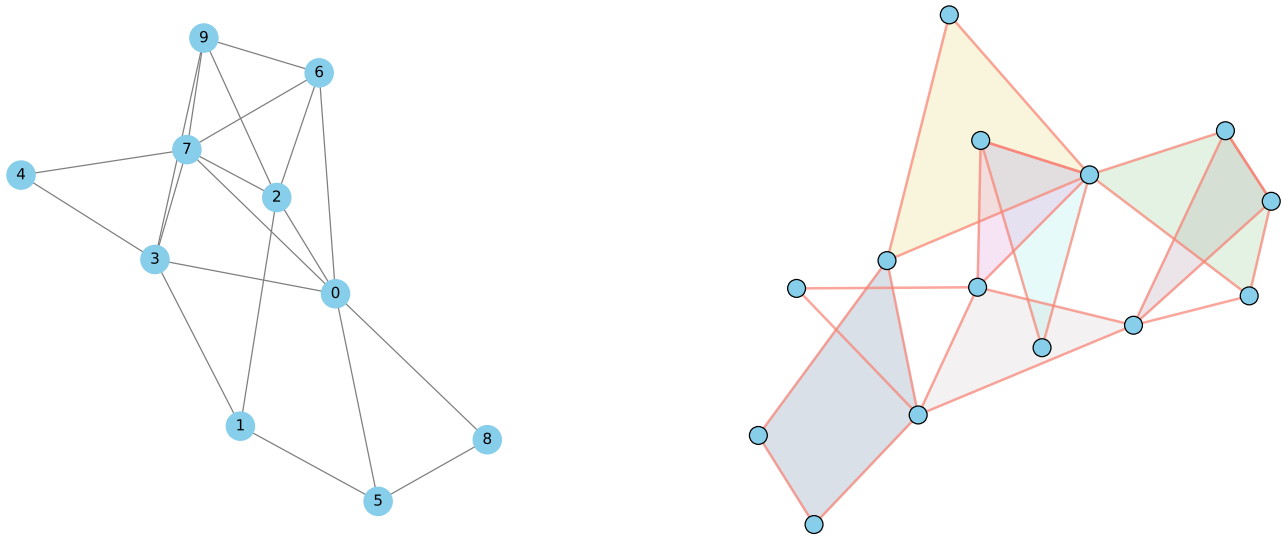
HYGENE: A Diffusion-based Hypergraph Generation Method

September 20, 2024

Outline

Introduction

Introduction



(a) Graph Example

(b) Hypergraph Example

Figure: Examples of a graph and a hypergraph

- Graphs are made of nodes connected by edges
- Hypergraphs allow edges (then called hyperedges) to connect as many nodes as desired

Motivation

- To the best of our knowledge, no deep-learning based method for hypergraph generation exists.
- Compared to graphs, hypergraphs can capture higher-order and more subtle relationships between nodes.
- Broader field of applications ?

Application Example 1: 3D Meshes

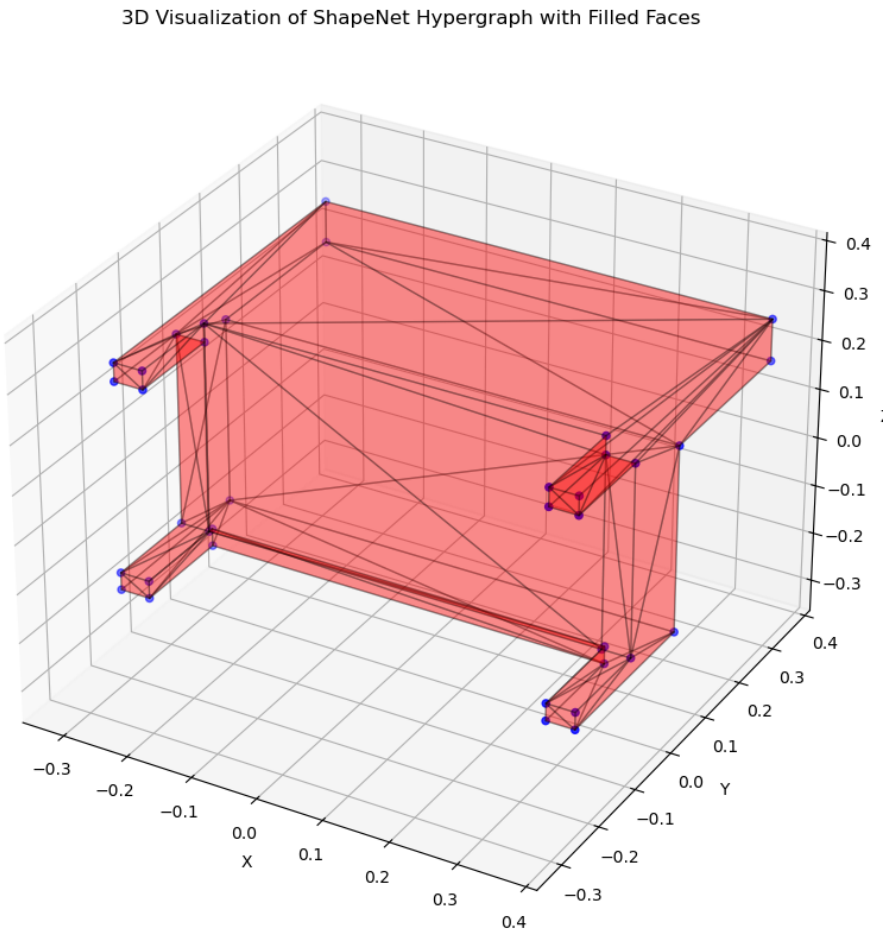


Figure: Hyperedges correspond to triangles in the mesh.

Application Example 2: Electronic Circuits

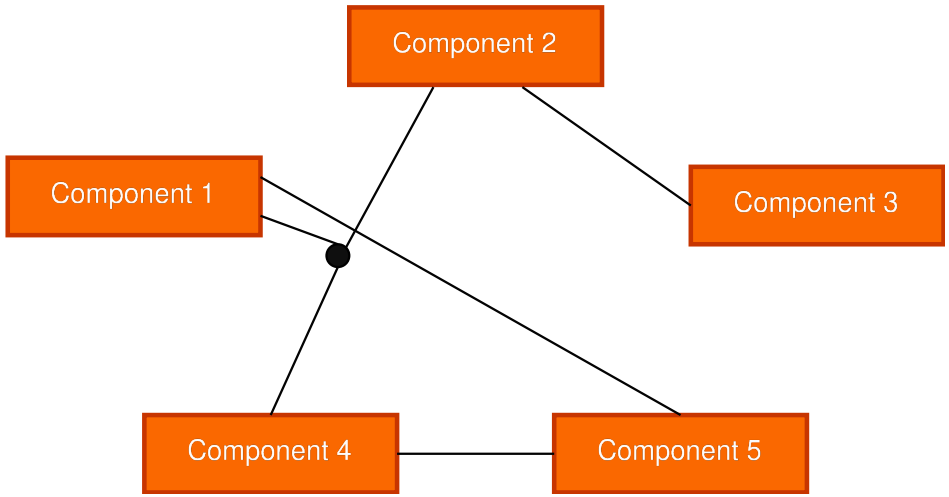


Figure: Hyperedges group components connected to the same router.

Application Example 3: Pharmaceutical Research?

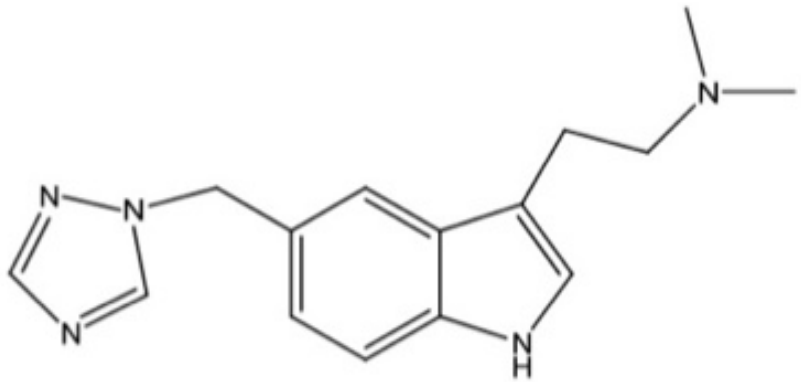


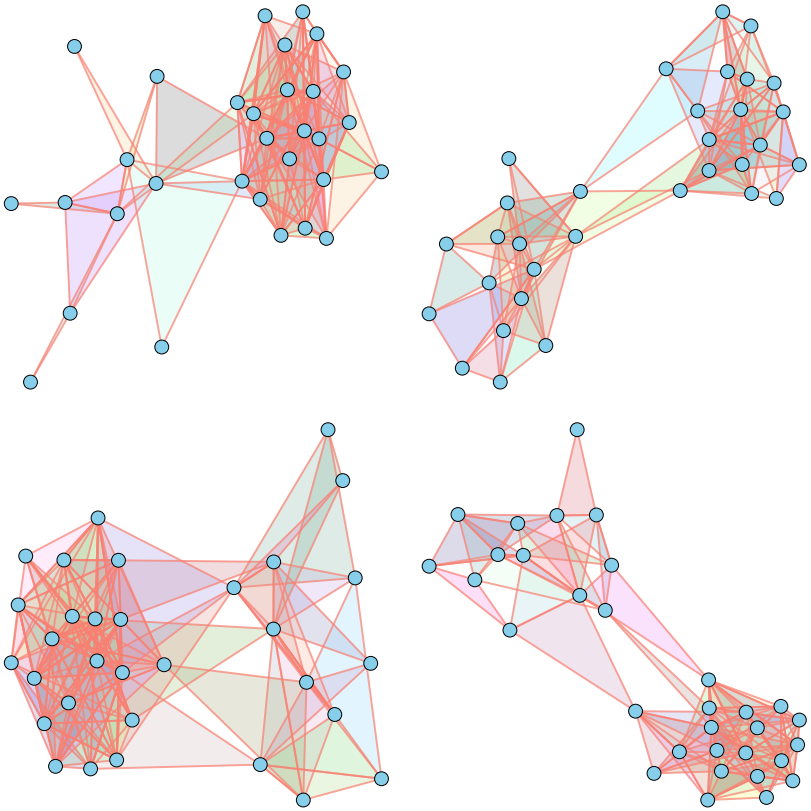
Figure: Hyperedges could group atoms belonging to the same compound (for example the three nitrogen atoms on the left side) in order to highlight that their relationship is not on a two-by-two basis but on a higher-order scale?

The Problem We Studied

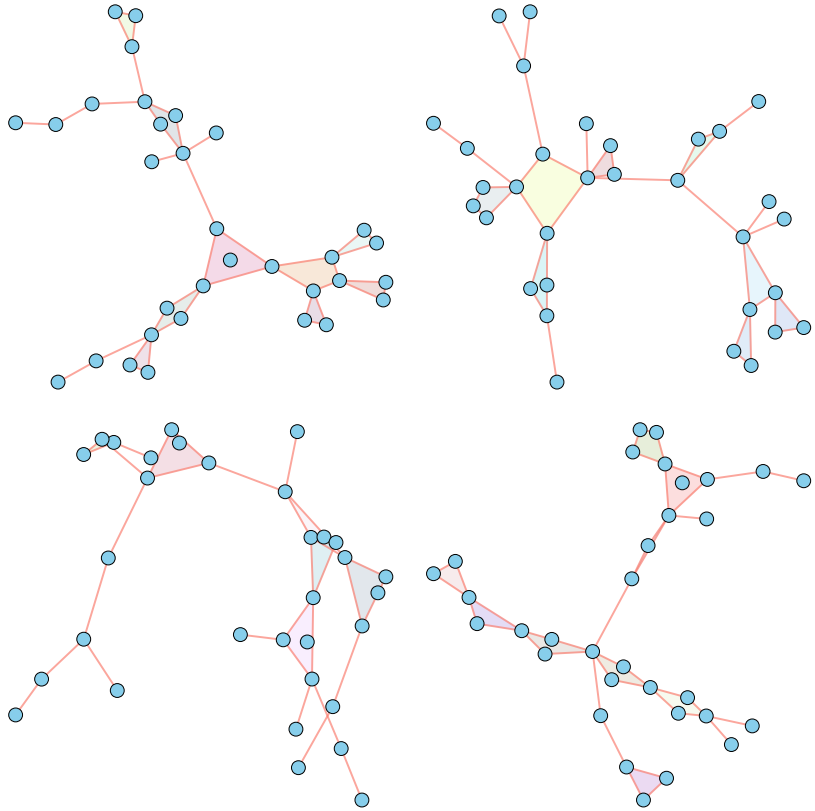
The Problem We Studied

- Given a dataset of hypergraphs belonging to the same "class" (i.e. sharing the same properties), train a model able to generate additional samples.
- First step: only for featureless hypergraphs (no 3D positions, component types, etc...).

Dataset Examples



(a) Stochastic Blocks model



(b) Tree hypergraphs

Short Graph Generation Course

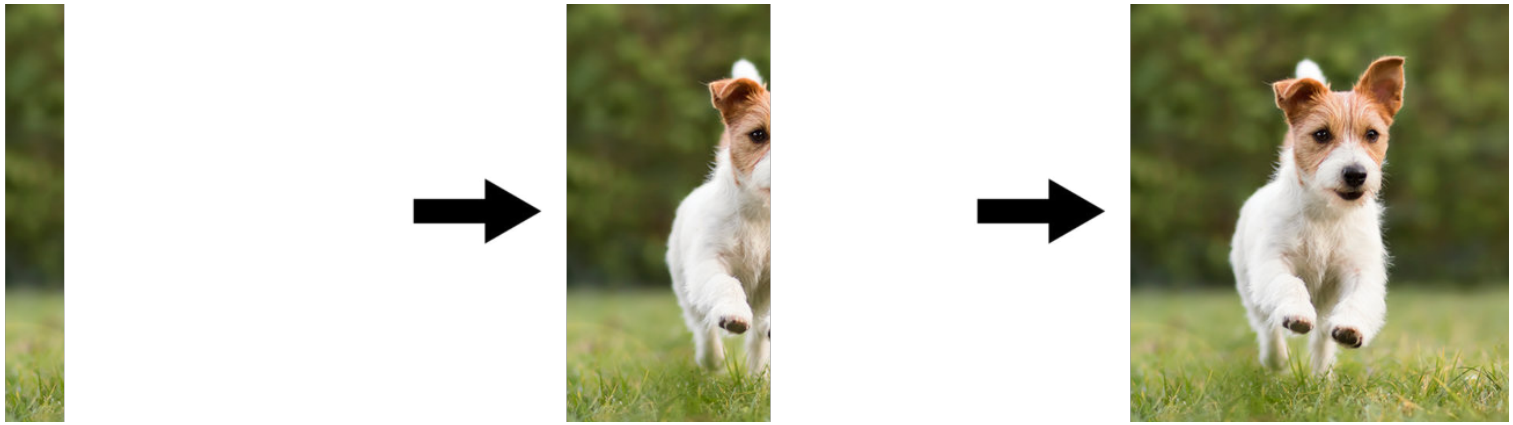
Next: Short Graph Generation Course

- Upcoming: Short course on graph generation
- Note: This focuses on regular graphs, not hypergraphs
- Will serve as a foundation for understanding what we did

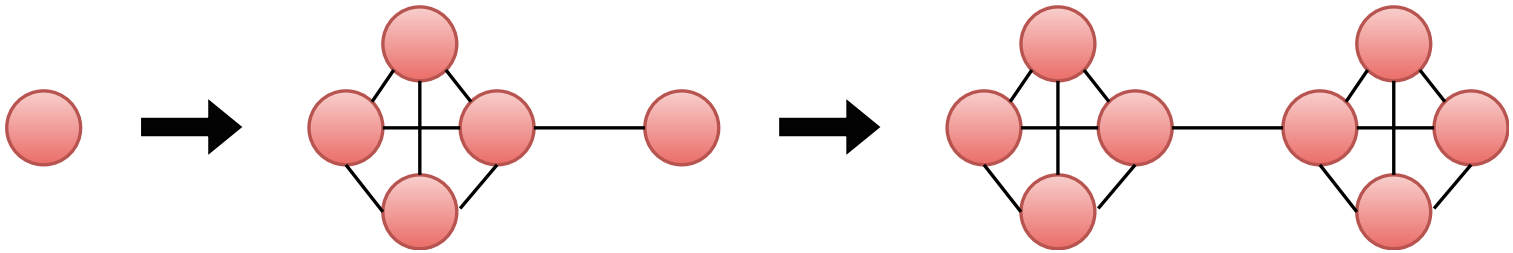
Graph Generation: Two Approaches

- Two classes of graph generators:
  - One-shot: Generate the whole graph at once
    - Faces complexity issues as graph size increases
    - Not suitable for large graphs
  - Auto-regressive: Generate the graph one chunk at a time, using the previous step as an input
    - "Spatial generation": starting from a side of the graph, gradually add the other parts
    - "Multi-resolution generation": starting from a coarse view of the graph, gradually add details

Spatial Generation



(a) Generating the image from left to right (the dog image comes from Adobe Stock)

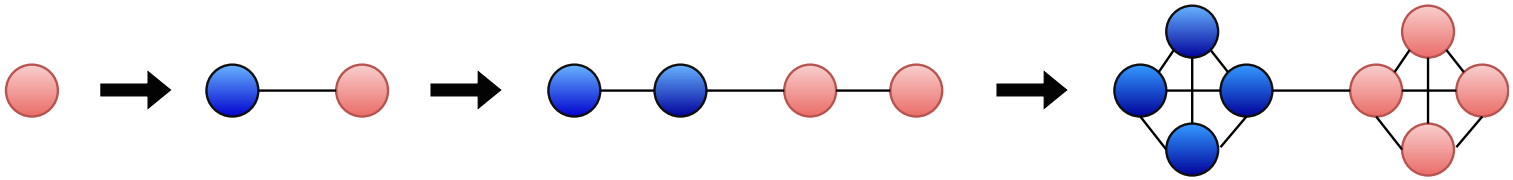


(b) Generating the graph from one side to the other

Figure: Analogy between the generation of an image and the generation of a graph

Multi-Resolution Generation

(a) Generating the image by adding details from a noisy image (the dog image comes from Adobe Stock)



(b) Generating the graph by adding details at multiple resulution scales

Figure: Analogy between the generation of an image and the generation of a graph

Spatial vs. Multi-resolution Generation

- Spatial generation:
  - Possible for image generation (natural left-to-right order)
  - Problematic for graphs (no natural order): no satisfying answer about how to order the nodes, requires an arbitrary decision
  - Possible divergence? Each step has a degree of freedom stacking on top of the previous iterations
- Multi-resolution generation:
  - Avoids ordering problem
  - Starts from coarse view (single node) and iteratively adds details to each region of the graph
  - Likely less divergence (locally contained patches)

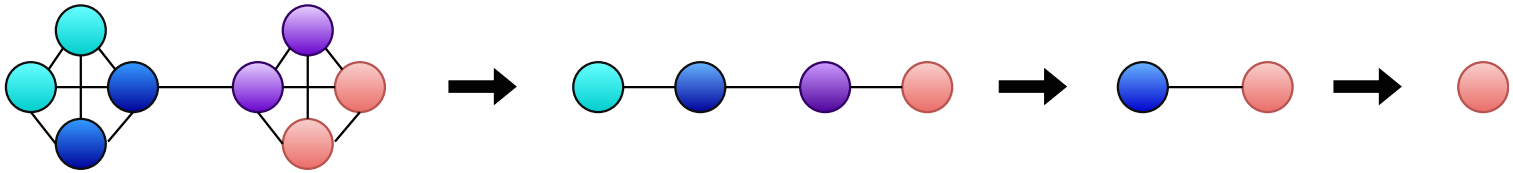
Our Selected Approach: Multi-resolution Generation

- Based on "EFFICIENT AND SCALABLE GRAPH GENERATION THROUGH ITERATIVE LOCAL EXPANSION" by Bergmeister et al.
- Start with a dataset of graphs
- For each graph:
  - Construct a "coarsening" sequence i.e. views of the same graph at different resolutions
  - Such that it preserves the aspect of the graph at multiple resolutions
- Train a model to:
  - Use information from the current level
  - Recover details of the previous finer resolution level
- Generation process:
  - Start from a single node
  - Apply the model iteratively to generate the next resolution
  - Continue until desired size is reached

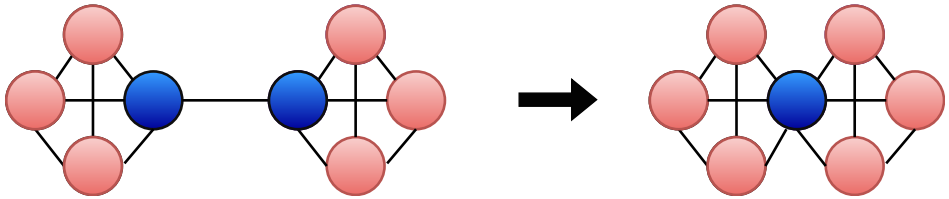
Technical Note: Coarsening Process

- Starting from a graph from the dataset
- We iteratively merge pairs of adjacent nodes until only a single node remains
- Each merging is done such that the spectral properties of the laplacian of the graph are preserved
- This is just a formalization of "keeping the information of the graph" (which is contained in the spectral properties of the laplacian)

Visual Explanation



(a) Ideal coarsening that preserves the aspect



(b) Coarsening losing properties

Figure: The preservation of the spectral properties ensures that the first coarsening is selected over the second one

Learning to invert the coarsening process

- Three-step process for inverting coarsening:
  - Selection step: Select nodes to duplicate (a model learns to identify which nodes were merged at the previous coarsening step)
  - Expansion step: The duplicata are then connected, and keep the original connections of their parent node
  - Refinement step: Select which edges to keep (a model learns to identify which edges were in the original graph before the coarsening step)
- Two models are trained:
  - First model: Learns the selection step
  - Second model: Learns the refinement step
- In practice they are diffusion models

Visual Example

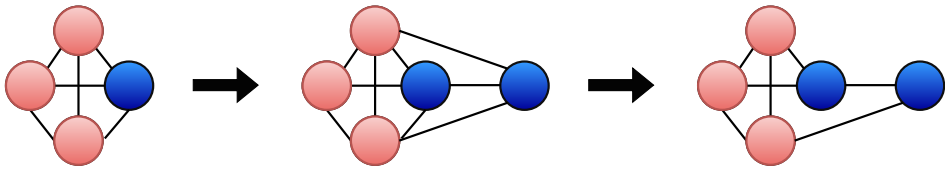


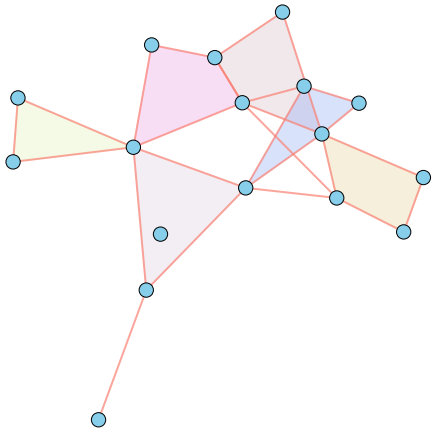
Figure: First the model identify which nodes need to be duplicated (here only the blue one), then the graph is expanded, and a second model decides which edges to keep

Our Work

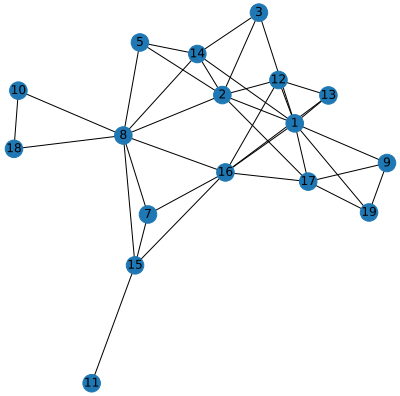
Next: Generalization to Hypergraphs

- Upcoming: Our generalization
- We use the clique expansion to select the coarsening sequences
- We maintain a parallel bipartite view of the hypergraph
- We train a customized version of the previous model to reconstruct the bipartite graph

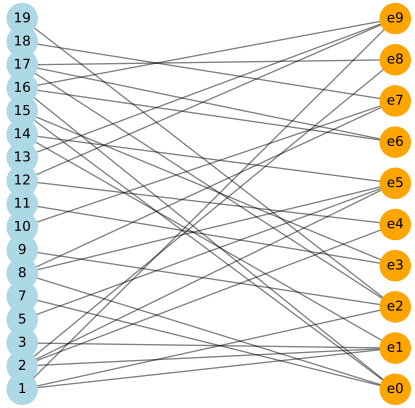
Hypergraph Projections



(a) Original Hypergraph



(b) Clique Expansion

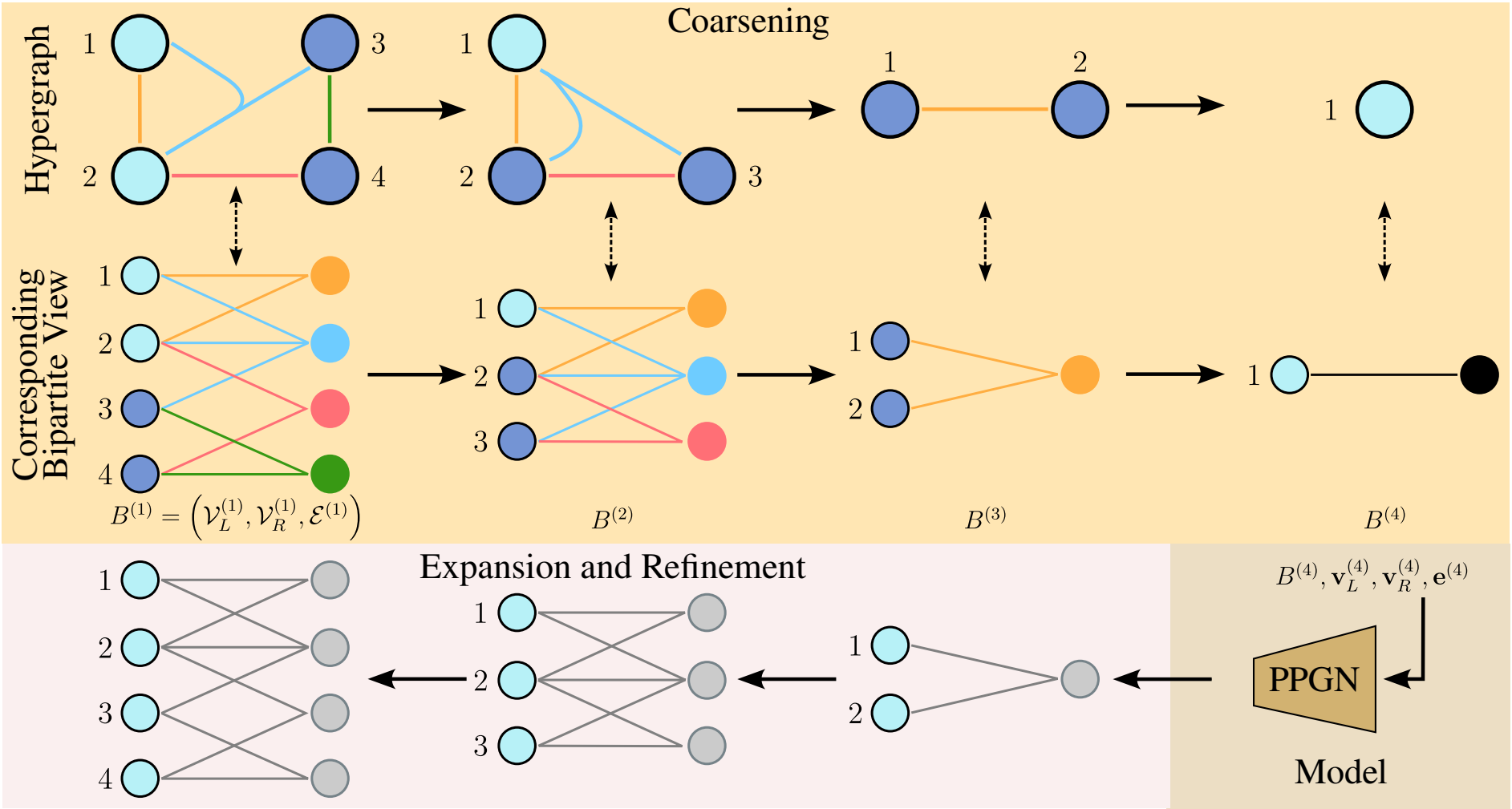


(c) Star Expansion

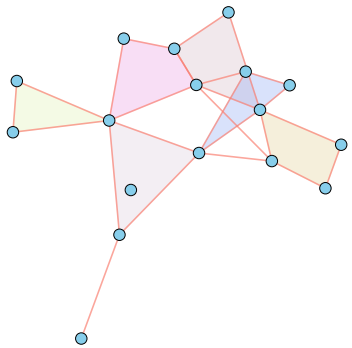
Figure: Comparison of the original hypergraph and its clique and star expansions.

- **Clique expansion:** hyperedges are collapsed into cliques (all pairs of nodes are connected by a regular edge). Can be weighted.
- **Star expansion:** One side correspond to nodes, the other to hyperedges, and each hyperedge is connected to all its nodes.

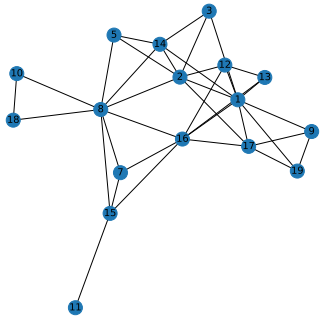
Pipeline


Hypergraph
Coarsening
Corresponding Bipartite View
$B^{(1)} = \left(\mathcal{V}_L^{(1)}, \mathcal{V}_R^{(1)}, \mathcal{E}^{(1)}\right)$
$B^{(2)}$
$B^{(3)}$
$B^{(4)}$
Expansion and Refinement
$B^{(4)}, \mathbf{v}_L^{(4)}, \mathbf{v}_R^{(4)}, \mathbf{e}^{(4)}$
PPGN
Model

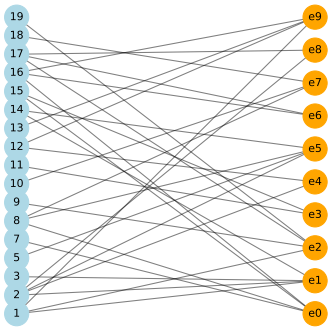
Why Different Projections



(a) Original Hypergraph



(b) Clique Expansion



(c) Star Expansion

- **Clique expansion:** if appropriately weighted, has the same laplacian as the hypergraph: spectrum-preserving graph coarsening can easily be applied. Suitable for downsampling, but getting the associated hypergraph is NP-hard.
- **Star expansion:** Suitable for generation: adding hyperedges and changing their content is easy (adding a right side node and connecting left-side nodes to it). Getting the associated hypergraph is easy.

Coarsening Sequence via Weighted Clique Expansion

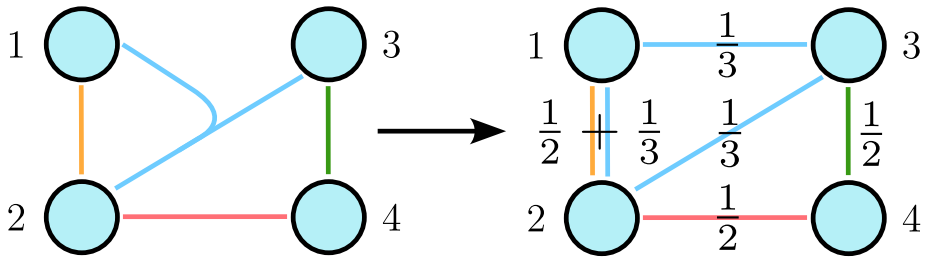
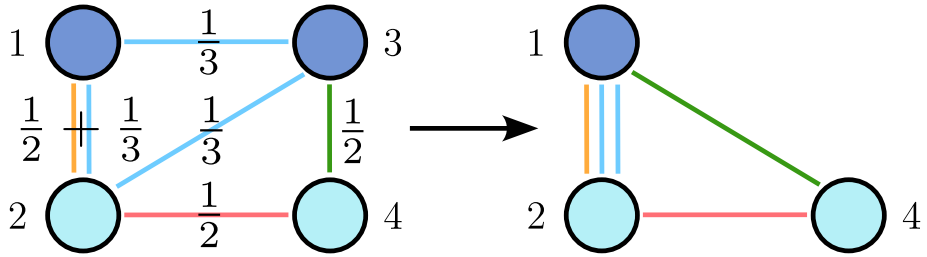


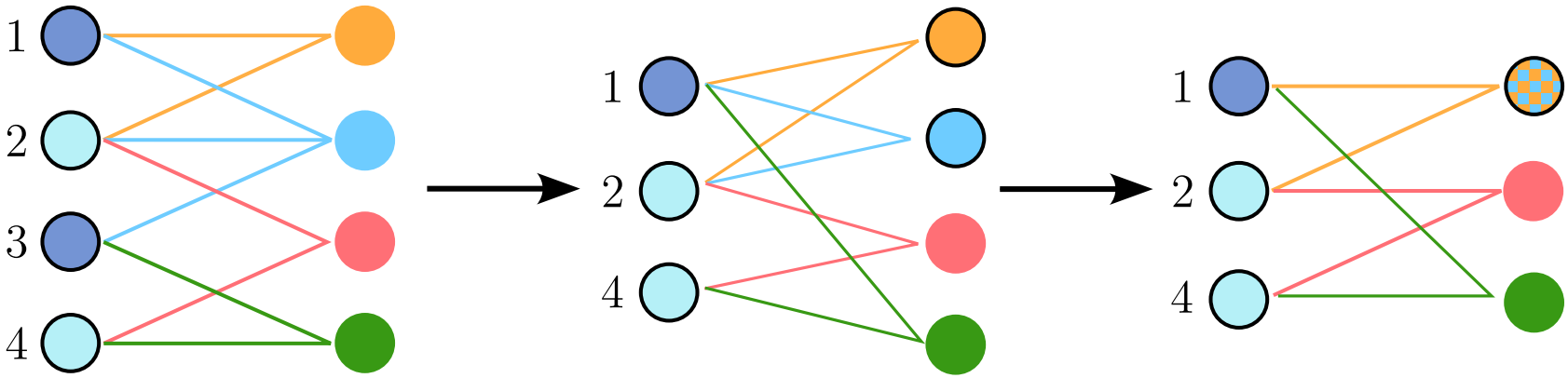
Figure: Each hyperedge is collapsed into a clique (i.e. we connect the nodes in the hyperedge via regular edges) and we attribute the weight $1/|e|$ - where $|e|$ is the size of the hyperedge - to these regular edges

- We select the nodes to merge via the standard procedure for graphs applied to the weighted clique expansion
- We maintain a parallel bipartite view of the hypergraph
- The final level of coarsening for the bipartite representation is two linked nodes with one on each side

Bipartite View Update



(a) A coarsening step on the weighted clique expansion. Weights are omitted because they are important only for the initialization of the procedure.



(b) Update of the equivalent bipartite representation

Figure: We first update the left side of the bipartite representation then the right side by merging the nodes representing the same hyperedge

Some Remarks

- We have the following result : for a single merging of a pair of nodes on the left side, each merging of nodes on the right side involves at most three nodes.
- Only true for a single merging, but at each coarsening step there can be many. If the mergings are not carefully chosen, a chain reaction can happen where tens of hyperedges can merge at once.
- To circumvent this, in practice we select the mergings on the left side such that their neighborhoods are disjoint.

Generation Procedure

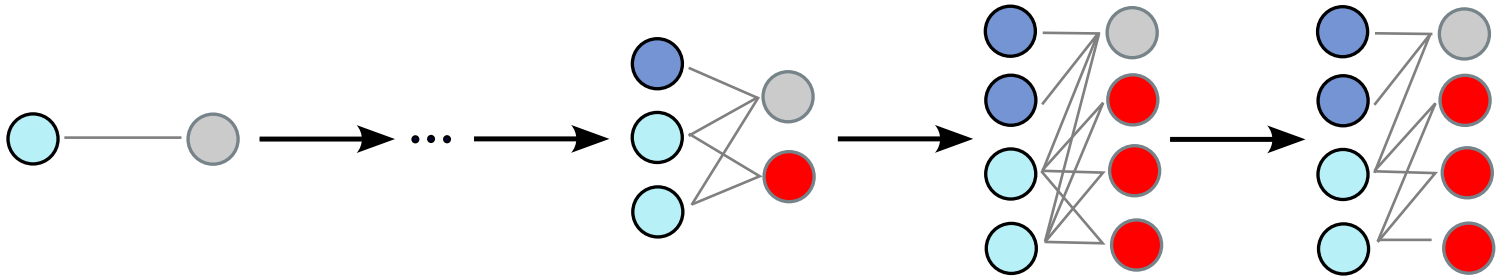


Figure: Generation of a hypergraph by our method

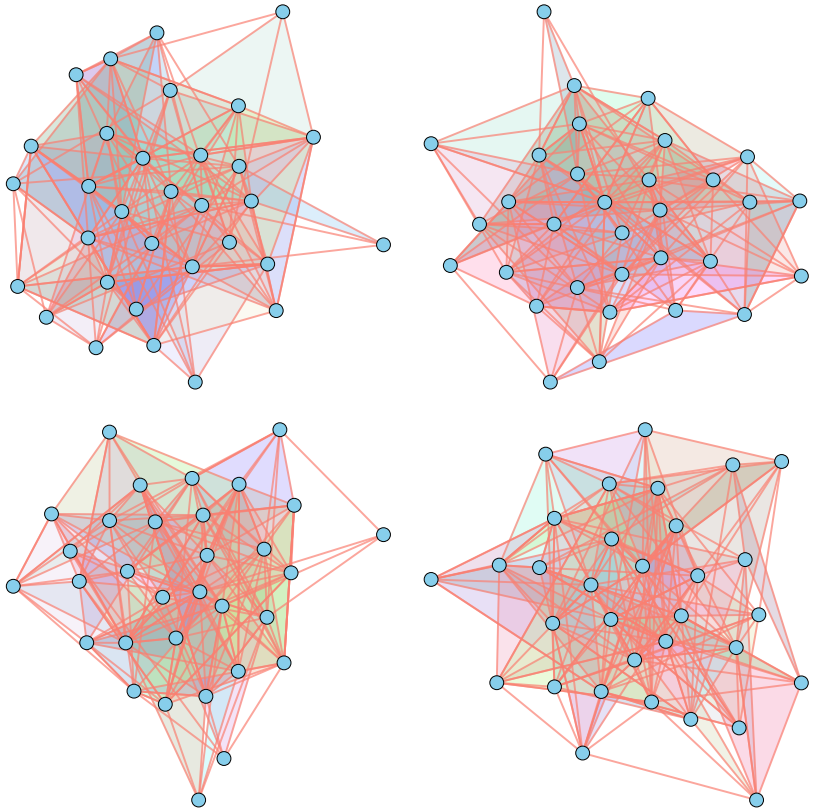
- Analogous to the simple graphs case applied to the bipartite representation:
  1. Select nodes on the left to duplicate (up to 2 times)
  2. Select nodes on the right to duplicate (up to 3 times)
  3. Choose which edges to keep in the bipartite graph
- Minor difference wrt Bergmeister *et al*'s paper: when duplicating a node, we do not add intra-cluster links.

Results

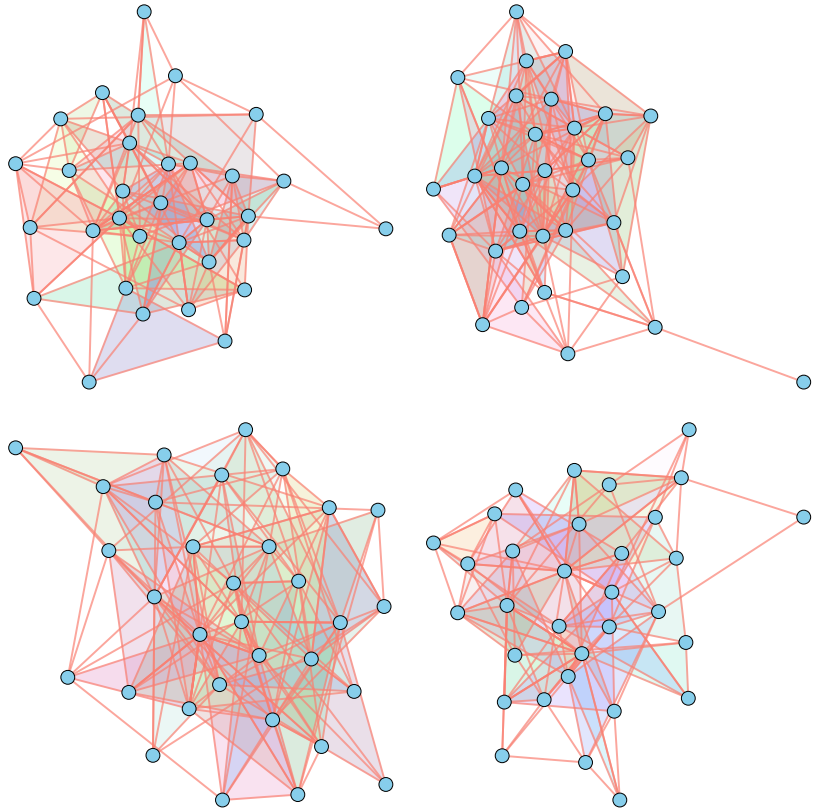
Results

- **Objective:** check that our method can imitate various properties
- Tested on several datasets: four synthetic, three taken from ModelNet40 (3D shapes, we removed the positions of the nodes)

Erdos-Renyi Hypergraphs

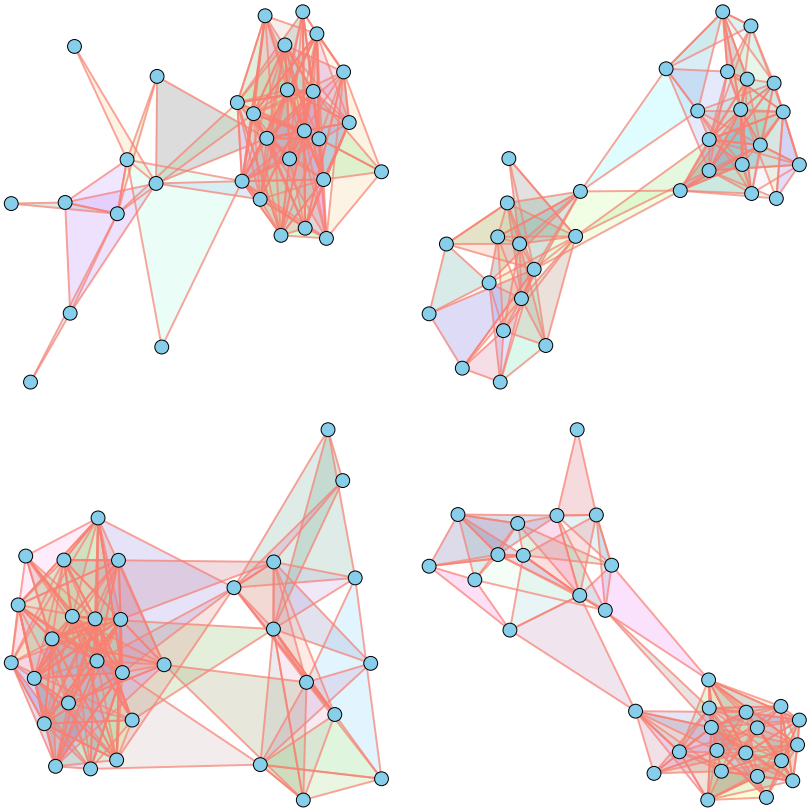


(a) Train samples

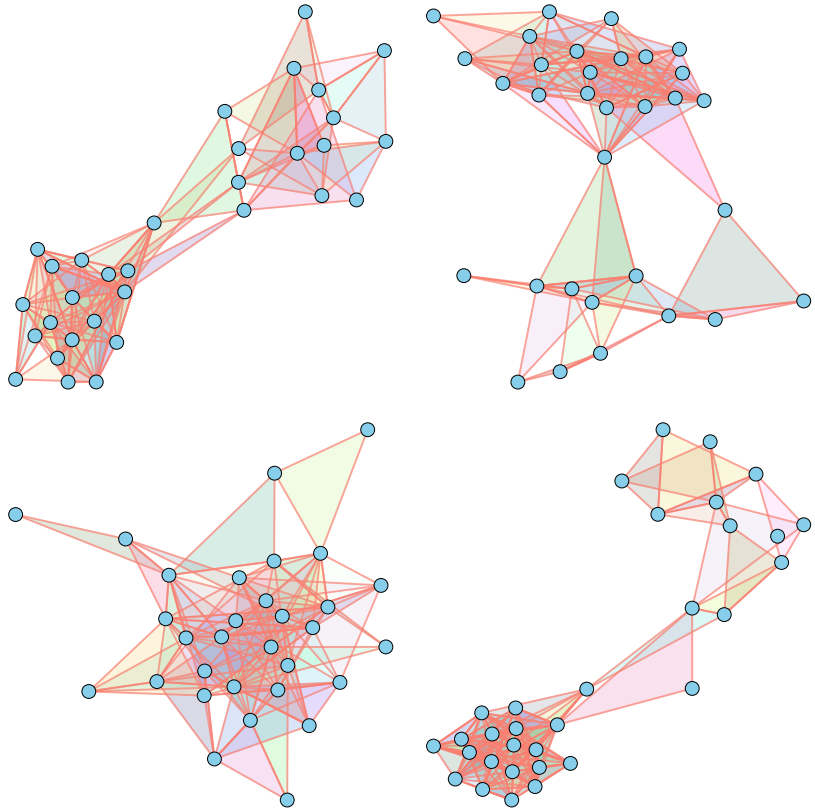


(b) Generated samples

Stochastic Block Model Hypergraphs

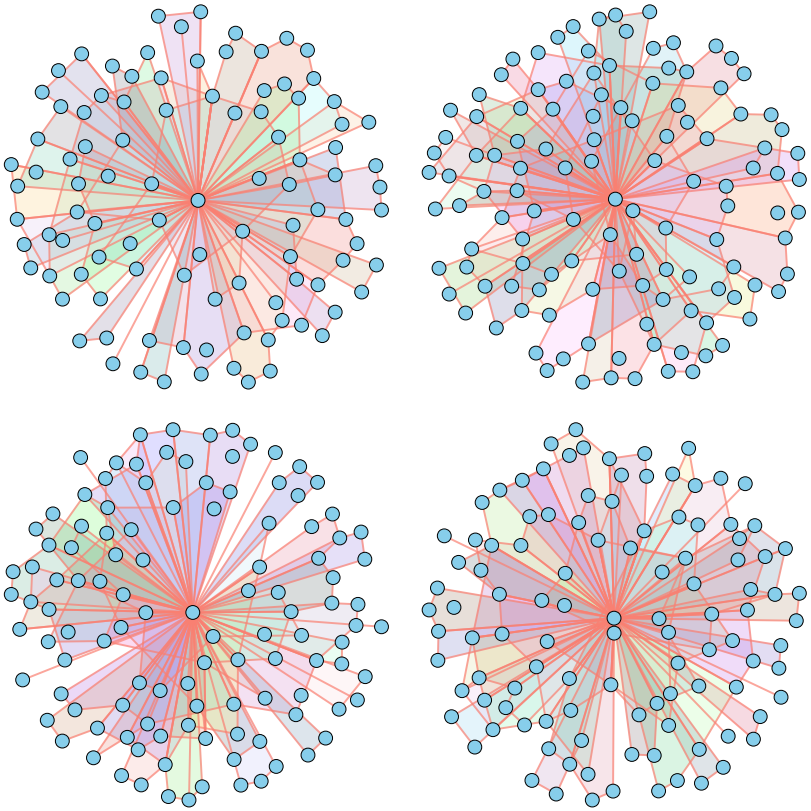


(a) Train samples

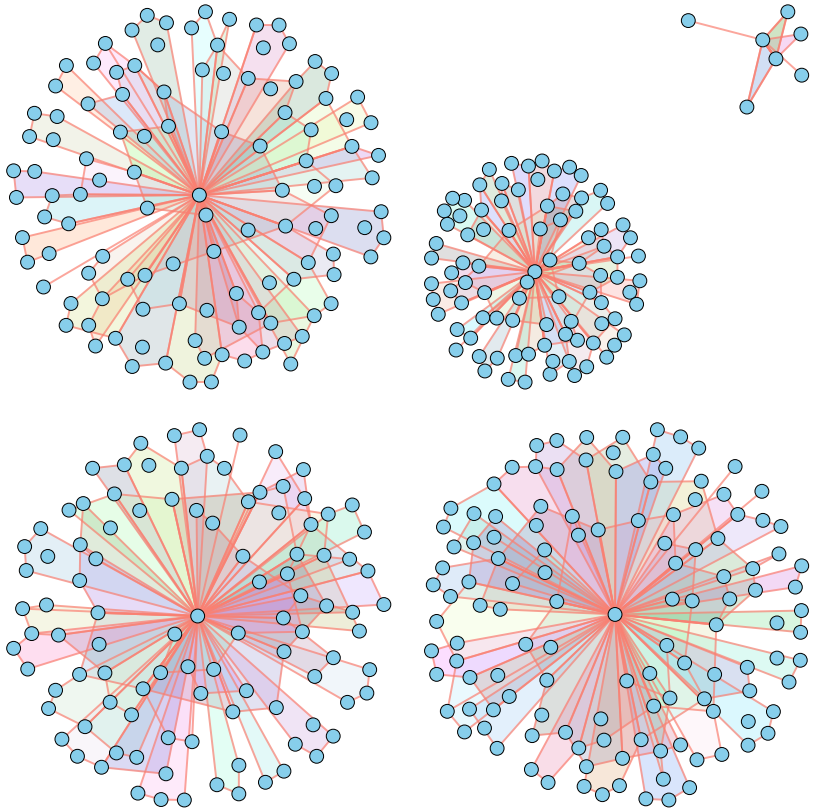


(b) Generated samples

Ego Hypergraphs

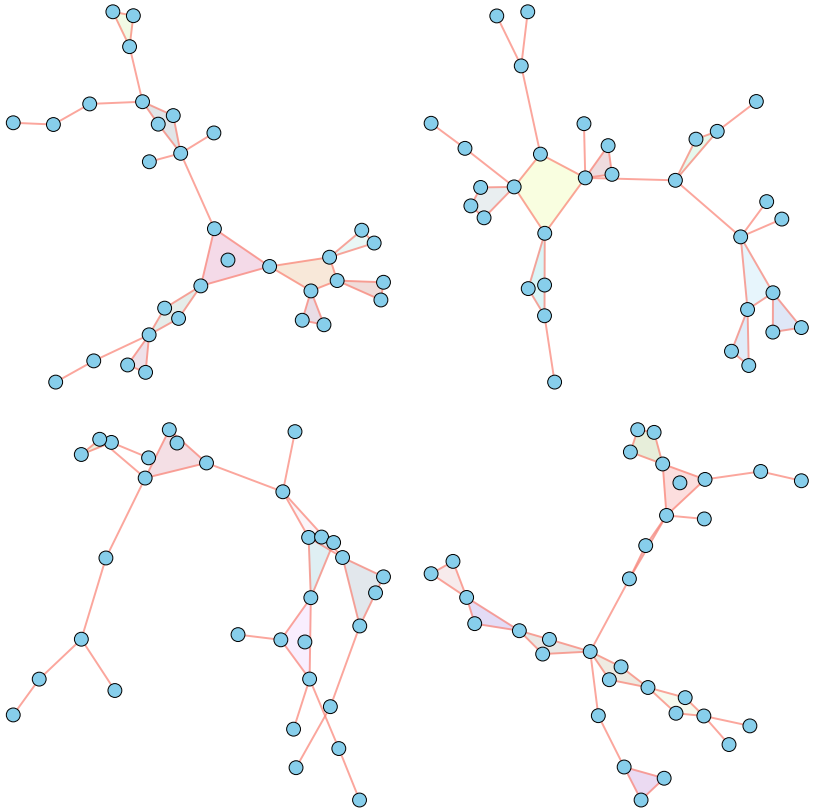


(a) Train samples

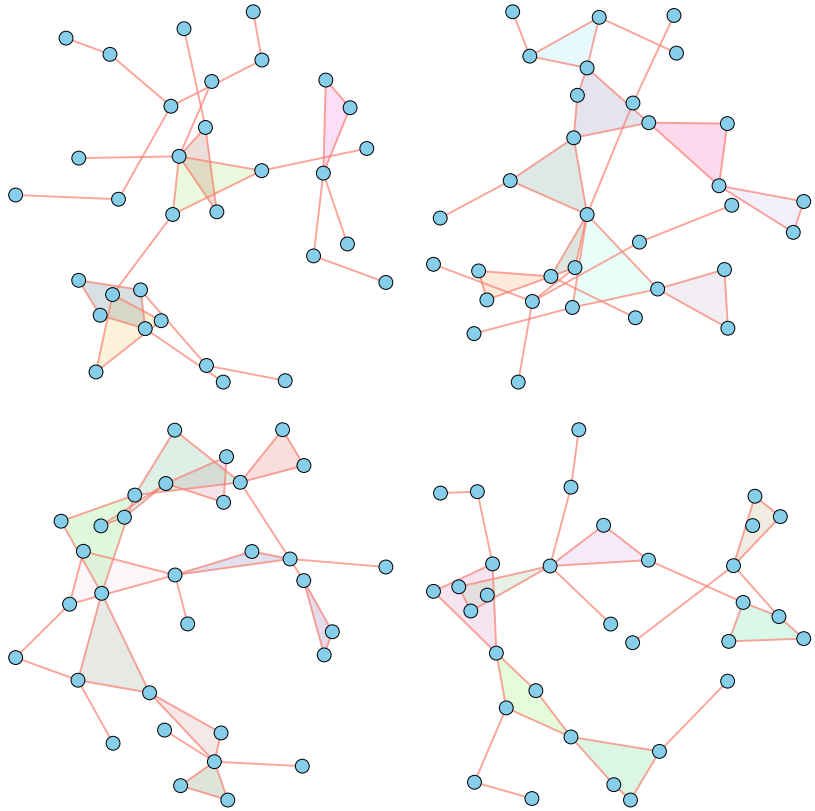


(b) Generated samples

Tree Hypergraphs

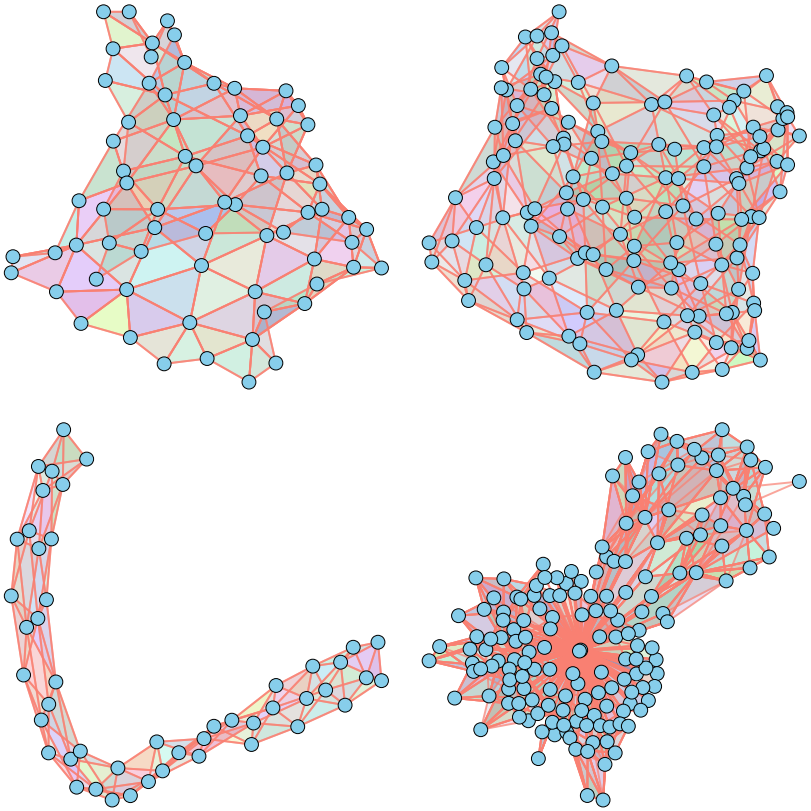


(a) Train samples

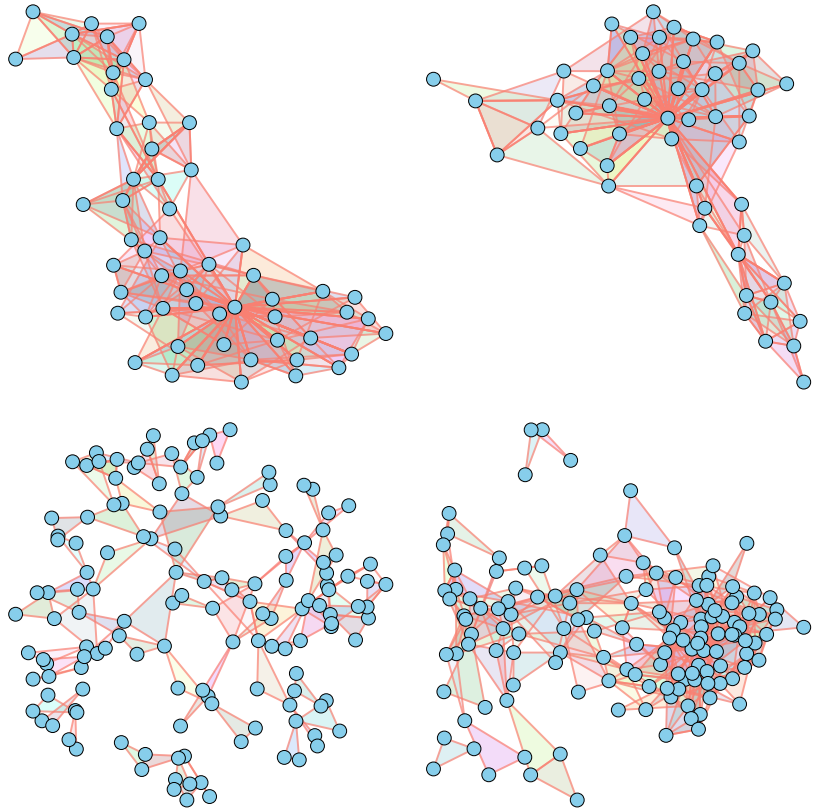


(b) Generated samples

Plant Meshes Topology

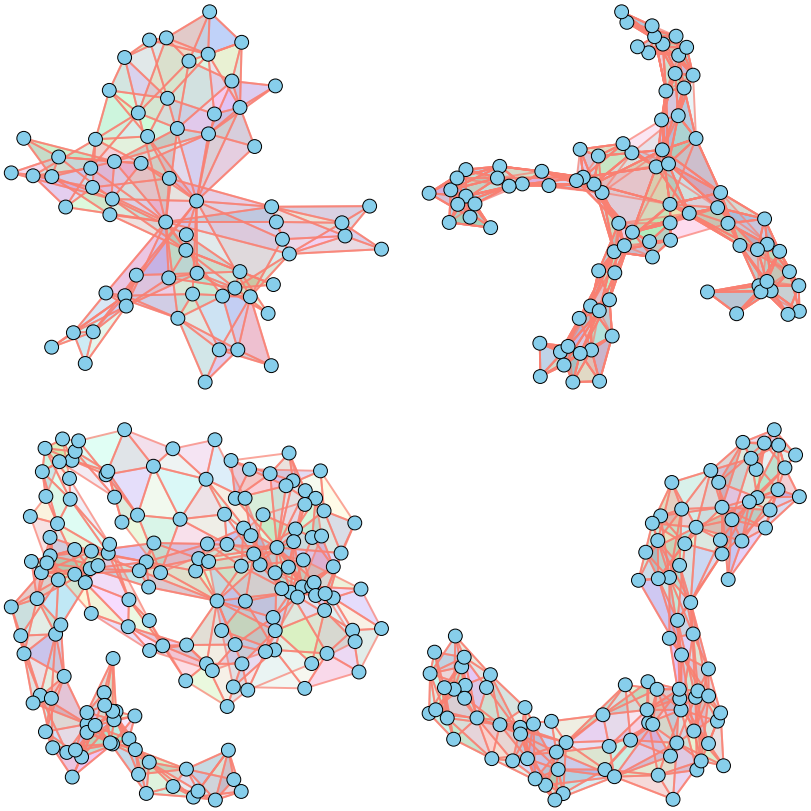


(a) Train samples

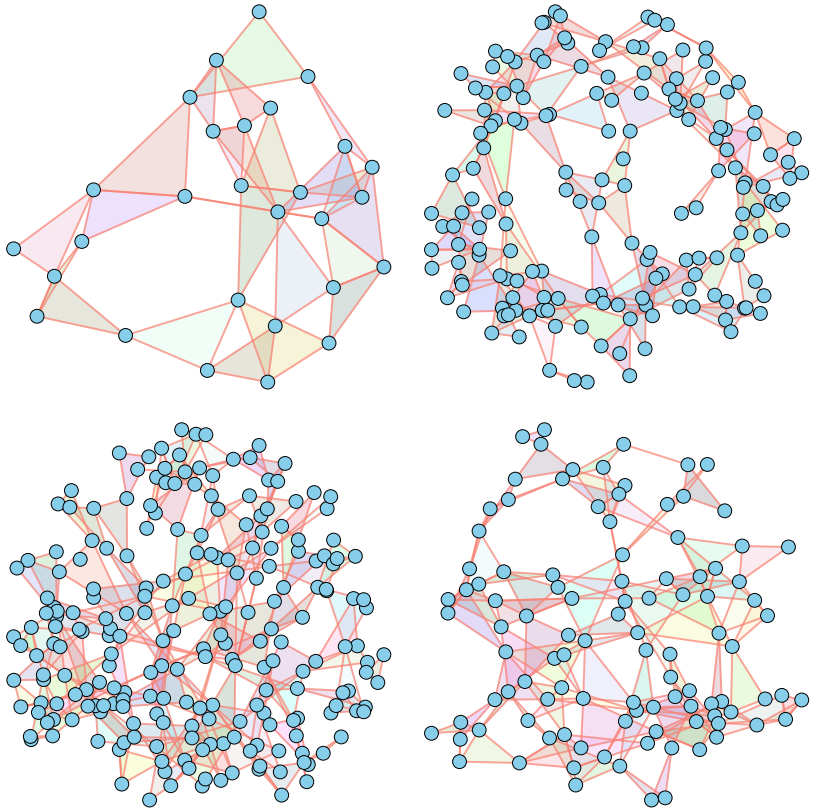


(b) Generated samples

Bookshelf Meshes Topology

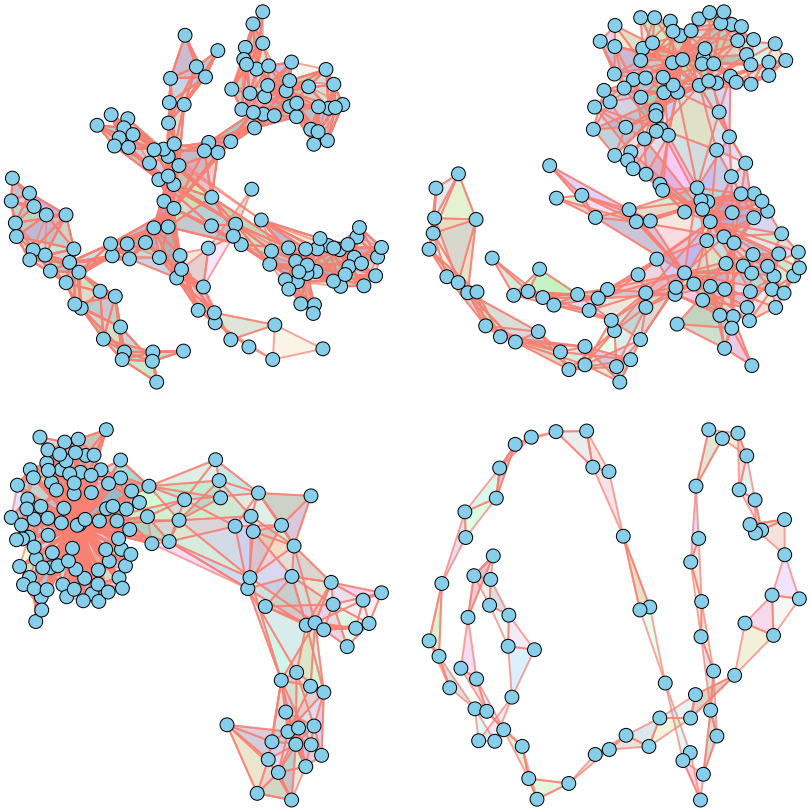


(a) Train samples

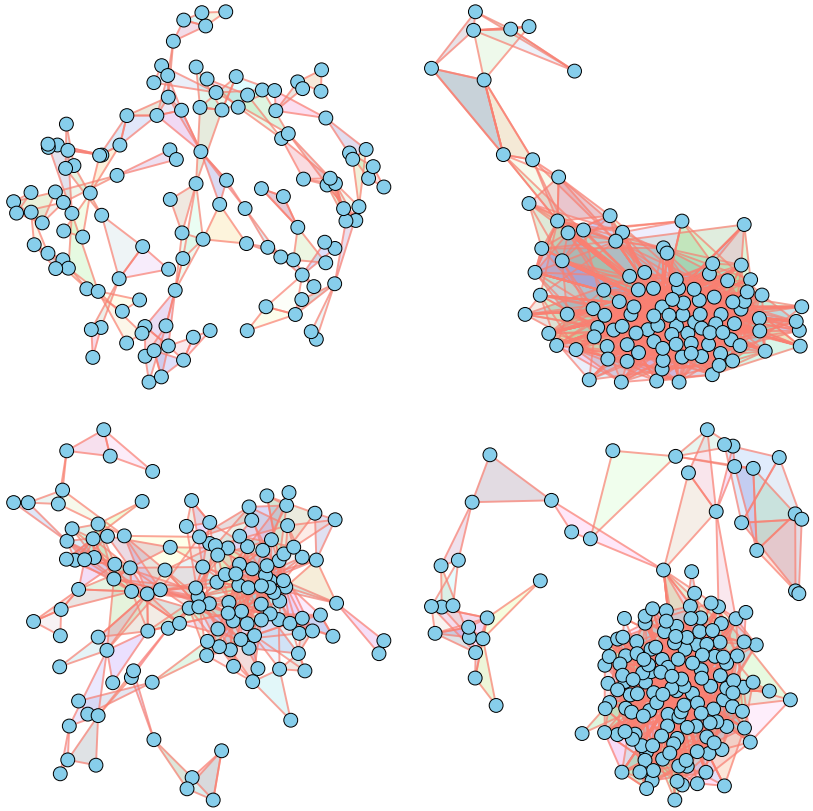


(b) Generated samples

Piano Meshes Topology



(a) Train samples



(b) Generated samples

Limitation / Current work

Limitations 1: Missing Nodes

- Does not generate the correct number of nodes
- The average number of missing nodes seems to closely approximate the variance in nodes number of the dataset
- **Hypothesis:** it is too difficult for the model to determine how many nodes are in the hypergraph as it requires a lot of information flow
- **Solution (?):** move the information from a global "distribution" to a "local" one. Instead of letting the model counts the number of nodes, give a budget of remaining nodes to create and split it between nodes during duplication. Generation would start with a single node having the budget for all the hypergraph.

Limitation 2: No Features

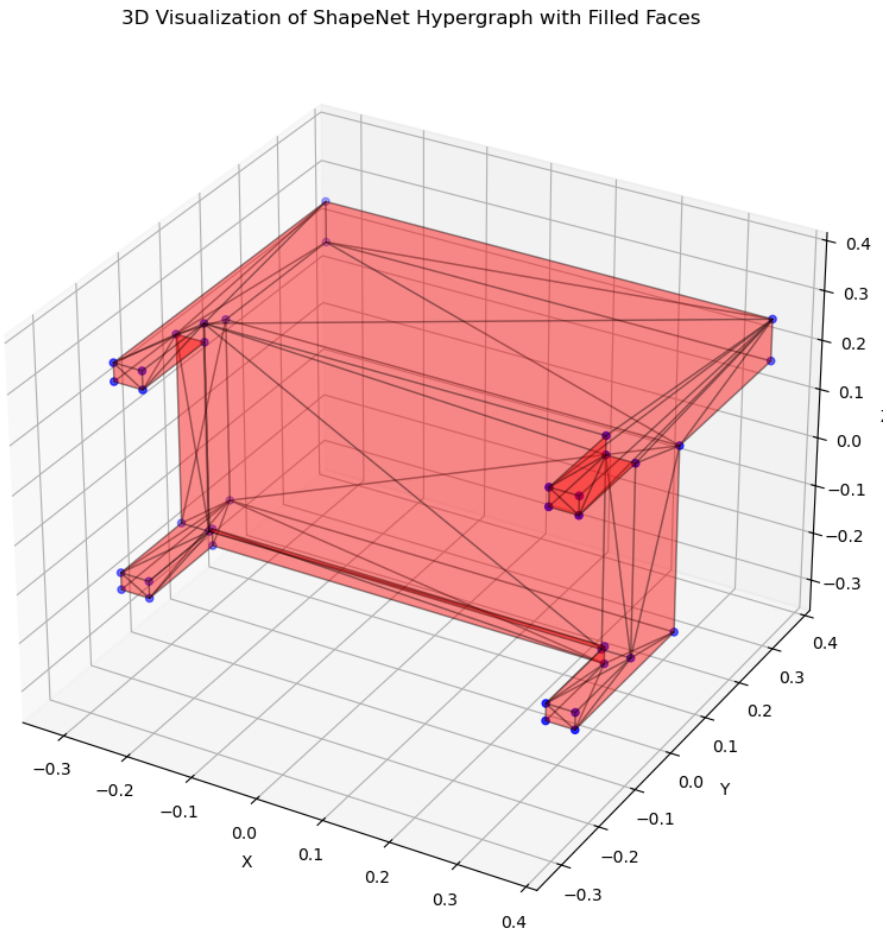
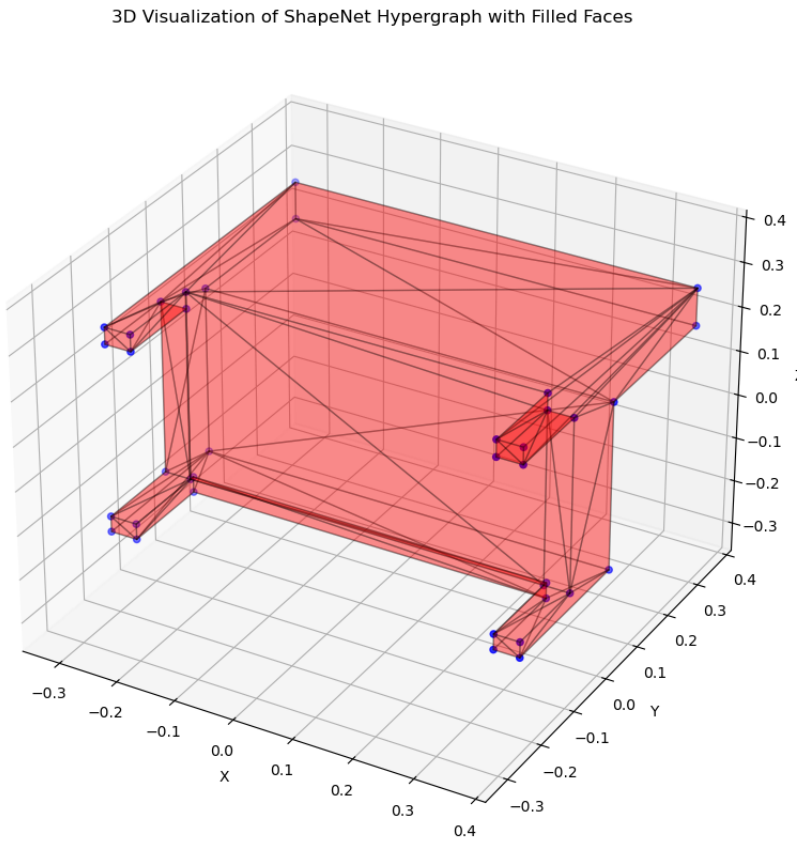


Figure: So far we cannot generate 3D positions

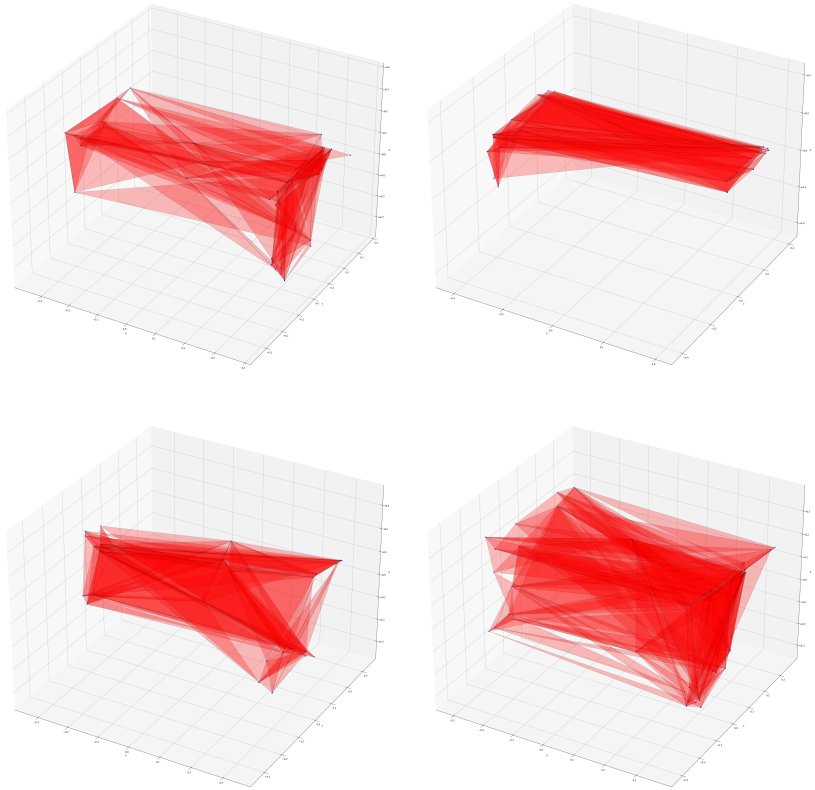
Current Work

- **Idea:** follow the same paradigm of putting the information "locally" to take a "local decision".
  1. When merging nodes, merge their features (how ? will depend on the manifold).
  2. When duplicating a node, both children get the feature of their parent.
  3. A model is trained to reconstruct the real features.

Current Work



(a) Training set



(b) Generated