Graph neural networks for solving the multicut problem

Oleh Shkalikov (5102818)

MLCV Seminar, TU Dresden

1 Summary

This expository article summarizes the work of Jung and Keuper (2022), focusing on ways to improve proposed approach which based on heuristics, ideas from research papers of Chen et al. (2019) on instance segmentation and self-prior learning introduced by Hanocka et al. (2020).

2 Preliminaries

The original paper consists of two basic parts: graph neural networks and minimum cost multicut problem.

2.1 Graph neural networks

Graph neural network is a type of neural network which works on arbitrary graphs and can be described, as it was shown by Gilmer et al. (2017), with use of message-passing scheme.

Definition 1. For the given differentiable update function $\gamma : \mathbb{R}^{d_2} \to \mathbb{R}^{d_3}$, differentiable message function $\phi : \mathbb{R}^{d_1} \times \mathbb{R}^{d_1} (\times \mathbb{R}^m) \to \mathbb{R}^{d_2}$, and commutative differentiable aggregation operation $\bigotimes : \mathbb{R}^{d_2} \times \mathbb{R}^{d_2} \to \mathbb{R}^{d_2}$, the update rule for every node i at step t with node feature vector $\mathbf{x} \in \mathbb{R}^{d_1}$ (where d_1 is a dimensionality of the node feature space), neighborhood $\mathcal{N}(i)$ and edge feature vector $\mathbf{e} \in \mathbb{R}^m$ (optional, m is a dimensionality of edge feature space) is

$$\mathbf{x}_i^{(t+1)} = \gamma^{(t)} \left(\mathbf{x}_i^{(t)}, \bigotimes_{j \in \mathcal{N}(i)} \phi^{(t)} \left(\mathbf{x}_i^{(t)}, \mathbf{x}_j^{(t)}, \mathbf{e^{(t)}}_{j,i} \right) \right)$$

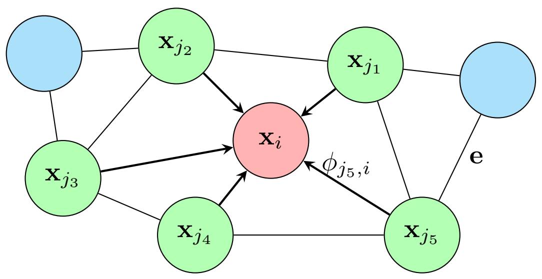

Figure 1 Illustration of the message passing scheme

Different graph neural networks architecture can be constructed by specifying all this functions, in particular the update function, since aggregation operation usually is chosen from the fixed list of functions such as min, max, mean, sum, product, etc.

Also it is worth to point out, that the graph neural network has a property of locality: the next state of the node depends only on the current state of the adjacent nodes (and incident edges if they exist), therefore the receptive field of every node is limited to the number of layers or number of applying the layer to the given graph (usually on each step different layers are applied).

2.2 Minimum cost multicut problem

The minimum cost multicut problem is a combinatorial problem which consists in binary edge labeling for a given, not necessarily complete, weighted graph. The resulting labeling induced a partitioning of the graph and can be recognized as an analogue of an instance segmentation for graphs and that's why has a lot real-world application in computer vision and other fields.

Definition 2. Let $G = (V, E, w)$ be a connected weighted graph, where V is a set of nodes, E – set of edges and $w : E \to \mathbb{R}$ is a weight/cost function. Then the minimum cost multicut problem[1] is the following:

$$\min_{y \in \{0,1\}^E} \quad c(y) = \sum_{e \in E} w_e y_e \tag{1}$$

$$\text{subject to:} \quad y_e \leq \sum_{e' \in C \setminus \{e\}} y_{e'} \tag{2}$$

$$\forall C \in \text{cycles}(G), \forall e \in C$$

The label 1 means that the corresponding edge is cut. And the cycle constraints ensure that if the edge was cut then there is no other path in the graph which connects incident nodes of this edges.

This constraints can be reduced to only chordless cycles as it was shown by Chopra and Rao (1993), but still for the non complete graphs (where triangles inequality is sufficient) the constraints grows with size of graph exponentially, so can be their enumeration can be practically infeasible.

From the point of view of hardness of this integer linear programming problem it is worth to mention that minimum cost multicut is NP-hard.

[1]The original paper has a typo in the definition of the problem

3 Problem statement

In their article Jung and Keuper (2022) address the problem of applying graph neural networks to the minimum cost multicut problem in order to optimize runtime, but with preserving an accuracy (in terms of cost) of the solution. And since every neural network consists of data, architecture, loss function and training process, answering these question is one of the main task of the paper as well.

4 Conceptual contributions

4.1 General pipeline

The main conceptual contribution of the paper is a pipeline of applying the graph neural network to the instance of minimum multicut problem defining by graph. This pipeline can be expressed in the following way.

Algorithm 1: GNN multicut pipeline

```
Input: $G = (V, E, w)$ – graph
Result: $y \in \{0,1\}^E$ – binary labeling of the edges
foreach $i \in V$ do
    $\mathbf{x}_i \leftarrow \left( \sum_{j \in \mathcal{N}^+(i)} w_{ij}, \sum_{j \in \mathcal{N}^-(i)} w_{ij} \right)$ ;
    $\mathbf{h_i} \leftarrow \mathrm{GNN}(\mathbf{x_i})$ ;
end
foreach $e_{ij} \in E$ do
    $y_{ij} \leftarrow \frac{1}{2}\left(\mathrm{MLP}\binom{\mathbf{h_i}}{\mathbf{h_j}} + \mathrm{MLP}\binom{\mathbf{h_j}}{\mathbf{h_i}}\right) > 0.5$ ;
end
Enforce cycle consistency via message passing
```

Generally, the idea of the approach can be expressed as transforming of the problem of graph partitioning into the real value clustering problem of node's embeddings, which can be solved by a various methods: in this case with use of MLP. Then, edges which are incident to the nodes which belong to the different clusters should be cut and edges which corresponds to the nodes of the same cluster should be preserved.

To perform this transformation from graph to real value node's embeddings an instance of the graph neural network is being used.

In order to perform the calculation of node embeddings by the message passing approach the input of graph neural network has to contain initial node features. But since an instance of the minimum cost multicut problem has only the edge weights authors have proposed the method of computing these features: for each node they provide 2 dimensional vector where the first component is the sum of all positive incident edge weights and the second component – sum of all negative incident edge weights.

Then high dimensional node's embeddings is computed by applying a graph neural network, which will be discussed in details in the next subsection. The main point to highlight in this regard is that message passing step can be run for every node in parallel on GPU which significantly improve runtime of the full pipeline and which is easy to implement because of existence of a lot deep learning libraries which provide GPU acceleration out of the box.

On the next step edge classification is performed with use of MLP, i.e. the node's embeddings $\mathbf{h_i}$ and $\mathbf{h_j}$ of the incident edge ij stacked together in two vectors $\binom{\mathbf{h_j}}{\mathbf{h_i}}$ and $\binom{\mathbf{h_i}}{\mathbf{h_j}}$. The mean of the MLP with sigmoid head output is used as a confidence whether an edge should be cut. The final prediction computed by thresholding this value at 0.5.

MLP based edge classification requires two vectors because the input graph is considered to be undirected and MLP is not permutation invariant: the order of inputs mattes. Also the averaging of the MLP outputs is not so expressible and reliable way to compute the labeling. The ideas of how to improve this step will be discussed in section 6.1.

The computed solution can be infeasible since there is no guarantee that cycle consistency (2) won't be violated. That's why the postprocessing step is required. This step consists in rounding (uncuting wrongly cut edges) of the computed labeling to enforce the feasibility of the solution. It can be computed by a message passing approach, but since neither source code nor detailed explanation has been provided by authors we propose our own algorithm 2.

Algorithm 2: Cycle consistency enforcing

```
Input: $G = (V, E, w)$ – graph, $y \in \{0,1\}^E$ – binary
       labeling of the edges
Result: $\hat{y} \in \{0,1\}^E$ – feasible solution
for $i \in \{1, \ldots, |V|\}$ do
    $c_i = i$ // initial class label
end
for $n \in \{1, \ldots, diag(G)\}$ do
    $c \leftarrow \mathrm{MPN}(c)$ // update rule (3)
end
foreach $e_{ij} \in E$ do
    $\hat{y}_{ij} \leftarrow c_i = c_j$ ;
end
```

For the given graph on the first step for every node we assign different labels which means that we start with totally separated clusters where only 1 node belong to every particular cluster. This labels is used as a node's features in the following step.

The key component of the postprocessing step is an message passing rule, which is the following:

$$c_i^{t+1} = \min\left\{c_i^t, \min_{j \in i \cup \mathcal{N}(i)} \left\{(1 - y_{ij})c_j^t + y_{ij} \cdot \infty\right\}\right\} \tag{3}$$

This message passing network on every their step relabel nodes in a way that the resulting label is equal to the minimum label among the node itself and their adjacent nodes unless adjacent node has been cut. So, this step is just some variation of a connected component algorithm. But since it is implemented as a message passing manner it can be parallelized on GPU in the same way as a GNN. But to be sure that we complete finding all components (merge every component which should be merged for the worst case: linked list) message passing relabeling has to be run

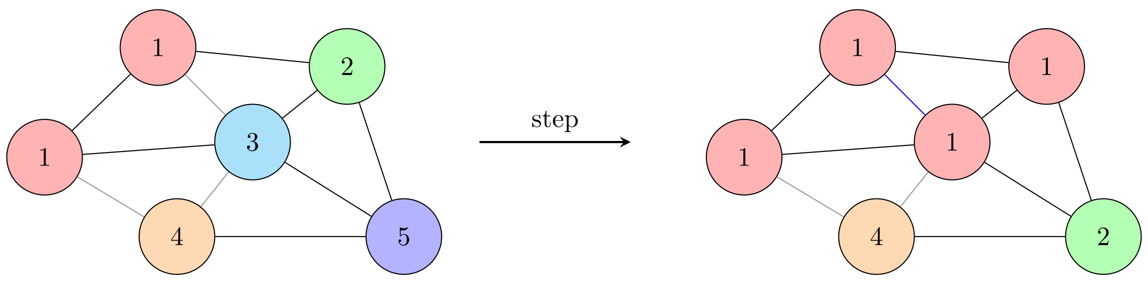

Figure 2 Illustration of the step of the message passing postprocessing network: colors and labels in nodes corresponds to the cluster of nodes: same color and label – same cluster. Black edges is a uncut edges of the original graph G and gray – cut edges of G. After a message passing step 3 nodes which have been connected by uncut edges changed their labels to the minimal label among the neighborhood. At the same time the node with label 4 didn't changed their label because it is not connected by uncut edge to any node with smaller label on this step. Also we denote with blue color edge that became incident to nodes from the same cluster and therefore on the last step of the postprocessing algorithm will be restored.

number of steps which is equal to diameter of a given graph (but in real application can be bounded by a relatively large number or computed precisely if the structure of graph (e.g. grid graph) is known). The example of the step of a message passing network is given on figure 2.

The last step of postprocessing algorithm is responsible for uncut wrongly cut edges by restoring edges which belongs to the same cluster (that means that there is a path in graph made by uncut edges which connect this two nodes and therefore constraints (2) have been violated).

4.2 Graph neural network architectures

While all other steps of GNN multicut pipeline 1 don't require additional changes, architectures of graph neural network should be adjusted to be able to incorporate edge weights into computation since originally minimum cost multicut problem has only this quantities and a lot popular models can not work properly without alteration with one dimensional real-valued edge features.

As a main baseline authors of paper have chosen the Graph Convolutional Network by Kipf and Welling (2016). The update rule of this architecture is the following:

$$\mathbf{x}_i^{(t+1)} = \gamma_\theta \left(\mathbf{x}_i^{(t)} + \sum_{j \in \mathcal{N}(i)} \mathbf{L_{ij}} \mathbf{x}_j^{(t)} \right),$$

where update function γ_θ is a matrix-vector multiplication with matrix of parameter θ followed by non-linear function like ReLU, $\mathbf{L_{ij}} = \frac{1}{\sqrt{\deg(i)\deg(j)}}$ and $\mathbf{L} = \mathbf{\tilde{D}^{1/2}\tilde{A}\tilde{D}^{1/2}}$ – normalized graph laplacian, $\tilde{D}$ – degree matrix and $\tilde{A}$ – adjacency matrix with additional self-loops.

In order to allow this model to work with negative valued weights (which is necessary because otherwise the solution is trivial) authors proposed to change the message function by multiplying by weight value and replacing degree of node with similar but non error-prone quantity $\overline{\deg}(i) = \sum_{j \in \mathcal{N}(i)} |w_{ij}|$. Thus the update rule of updated Graph Convolutional Network for minimum cost multicut problem is the following:

$$\mathbf{x}_i^{(t+1)} = \gamma_\theta \left(\mathbf{x_i^{(t)}} + \sum_{j \in \mathcal{N}(i)} \frac{w_{ij}\mathbf{x}_j^{(t)}}{\sqrt{\overline{\deg}(i)}\sqrt{\overline{\deg}(j)}} \right)$$

Despite the fact that all further experiments from the main content of the paper use only this architecture it is worth to point out that authors also proposed updated version of Signed Graph Convolutional Network by Derr et al. (2018) and Graph Isomorphic Network by Xu et al. (2018). The explanation of choice has not been provided but the given in the appendix graphs allow us to conclude that decision was based on the stability in terms of training and evaluation loss especially in the case of Graph Isomorphic Network which has shown big fluctuations. In addition experiments on some real problems (which will be described further) have shown that the accuracy of predictions of Graph Convolutional Network significantly higher than the metric value of all others.

4.3 Loss function

Having determined the architecture the next important point in training neural networks and in particular graph neural network is to define the loss function. In this regard authors decided to formulate the minimum cost multicut problem as a supervised learning problem with loss which consists of two parts. They use binary cross entropy loss as the first part of the loss. This part of the loss is responsible for minimizing the cost (1) of multicut by making the predicted solution as close as possible to optimal.

The second part of the loss has been proposed by authors and based on cycle consistency constraints (2) and responsible for enforcing feasible solutions. They called id CCL (cycle consistency loss) and it is computed as follows:

$$\mathcal{L}_{\text{CCL}} = \sum_{C \in cc(G,l)} \sum_{e \in C} \hat{y}_e \prod_{e' \in C \setminus \{e\}} (1 - \hat{y}_{e'}), \qquad (4)$$

where $cc(G, l)$ – chordless cycles with length less or equal to l. This upper boundary is determined experimentally and necessary since the number of cycles grows exponentially.

The total loss is computed as a weighted sum of these losses. But it worth to point out that supervised formulation is unnatural for the minimum cost multicut problem and ideas to get rid of this type of formulation will be discussed in further sections as well as formulation of loss which doesn't require enumeration of all chordless cycles in the graph.

4.4 Training process and datasets

To test their methods authors have trained a lot of networks changing the following parameters: architectures that we have already discussed in the section 4.2, with and without using the cycle consistency loss (4). In addition they varied the architecture of the Graph Convolutional Network in particular trained the plain (original) model, model with edge weight but without adding signed normalization term $\left(\overline{\deg}(i)\overline{\deg}(j)\right)^{-\frac{1}{2}}$ and models with both of this addition. They tested different cycle length for the the cycle consistency loss (4), batch normalization, depth (number of message passing layers) and dimensionality of embeddings.

Since the proposed formulation is a supervised problem, it requires labeled dataset in order to train GNN and MLP. Despite the fact that the minimum cost multicut problem is widespread number of publicly available problem instances with provided optimal solutions is limited. To beat this problem the artificially generated datasets has been created. The final training datasets consists of 2 datasets called IrisMP and RandomMP.

IrisMP datasets consists of minimum cost multicut problem instances defined in a complete graphs and has been generated based on the well-known Iris flower dataset Fisher (1936) by the following algorithm: first of all, for each graph only 2 column of the original dataset has been drown uniformly at random. Then uniformly from 16 to 24 rows has been selected as a nodes and since the graph is complete all nodes has been connected. Edge weighs have been computed as a output of the logit function applied to a result of computation of Gaussian kernel with $\sigma = 0.6$ to L_2 distances (with respect to the value of 2 selected from the original Iris dataset columns) between the nodes. In accordance to this algorithm 20000 of problem instances has been generated for training and 1000 graphs for test and validation splits.

RandomMP dataset contains not complete graphs but with larger number of nodes. The size of this dataset is equal to the size of IrisMP. To generate this dataset the following algorithm has been used: the number of nodes for each each graph was sampled from the normal distribution with parameters $\mu = 180, \sigma = 30$. For each node of the graph its coordinate from $[0, 1]^2$ set has been sampled. Nodes was connected to the k nearest neighbors, where k was also sampled from the normal distribution with $\mu = 180, \sigma = 30$ but constrained to be at least 1. For computing edge weights from the L_2 distance between nodes on the plane the median value of distance has been subtracted in order to end up with approximately equal number of positive and negative.

It is not clear from the paper how the optimal solutions for these datasets has been computed. The source code of the paper has not been provided as well. So, with high probability we can consider that these solutions has been computed with use of some ILP solver (like Gurobi Optimization, LLC (2023)), since determining labels based on the generation method (for example determine whether edge between nodes of the graph from IrisMP dataset should be cut or not based on corresponding flower's species (different species means cut)) may lead to suboptimal solutions.

In addition, it is worth to mention that problem instances from artificially generated datasets can significantly differ from the real-world problem instances which can lead to lower performance of the model trained on this data. But with use of this approach authors could generate sufficiently large dataset for training neural networks.

As a metric for the evaluation authors proposed the optimal objective ratio:

$$m = \max\left\{0, \frac{c(\hat{y})}{c(\tilde{y})}\right\},$$

where c is a an objective or cost (1), $\hat{y}$ – predicted solution, $\tilde{y}$ – optimal solution. This metric is computed on postprocessed output of the pipeline 1 and can be considered as better than classical classification metrics like F1, precision and recall since the main objective of the optimization task is to minimize the total cost and therefore incorrect predictions of edge labels has different impact (higher absolute value of the weight – higher impact) to the cost whereas classical metrics considered this impact as equal for all edges.

5 Empirical contributions

For the evaluation in addition to generated instances of problem authors used dataset Andres et al. (2011) based on the Berkeley Segmentation Dataset (BSDS300) Martin et al. (2001), volume segmentation dataset Knott3D Andres et al. (2012) and Circuit Reconstruction from Electron Microscopy Images (CREMI) Beier et al. (2017). The proposed approach has been compared to exact ILP solver as well as Branch & Cut LP (Kappes et al. (2015)) and GAEC (Keuper et al. (2015)) (time-bounded and not).

The main results of the experiment can be summarized as follows: the GNN pipeline 1 is the best solver in the context of trade-off between runtime and resulting cost of multicut. Whereas classical methods show better objective value, the GNN (in particular proposed GCN architecture) significantly faster and computed solution is comparable to the competitors. Time bounded GAEC in comparison to the discussed approach is tremendously worse. These results preserves with growing of the size of input problem instance.

Ablation study proves the efficiency of proposed cycle consistency loss (4) and changes committed to the GNN architecture. Additionally, authors denoted examples where for the particular instances of problem GNN generated embeddings for nodes such that their cosine similarities reflects the resulting cluster.

6 Ideas to improve the proposed approach

The proposed approach can be possibly improved by using additional heuristics, ideas from other methods and by analyzing the experimental result. In this section we will offer some ideas of how to do it.

6.1 Orthogonal embeddings

As it has been discussed in section 5, it is possible for model to produce node's embeddings for which cosine similarities will be enough to decide whether edge should be cut or not. The idea is to train a model explicitly to produce such embeddings. But since usual cosine similarities has 3 key values which corresponds to 3 "states": similar (value close to 1), orthogonal (value close to 0) and opposite (value close to -1), - whereas the labeling of edges (node pairs) has 2 states: uncut (nodes belong to the same cluster) and cut (nodes belongs to the opposite cluster) we propose to use the following quantity from the paper of Chen et al. (2019):

$$1 - |\cos\langle \mathbf{x}_i, \mathbf{x}_j \rangle| \tag{5}$$

By thresholding quantity (5) with some number $t \in [0, 0.5]$ we label edges between nodes in a way that adjacent nodes with collinear embeddings will belong to the same component (edge is not cut) and with orthogonal – to different components (edge is cut). And as it has been shown in the paper we don't need to care (if we chose sufficiently large embedding's dimensionality) about situation when non adjacent has collinear embeddings and utilize only local orthogonality.

The loss function which can help us to learn it is similar to the loss from Chen et al. (2019) as well:

$$\begin{aligned} \mathcal{L}_{emb} = & \lambda_1 \sum_{e_{ij} \in y^{-1}(0)} (1 - |\cos\langle \mathbf{x}_i, \mathbf{x}_j \rangle|) + & (6) \\ & \lambda_2 \sum_{e_{ij} \in y^{-1}(1)} |\cos\langle \mathbf{x}_i, \mathbf{x}_j \rangle|, & (7) \end{aligned}$$

where λ_1, λ_2 – some normalization weights which depends on the number of cut and non cut edges.

Unfortunately, this approach can not enforce cycle consistency in a transitive manner (suppose $t < 0.5$)

$$\begin{cases} 1 - |\cos\langle \mathbf{x}_i, \mathbf{x}_j \rangle| \le t \\ 1 - |\cos\langle \mathbf{x}_j, \mathbf{x}_k \rangle| \le t \end{cases} \implies$$
$$1 - |\cos\langle \mathbf{x}_i, \mathbf{x}_k \rangle| \le 1 - |2(1-t)^2 - 1| \ge t$$

As a result, this approach allow us to get rid of MLP classifier on the head of the GNN and give a meaningful explanation of node embeddings and edge classification / labeling.

6.2 Relaxed Cycle Consistency Loss

The existing cycle consistency loss (4) requires enumeration of all chordless cycles of length $\le l$ which can lead to enormous number of cycles since their number grows exponentially.

In the same time it has been shown by authors that increasing length of cycles doesn't improve the quality of the model. This explained by the fact that in a process of calculation we multiply a lot number in range $[0, 1]$. And whereas the problem of machine zero can be solved by exp-sum-log trick, the conceptual problem, that for large l the loss value will be very small and therefore impact of this wrong prediction is imperceptible, still exists.

To handle thess issues we propose the following relaxed cycle consistency loss:

$$\frac{1}{|E|} \sum_{e \in \tilde{E}} \hat{y}_e,$$

where $\tilde{E}$ – set of all cut edges which violate cycle consistency constraint (2).

This loss doesn't require enumeration of cycles as well as doesn't suffer from the problem of multiplication of small numbers. But on the other hand loss will be "backpropagated" only to the wrongly cut edges where the original loss penalties all edges within the cycle for which constraint has been violated.

6.3 Edge Weight Embedding

There are a lot of GNN architectures (e.g. Graph Attention Network Velickovic et al. (2017)) that work with edge attributes. But attributes should be high dimensional (since multiplication of weight by some shared parameter doesn't change anything). The idea of edge weight embedding is to split weights into bins (ranges) and then assign for every bin learnable embedding (similar to assigning embedding to token in NLP).

The important point in this regard is that negative and positive values always should belong to different bins. The figure 3 shows the idea of the method.

This idea is the most controversial and there is no even hypothetical clue that it can improve the performance of the model.

6.4 Unsupervised formulation

The original formulation of the problem is supervised which requires having an labeled dataset for training. But in their nature minimum cost multicut problem is an unsupervised problem. But the only thing which make the current formulation supervised is a binary cross-entropy loss (even orthogonal loss (6) doesn't depend on optimal solution).

We can try replace $\mathcal{L}_{BCE}$ loss with loss which depends on the cost of cut (1) itself. For example, let β_t be a decreasing sequence, then multicut cost loss can be as follows:

$$\mathcal{L}^t_{MC}(\hat{y}) = \beta_t \left(c(\hat{y}) - c_{LB} \right),$$

where $c_{LB} = \sum_{w_i < 0} w_i$ – a lower bound of the cost.

It is supposed that on the first steps of training, when β is high, our model will learn to assign the minimum possible cost and then, when β become small we will take care of the cycle consistency (2) and other losses.

Moreover, from some point of view this loss is more natural for the given multicut problem since classical cross-entropy doesn't distinguish between incorrect labeling of the edge with small and big absolute value, whereas the impact on the cost (which we actually try to minimize) is not equal for different weight's values. And the proposed formulation considers these different impacts.

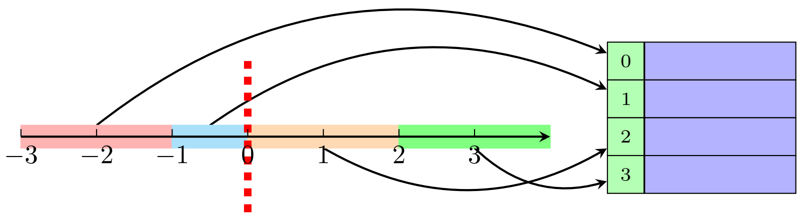

Figure 3 Illustration of assigning embeddings to bins of weight's value

6.5 Self-prior learning / finetuning

Learning on large datasets allow model generalize based on data instead of overfit, but with the generalization model can defeat on particular examples. But our task is to assign the best possible costs for all problems and initially we don't need generalization property.

Unsupervised formulation allow us to perform self-prior (inspired by Hanocka et al. (2020)) learning: overfit different, possibly graph depended model (e.g. vary number of layers w.r.t. the size of graph) on every particular graph which can allow model to output the best result for this specific instance of minimum cost multicut problem.

In this of course we have to retrain model for every graph and it requires time (which is the main advantages of the GNN pipeline). It is not just an simple inference as it was in the previous cases, but it can be relaxed if the "training" process will start from already pretrained network (overfitting by finetuning).

References

Bjoern Andres, Jörg H Kappes, Thorsten Beier, Ullrich Köthe, and Fred A Hamprecht. Probabilistic image segmentation with closedness constraints. In *2011 International Conference on Computer Vision*, pages 2611–2618. IEEE, 2011.

Bjoern Andres, Thorben Kroeger, Kevin L Briggman, Winfried Denk, Natalya Korogod, Graham Knott, Ullrich Koethe, and Fred A Hamprecht. Globally optimal closed-surface segmentation for connectomics. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part III 12*, pages 778–791. Springer, 2012.

Thorsten Beier, Constantin Pape, Nasim Rahaman, Timo Prange, Stuart Berg, Davi D Bock, Albert Cardona, Graham W Knott, Stephen M Plaza, Louis K Scheffer, et al. Multicut brings automated neurite segmentation closer to human performance. *Nature methods*, 14(2): 101–102, 2017.

Long Chen, Martin Strauch, and Dorit Merhof. Instance segmentation of biomedical images with an object-aware embedding learned with local constraints. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part I*, pages 451–459. Springer, 2019.

Sunil Chopra and Mendu R Rao. The partition problem. *Mathematical programming*, 59(1-3):87–115, 1993.

Tyler Derr, Yao Ma, and Jiliang Tang. Signed graph convolutional networks. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 929–934. IEEE, 2018.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2mesh: A self-prior for deformable meshes. *ACM Trans. Graph.*, 39(4), 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392415. URL https://doi.org/10.1145/3386569.3392415.

Steffen Jung and Margret Keuper. Learning to solve minimum cost multicuts efficiently using edge-weighted graph convolutional neural networks. *arXiv preprint arXiv:2204.01366*, 2022.

Jörg H Kappes, Bjoern Andres, Fred A Hamprecht, Christoph Schnörr, Sebastian Nowozin, Dhruv Batra, Sungwoong Kim, Bernhard X Kausler, Thorben Kröger, Jan Lellmann, et al. A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, 115(2):155–184, 2015.

Margret Keuper, Evgeny Levinkov, Nicolas Bonneel, Guillaume Lavoué, Thomas Brox, and Bjorn Andres. Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proceedings of the IEEE international conference on computer vision*, pages 1751–1759, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423. IEEE, 2001.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.