combined level-weight vector $\mathbf{p}$ as

$$\mathbf{p} = \xi\, \mathbf{p}_r + (1 - \xi)\, \mathbf{p}_s.$$

By sampling a fraction ξ of hyperedges according to $\mathbf{p}_r$ and a fraction $(1-\xi)$ according to $\mathbf{p}_s$, we ensure explicit control over the proportion of random hyperedges, while preserving the normalization condition $\sum_{\ell=1}^{L} p_\ell = 1$.
We propose to initially add 20% noise ($\xi = 0.2$), consistent with Kaminski et al.[4]. This value can be adjusted if necessary.

3.6 Minimum Part Size for k = 64 Parts

To ensure sufficient hypergraph size for a 64-way partition, and following Gottesbüren et al.[5], who partition medium-sized hypergraphs with at least 7,500 nodes into up to 128 parts ($\sim$ 58 nodes per part), we set the minimum difficulty to

$$\texttt{num\_hyperedges} = 4000.$$

Given that the number of nodes is roughly 92% of the number of hyperedges, this corresponds to approximately

$$3{,}680 \text{ nodes} \approx 58 \text{ nodes per part.}$$

Most state-of-the-art partitioners employ a *coarsening–partitioning–uncoarsening* pipeline, typically coarsening down to hypergraphs of approximately 160k nodes before partitioning. As the challenge evolves, we expect to scale our minimum difficulty toward this standard.

4 Tracks

Within each challenge, there are various challenges tracks. These can range over instance size and/or type. Currently, the challenge supports varying sizes of Hyperlap generated instances:

- 10000
- 20000
- 50000
- 100000
- 250000

References

[1] Scott Kolodziej et al. "The SuiteSparse Matrix Collection Website Interface". In: *Journal of Open Source Software* 4 (Mar. 2019), p. 1244. DOI: 10.21105/joss.01244.

[2] Aleksandar Trifunovic and William Knottenbelt. "Parallel multilevel algorithms for hypergraph partitioning." In: *J. Parallel Distrib. Comput.* 68 (Jan. 2008), pp. 563–581.

[3] Geon Lee, Minyoung Choe, and Kijung Shin. *How Do Hyperedges Overlap in Real-World Hypergraphs? – Patterns, Measures, and Generators*. 2021. arXiv: 2101.07480 [cs.SI]. URL: https://arxiv.org/abs/2101.07480.

[4] Bogumił Kamiński, Paweł Prałat, and François Theberge. "Hypergraph Artificial Benchmark for Community Detection (h–ABCD)". In: *Journal of Complex Networks* 11 (2023). DOI: 10.1093/comnet/cnad028.

[5] Lars Gottesbüren et al. "Scalable high-quality hypergraph partitioning". In: *ACM Transactions on Algorithms* 20.1 (2024), pp. 1–54. DOI: 10.1145/3626527.