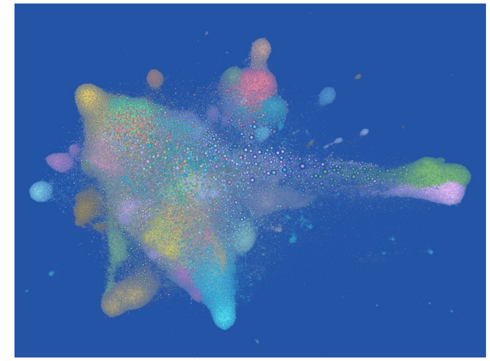
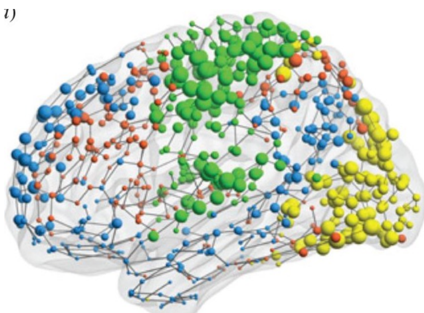
Alignment-free Node Embedding for EDA Congestion Prediction





Mark Coates

Department of Electrical and Computer Engineering

McGill University

Networks Research Lab

networks.ece.mcgill.ca

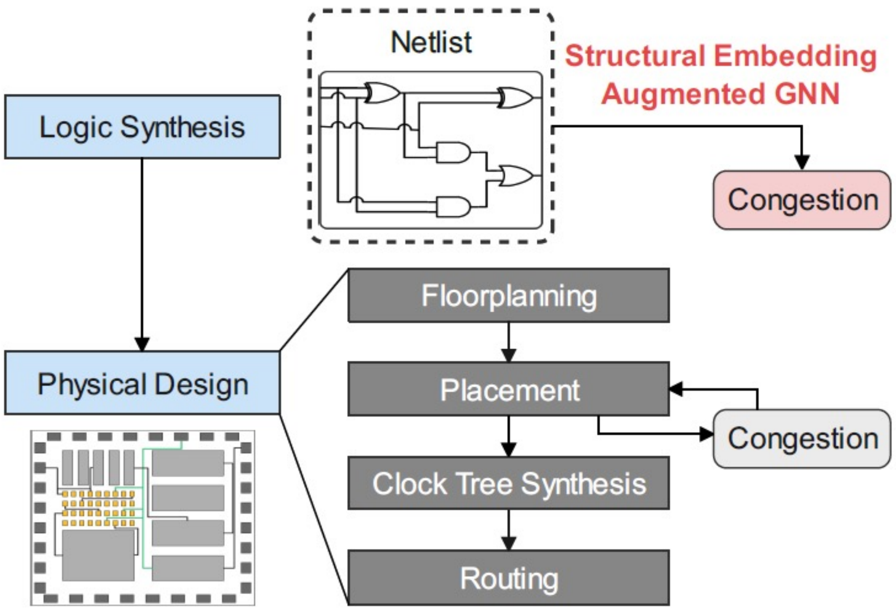
Research Sponsored by NSERC, MITACS, Department of National Defence, Huawei

Collaborators: Amur Ghose, Yingxue Zhang (Huawei)

Figures: Vertés et al. 2014; Axel Bruns / QUT Digital Media Research Centre

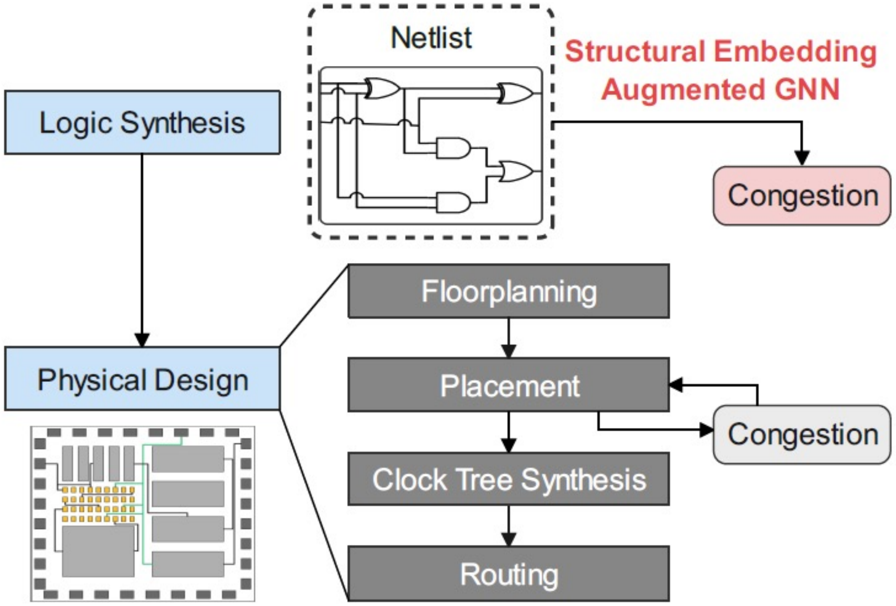
Electronic Design Automation (EDA) Workflow

- Register Transfer Level (RTL) design: VDHL/ Verilog
- models a synchronous digital circuit in terms of
  - flow of signals between hardware registers
  - logical operations performed on signals
- Convert to physical layout through logic synthesis & physical design



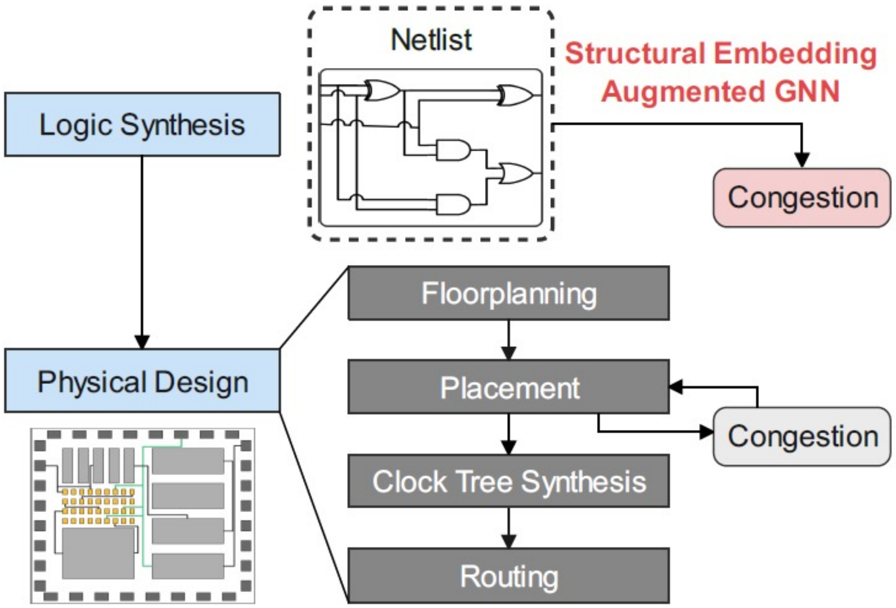
Electronic Design Automation (EDA) Workflow

- Logic synthesis
  - Convert to a netlist: contains interconnection information of all circuit elements
  - Cells: groups of transistors & interconnects that provide a Boolean logic function
- Physical design
  - All circuit elements placed on circuit boards & connected by wires



Routing Congestion

- Routing congestion: important metric that reflects the quality of the chip design
- Most EDA tools: congestion predicted AFTER cell placement
- Used as a feedback signal to optimize placement solution

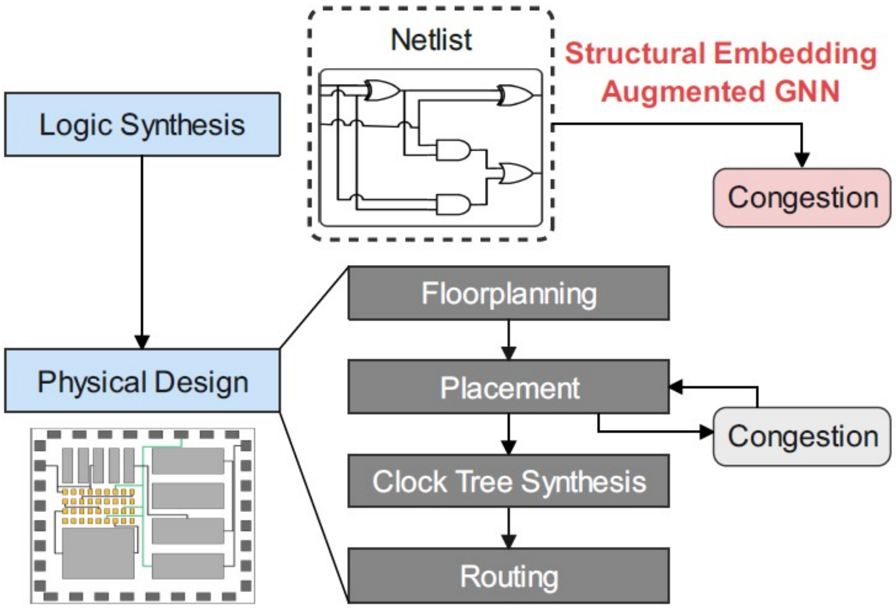


Routing Congestion

- Routing congestion: important metric that reflects the quality of the chip design.
- Most EDA tools: congestion predicted AFTER cell placement
- Used as a feedback signal to optimize placement solution

Problems:

- Large scale circuits - placement iteration is computationally expensive
- Some congestion caused by poor logic structures cannot be fixed by placement

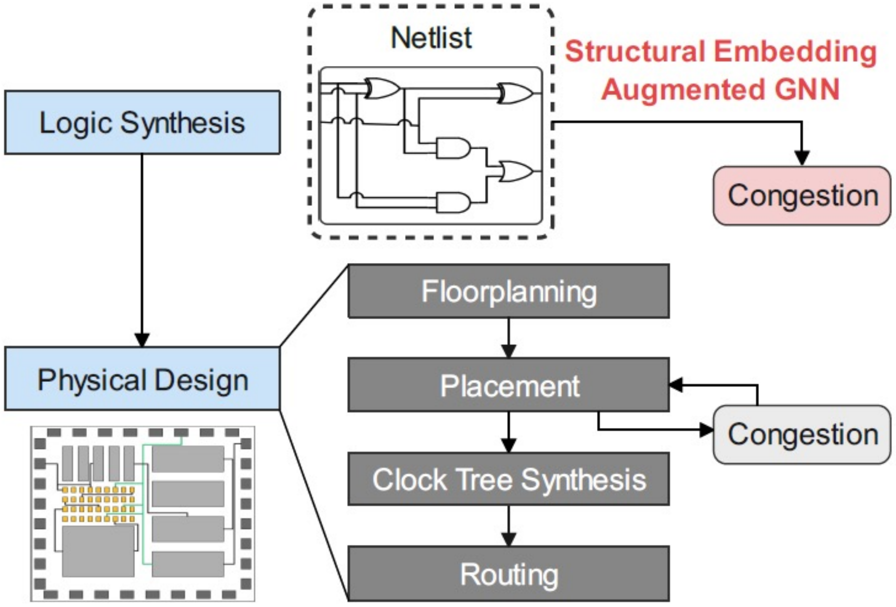


Routing Congestion

- Routing congestion: important metric that reflects the quality of the chip design.

- Goal: Estimate logic-induced congestion at logic synthesis stage
- Provide quick feedback and shorten design cycles.

- Map to node regression task
  - Train on one set of netlists (graphs)
  - Predict on another set of netlists



Background – Prior Work

- Candidate metrics to identify network structures with high potential of routing congestion:
  - size of the local neighborhood
  - adhesion of a logic network
  - Groups of Tangled Logic (GTL) metric

M. Saeedi et al., "Prediction and reduction of routing congestion," in *Proc. Int. Symp. Physical Design*, 2006.

T. Lin and C. Chu, "Polar 2.0: An effective routability-driven placer," in *Proc. Design Automation Conf.*, 2014

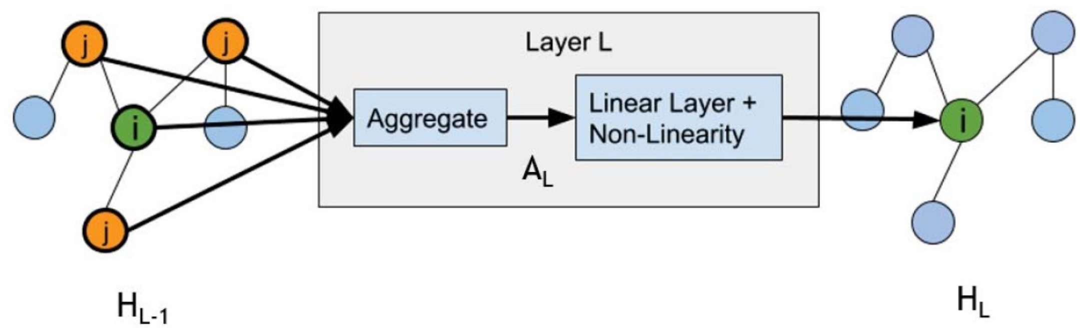
P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Proc. Design, Automation & Test in Europe Conf. , 2007*

T. Jindal et al., "Detecting Tangled Logic Structures in VLSI Netlists," in *Proc. Design Automation Conf., 2010.*

P. Kudva et al., "Metrics for structural logic synthesis," in Proc. *Int. Conf. Computer Aided Design, 2002.*

Background – CongestionNet

- 8 layer Graph Attention Network
- Features:
  - Trainable embedding of length 50 for each cell type
  - Cell's logic description
  - Pin count
  - Cell size



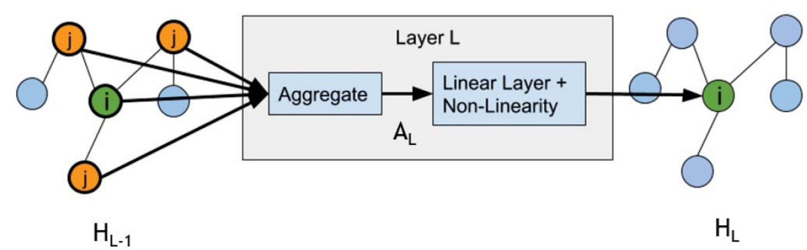
Source: R. Kirby, etc. (2019)

R. Kirby et al. ,"CongestionNet: Routing congestion prediction using deep graph neural networks," in *Proc. IEEE Int. Conf. Very Large Scale Integration, 2019.*

Background – CongestionNet

Limitations of CongestionNet:

- informative cell features (cell type, pin count and cell size) are not available at the early logic synthesis stage.
- Requires a large training dataset (over 50 million cells).
- Deep GAT exhibits oversmoothing

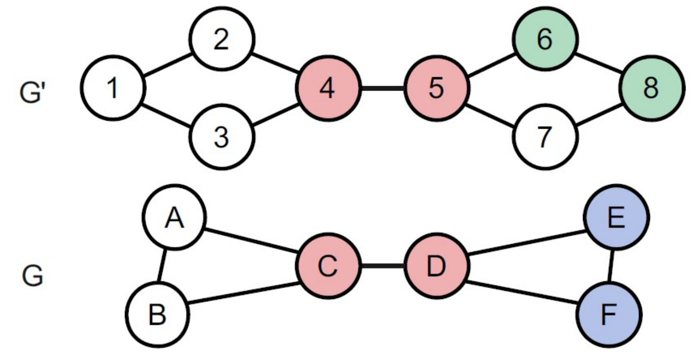


Source: R. Kirby, etc. (2019)

R. Kirby et al. ,"CongestionNet: Routing congestion prediction using deep graph neural networks," in *Proc. IEEE Int. Conf. Very Large Scale Integration, 2019.*

Embedding Methods: Proximity

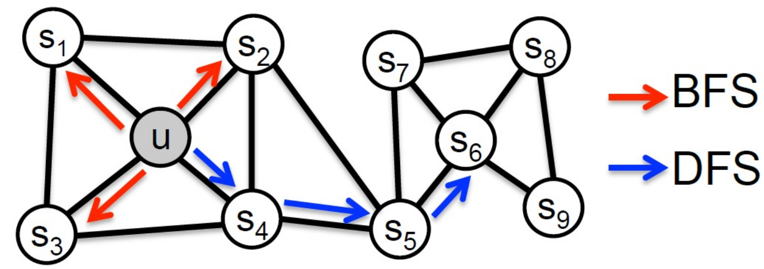
- Relates to the distance between two vertices.
- Neighboring vertices have most similar embeddings.
- Embeddings learned on one graph cannot be directly used in another distinct graph.
- Random-walk based methods like node2vec, LINE and DeepWalk.



Proximity-based (blue and green) similarity across two disjoint graphs.

A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *KDD* 2016.
J. Tang et al., "LINE: Large-scale information network embedding," *WWW* 2015.
B. Perozzi et al., "Deepwalk: Online learning of social representations," KDD 2014.

Example: node2vec



Source: A. Grover (2016)

- Maximize log-probability of observing a network neighbourhood $N_S(u)$ for a node u conditioned on its feature representation f(u)

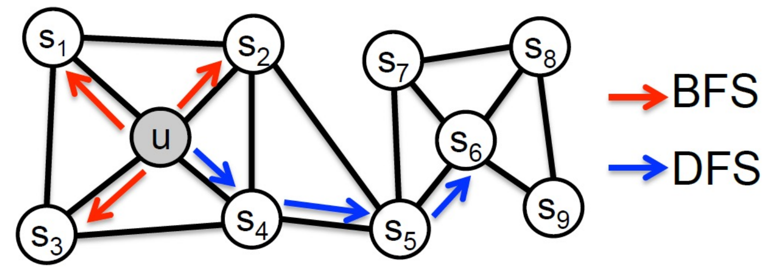
$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u))$$

- Assumptions: conditional independence $Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u))$

- Conditional likelihood is softmax of dot-product

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}$$

Example: node2vec



Source: A. Grover (2016)

- Problem becomes:

$$\max_f \sum_{u \in V} \left[-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]$$

- Expensive to compute the partition function $Z_u = \sum_{v \in V} \exp(f(u) \cdot f(v))$

- Can use stochastic gradient ascent, but still slow for million-node graphs

- Sample the neighborhood: biased random walk

Example: DeepWalk

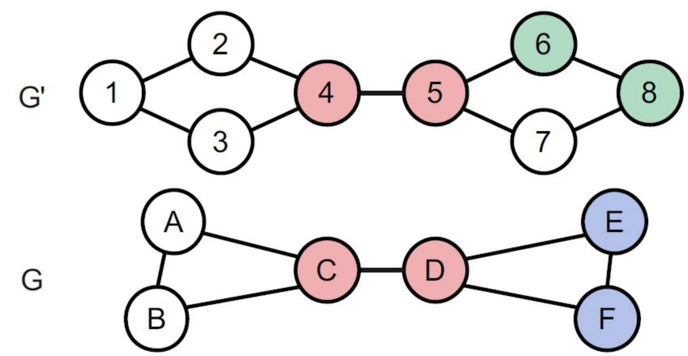
- Problem becomes:

$$\max_f \sum_{(u,v)\in\mathcal{D}} \log Pr(v|u) \qquad\qquad Pr(v|u) = \frac{\exp(f(v)\cdot f(u))}{\sum_{v'\in V}\exp(f(v')\cdot f(u))}$$

- Set D derived by:
  - co-occurrences in windows of length T hops in a set of length L random walks (γ walks starting at each node)

Embedding Methods: Structural similarity

- Relates to properties of a node such as its degree or spectral properties,
- Two nodes can be structurally similar even if they belong to two different graphs.
- GraphWAVE, Role2vec, and struct2vec



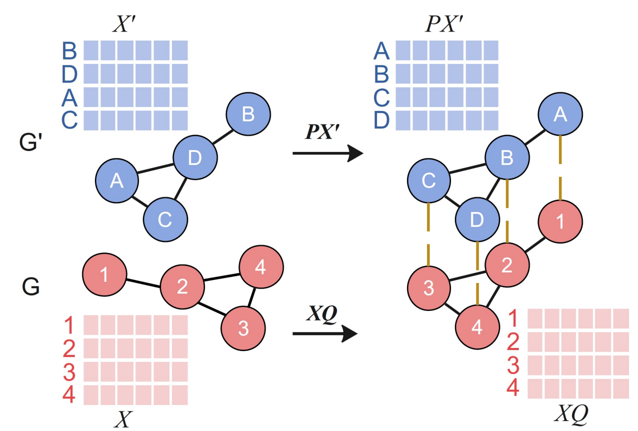
Structural (red) similarity across two disjoint graphs.

4, 5 and C, D are similar even though they belong to two different graphs.

Embedding Alignment

- Consider two embeddings X, X' obtained using a proximity embedding method on two graphs G, G'
- Wasserstein-Procrustes graph alignment:
  - identify orthogonal matrix Q and a permutation matrix P to minimize the distance between XQ and PX'

$$\underset{\boldsymbol{P} \in P_n, \boldsymbol{Q} \in O_n}{\arg\min} \boldsymbol{P}, \boldsymbol{Q} \; ||\boldsymbol{XQ} - \boldsymbol{PX'}||^2$$



This becomes more difficult to understand when X, X' are of different sizes

Matrix Factorization

Pointwise Mutual Information (PMI) Matrices

- X: $|V| \times d$ embedding matrix representing the d-dimensional embeddings for all $v \in |V|$
- Similarity metric between two nodes i, j can be measured as $\langle X_i, X_j \rangle$.
- Similarity pattern for all pairs (i, j) of nodes is fully captured in the matrix XX^T

J. Qiu et al., "Network embedding as matrix factorization: Unifying Deepwalk, LINE, PTE, and node2vec," *WSDM 2018.*

Matrix Factorization

Pointwise Mutual Information (PMI) Matrices

- X: $|V|\times d$ embedding matrix representing the d-dimensional embeddings for all $v \in |V|$
- Similarity metric between two nodes i, j can be measured as $\langle X_i, X_j \rangle$.
- Similarity pattern for all pairs (i, j) of nodes is fully captured in the matrix XX^T
- XX^T is the PMI matrix
- Node embeddings can be directly obtained by factoring the PMI matrix.
- After conducting an eigendecomposition of the PMI matrix, we have USU^T and use $US^{1/2}$ as the embeddings.

Obtaining the PMI matrix: Infinite Walk

For DeepWalk, as the number of walks γ at each node approaches infinity and the walk length L approaches infinity, the PMI matrix approaches

$$\mathbf{M}_{\mathrm{T}} = \log\left(v_G\left(\frac{1}{T}\sum_{k=1}^{T}\mathbf{P}^k\right)\mathbf{D}^{-1}\right)$$

- v_G : volume of the graph – sum of degrees of all vertices
- $P = D^{-1}A$: random walk transition matrix

J. Qiu et al., “Network embedding as matrix factorization: Unifying Deepwalk, LINE, PTE, and node2vec,” *WSDM 2018.*

Infinite Walk

If we let T approach infinity too:

$$\mathbf{M}_\infty = \mathbf{1}\mathbf{1}^\top + \tilde{\mathbf{D}}^{-1/2}\left(\tilde{\mathbf{L}}^+ - \mathbf{I}\right)\tilde{\mathbf{D}}^{-1/2}$$

- Here $\widetilde{\boldsymbol{D}} = \frac{\boldsymbol{D}}{v_G}$ and $\tilde{\boldsymbol{L}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$ is the normalized Laplacian

S. Chanpuriya and C. Musco, “Infinitewalk: Deep network embeddings as Laplacian embeddings with a nonlinearity,” KDD 2020.

Matrix Factorization

Pseudo-inverse is annoying. Let's directly use the Laplacian instead.

$$\mathbf{M}'_P = \mathbf{1}\mathbf{1}^T + \mathrm{Tr}(\mathbf{D}_P)\mathbf{D}_P^{-1/2}\mathbf{L}_P\mathbf{D}_P^{-1/2}$$

$$\mathbf{M}''_P = \mathbf{1}\mathbf{1}^\top + \frac{\mathbf{M}'_P}{C}$$

Clamp $\mathbf{M}''_P$ to range [L,H] with L>0 and set $\mathbf{M}''_P \leftarrow \log \mathbf{M}''_P$

L & H: numerical stability; C controls the extent to which a node influences its neighbour

S. Chanpuriya and C. Musco, "Infinitewalk: Deep network embeddings as Laplacian embeddings with a nonlinearity," KDD 2020.

Datasets

Dataset: We extract two publicly available netlist datasets:

(1) Superblue circuit lines from DAC 2012 which we place via DREAMPLACE

(2) A collection of circuits provided with the OPENROAD framework.

Graph Generation:

- Netlist data of each design is converted to a graph
- This represents the circuit elements as nodes and their interconnections as edges.

Datasets

Dataset: We extract two publicly available netlist datasets:

(1) Superblue circuit lines from DAC 2012 which we place via DREAMPLACE

(2) A collection of circuits provided with the OPENROAD framework.

Circuit name	**Nodes**	**Terminals**	**Nets**
Train set			
Superblue2	1014029	92756	990899
Superblue3	919911	86541	898001
Superblue6	1014209	95116	1006629
Superblue7	1364958	93071	1340418
Superblue9	846678	57614	833808
Superblue11	954686	94915	935731
Superblue14	634555	66715	619815

Datasets

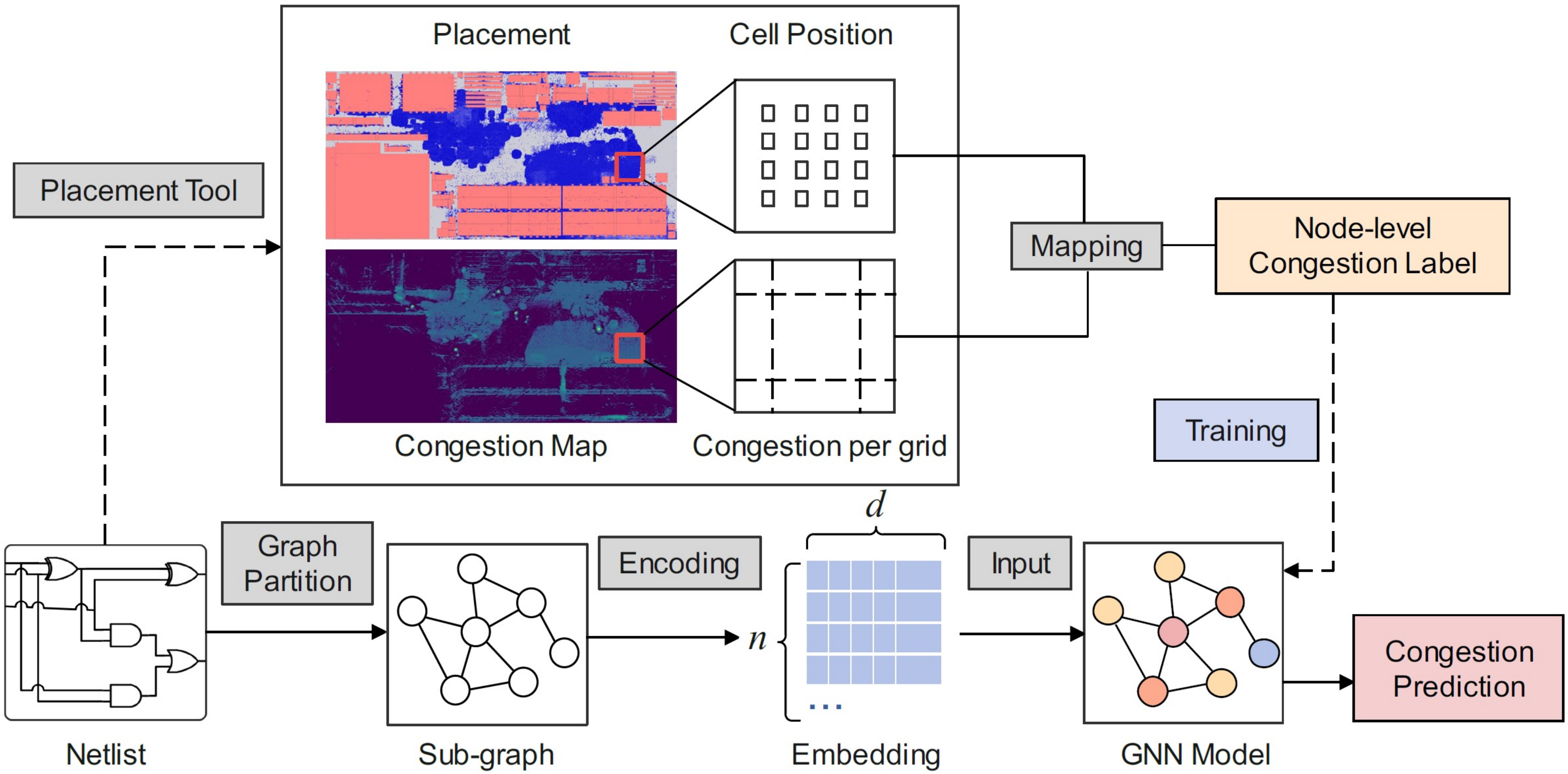
Congestion Maps:

- Generated by NCTUGR for Superblue circuit lines
- Used built-in FastRoute for OPENROAD dataset.

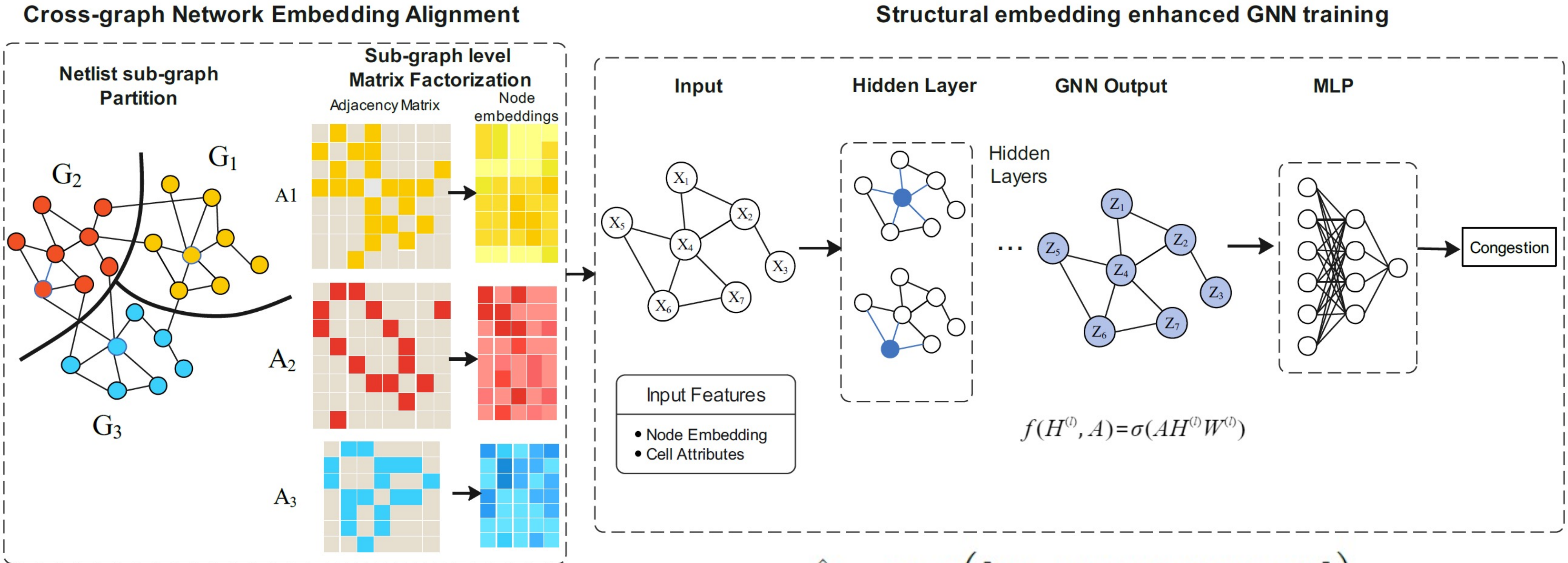
Label Generation:

- Board is partitioned into grids
- Congestion value for each grid cell = wiring demand divided by routing capacity.
- Congestion value for each grid cell is assigned to all nodes located in the grid cell
- This is used as ground-truth labels for the training.

Method – Training and inference



Schematic of our GNN method



$$\hat{Y} = \mathrm{MLP}\Big([[\boldsymbol{X};\boldsymbol{E}];\mathrm{SAGE}([\boldsymbol{X};\boldsymbol{E}])]\Big),$$

GraphSAGE: 2 hidden layers of size 200 and 160 + MLP with 2 hidden layers

Metrics and baselines

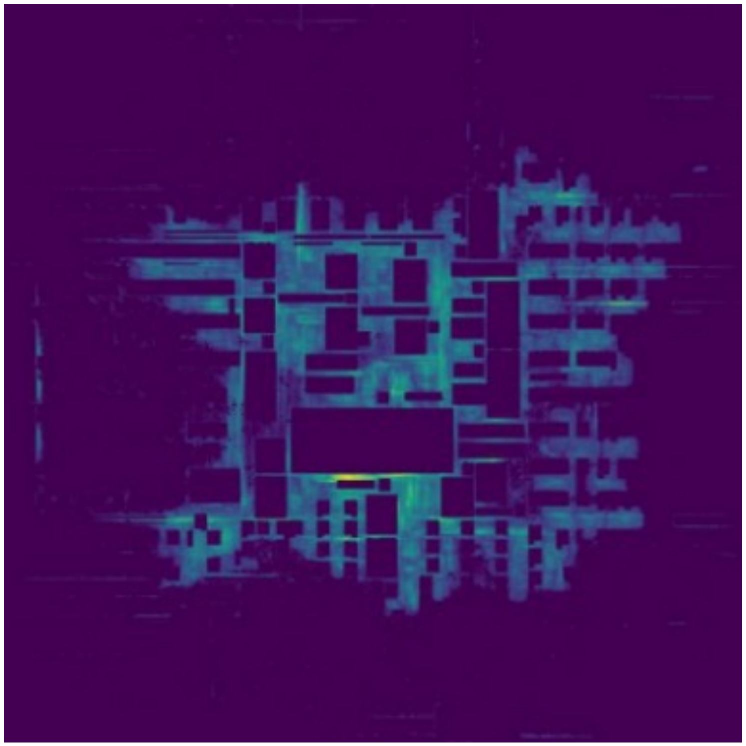
Metrics

- Pearson correlation coefficient (PCC)
- Spearman correlation
- Kendall correlation

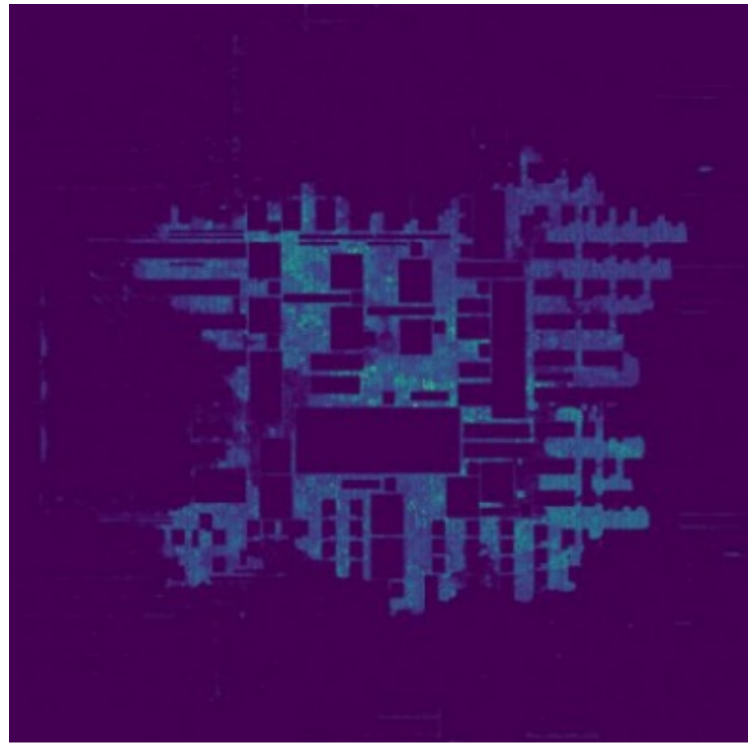
Baselines

- Neighborhood size: the number of nodes, reachable from v, within distance k.
- Adhesion: maximum min-cut between v and other nodes in the neighborhood.
- Groups of tangled logic (GTL):
  - Measure that examines the structure of the graph cut around node v.
  - Reflects how interconnected the local clusters of the cell are
- CongestionNet

Results – Prediction



Ground truth congestion map.



Predicted congestion map

Results – Prediction Accuracy

Methods	Lower level congestion					
	Pearson		Spearman		Kendall	
	Node	Grid	Node	Grid	Node	Grid
Adhesion metric	0.09	0.16	0.06	0.20	0.06	0.14
Neighbourhood metric	0.02	0.04	0.18	0.27	0.13	0.18
GTL metric	0.02	0.01	0.14	0.23	0.10	0.16
CongestionNet	0.26	0.35	0.27	0.33	0.19	0.24
Embedding-enhanced GNN (ours)	0.31	0.43	0.34	0.44	0.25	0.31

Results – Runtime

RUNTIME COMPARISON (SECONDS) BETWEEN SUBGRAPH-LEVEL TRAINING AND BLOCK SAMPLING

Architecture runtime comparisons (per graph per epoch)				
	Superblue		OPENROAD	
GNN architecture	Training time	Inference time	Training time	Inference time
No partitioning (ours + block sampler)	56.2	65.4	9.8	14.1
With partitioning				
Our architecture	2.2	6.1	0.22	2.5
CongestionNet	6.4	8.7	0.78	3.8

- Partitioning the graph leads to significant improvements in runtime
- The inference time is reduced by up to 30% on both datasets compared to CongestionNet.

Results – Runtime

MATRIX FACTORIZATION RUNTIME VS OTHER EMBEDDING METHODS IN SECONDS

Embedding runtime comparisons				
Embedding method	Train time	Alignment time	Train time	Alignment time
Node2vec	250.6	1750.4	45.2	733.5
LINE	167.8	1355.2	19.7	802.1
DeepWalk	143.7	1566.5	22.1	783.4
Ours	80.4	-	18.2	-

- Matrix factorization embedding method saves up to 90% runtime compared to classic methods plus explicit alignment

Key Findings

- Proximity-based node-level embedding methods require post-processing (alignment) to be used for cross-graph prediction.
- Matrix-factorization based embedding learning combined with subgraph level training
  - faster, more effective, and can generalize to unseen graphs.
- Concatenating cell structural embeddings with cell attributes directly improves performance.
- No informative cell attributes → meaningful prediction from netlist embeddings alone.
- Instead of deep GATs, wide and shallow SAGE-GNNs achieve superior performance.