Graph Methods for Particle Tracking

JEREMY FERGUSON, UNIVERSITY OF CALIFORNIA-BERKELEY

MENTOR: DANIEL MURNANE, LAWRENCE BERKELEY NATIONAL LABORATORY

JUNE 28, 2021

Background

- ATLAS – studying particle interactions and collisions
- Particle collisions need track reconstruction
- Current methods scale quadratically
- Need faster methods to complement current methods with new high-luminosity era LHC
- Neural networks can accomplish this

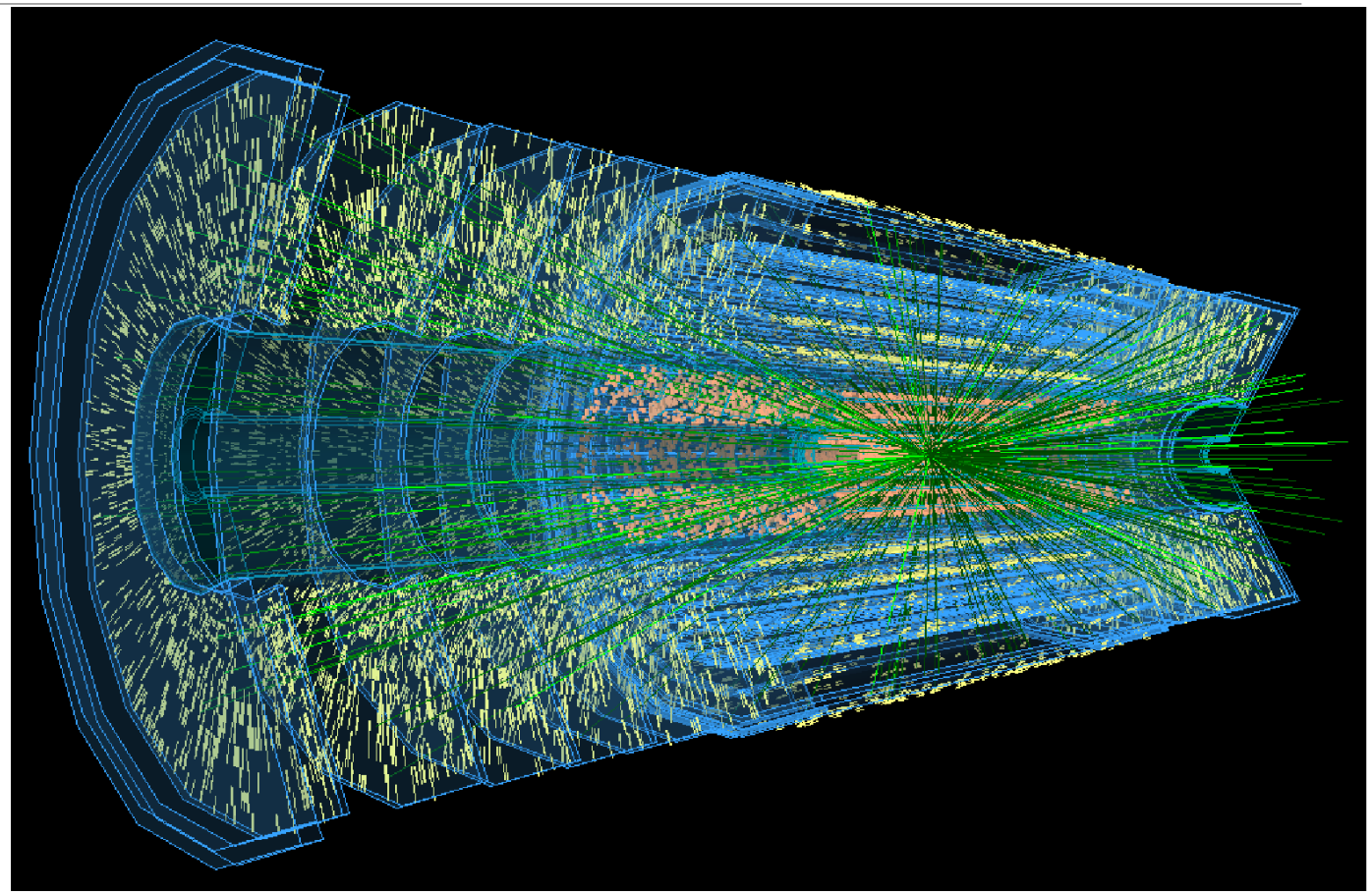


Image from: GNNs and Metric Learning for Particle Physics, Daniel Murnane

Exatrkx project

- Overall Exatrkx project goal – develop methods for identifying particle tracks using machine learning
- Use GNN – Graph Neural Network – pipeline for achieving this
  - Graphs capture sparsity of data
  - GNN's are growing in popularity, lots of algorithms and models already exist

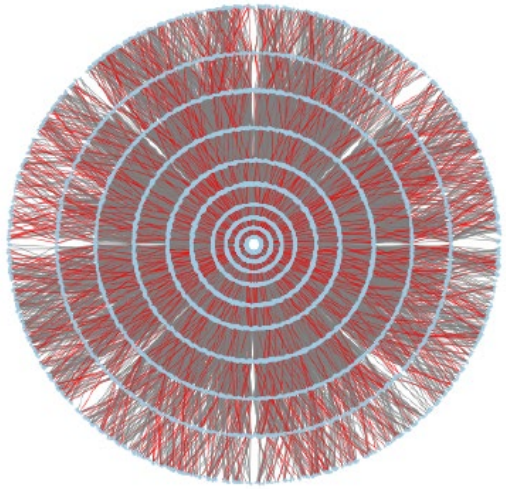


Image from: GNNs and Metric Learning for Particle Physics, Daniel Murnane

Graphs

- Graph is a collection of:
- Nodes - Hits recorded by detector as particles pass through
- Edges - should exist between hits that belong to the same track/particle and not exist between other nodes

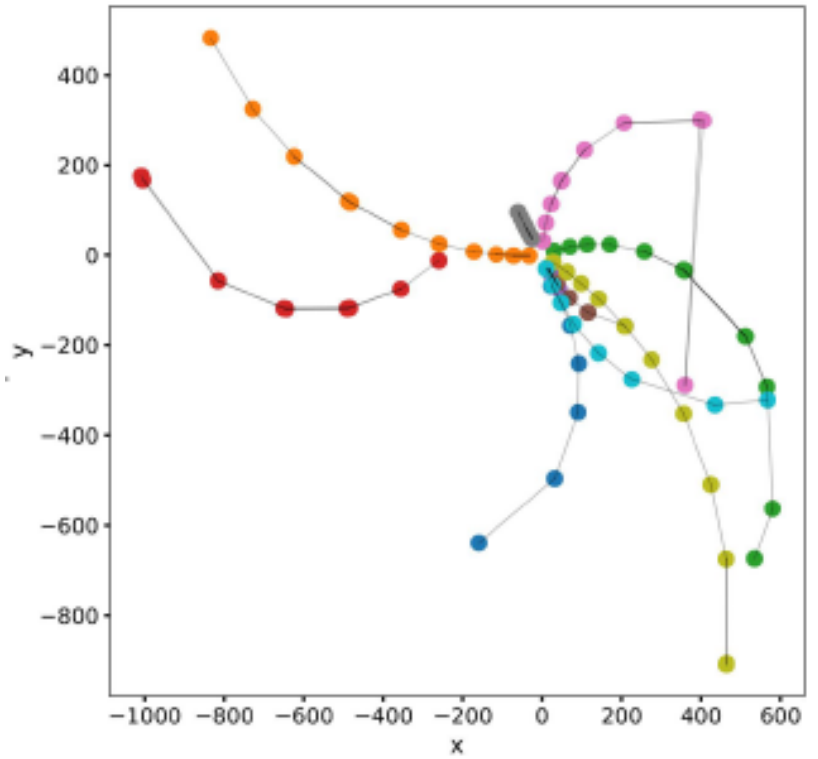


Image from: GNNs and Metric Learning for Particle Physics, Daniel Murnane

TrackML Dataset

- 9000 events
- 100,000 hits from around 10,000 particles
- Contains duplicate hits and noise

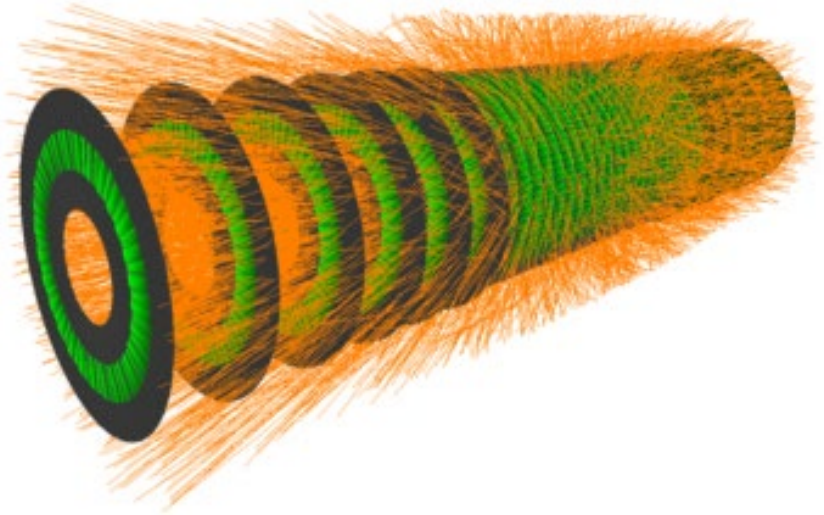
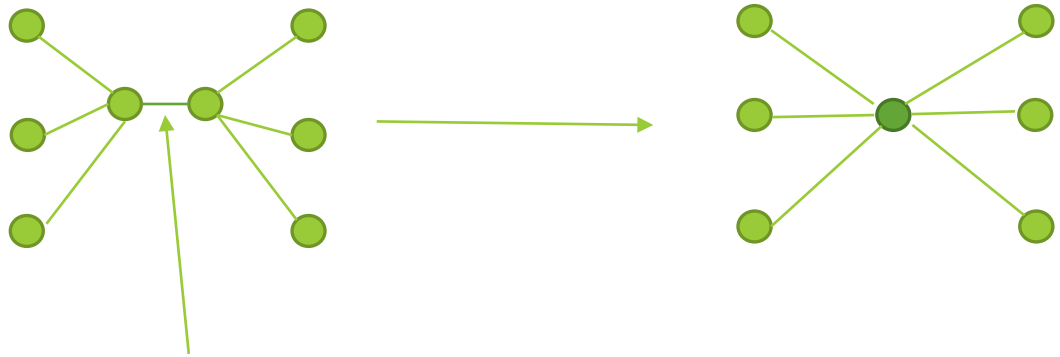


Image from: GNNs and Metric Learning for Particle Physics, Daniel Murnane

Fellowship Objective

- Improve graph segmentation for track parameter regression
- Original track parameter regression- learn to predict each nodes track parameter
- Proposed track parameter regression – cluster all hits from each track into a single node and predict track parameters from that
- Requires clustering algorithm – ended up as main goal of fellowship

Clustering algorithm



Contract this edge

- Combines nodes and removes edges
- Challenge is to train a neural network to cluster the correct edges

Clustering algorithms

- Many proposed implementations
- Existing open-source implementation in Pytorch Geometric library:
  - EdgePool class
  - Based off of paper - “Towards Graph Pooling by Edge Contraction” https://graphreason.github.io/papers/17.pdf
- New vectorized implementation – different algorithm that is meant to produce similar results as Pytorch Geometric implementation

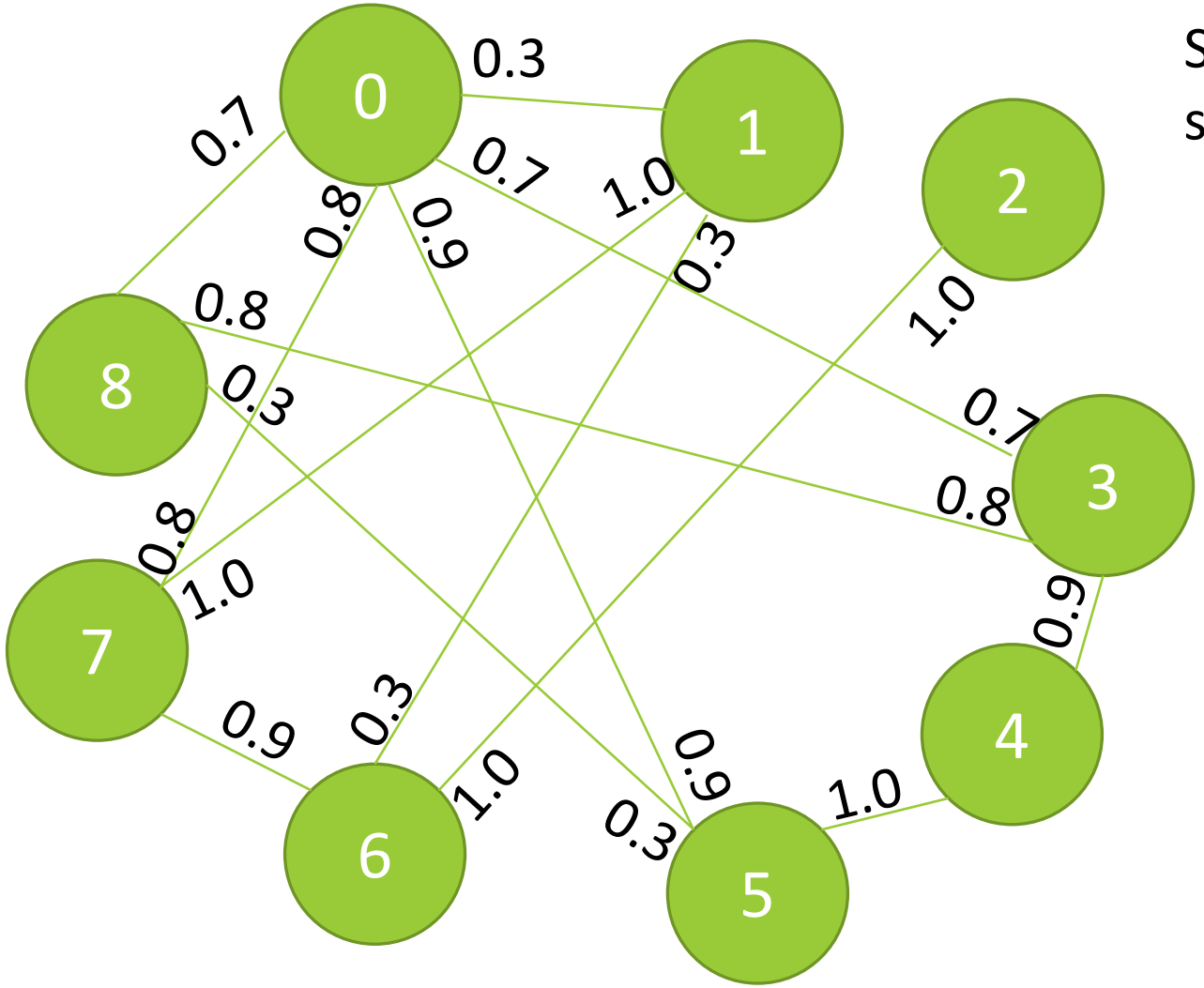
Pytorch Geometric edge pooling

- Score edges using neural network layer
- Sort edges by descending score
- Iterate over list and contract highest-scoring edges until all nodes have been contracted
- Achieves ~50% contraction rate per iteration

New vectorized implementation

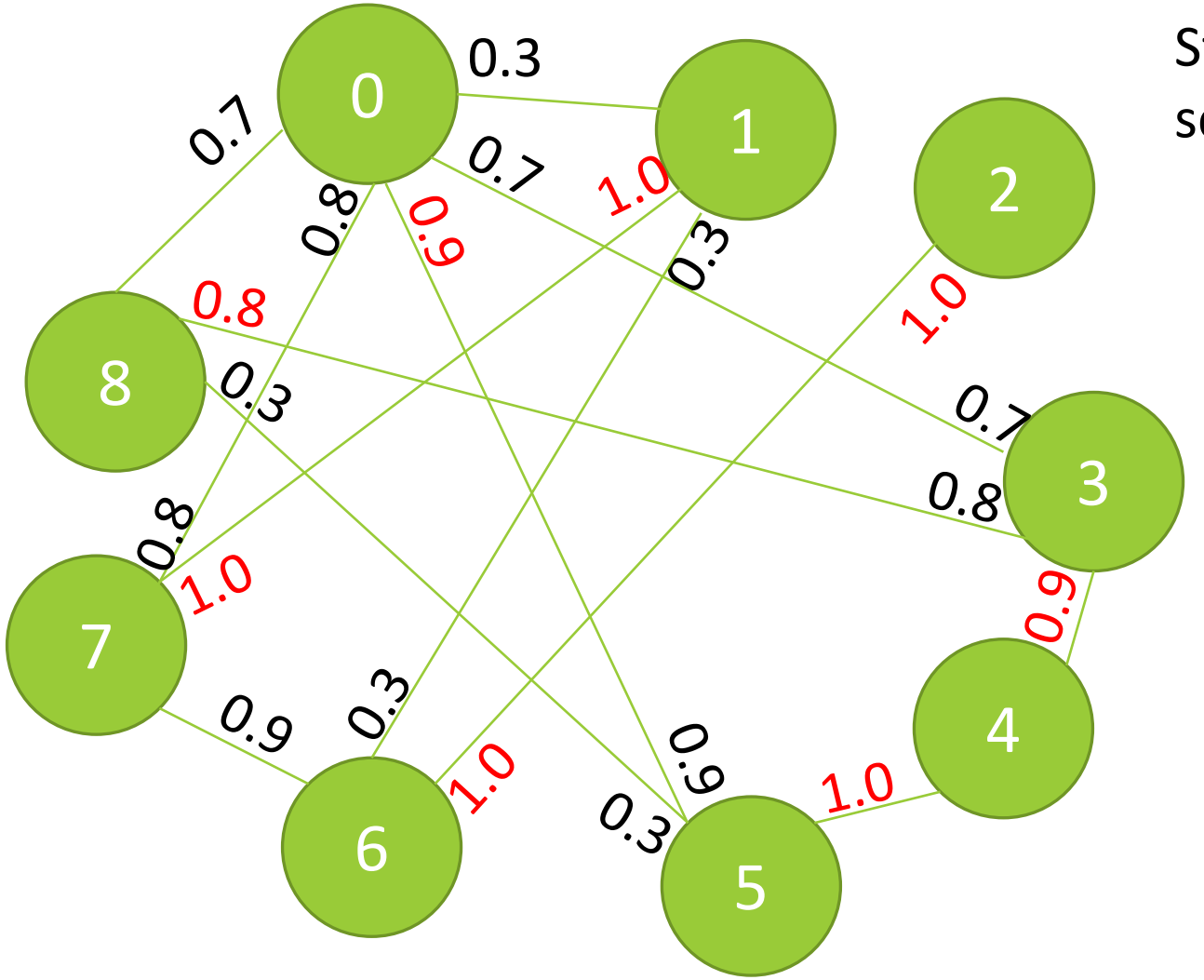
- Score edges using a neural network
- Goal is to find optimal pairing of nodes based on edge scores
- Use Pytorch and CUDA functionality to gain speedup
- Algorithm must be changed due to vectorization

Clustering particle example



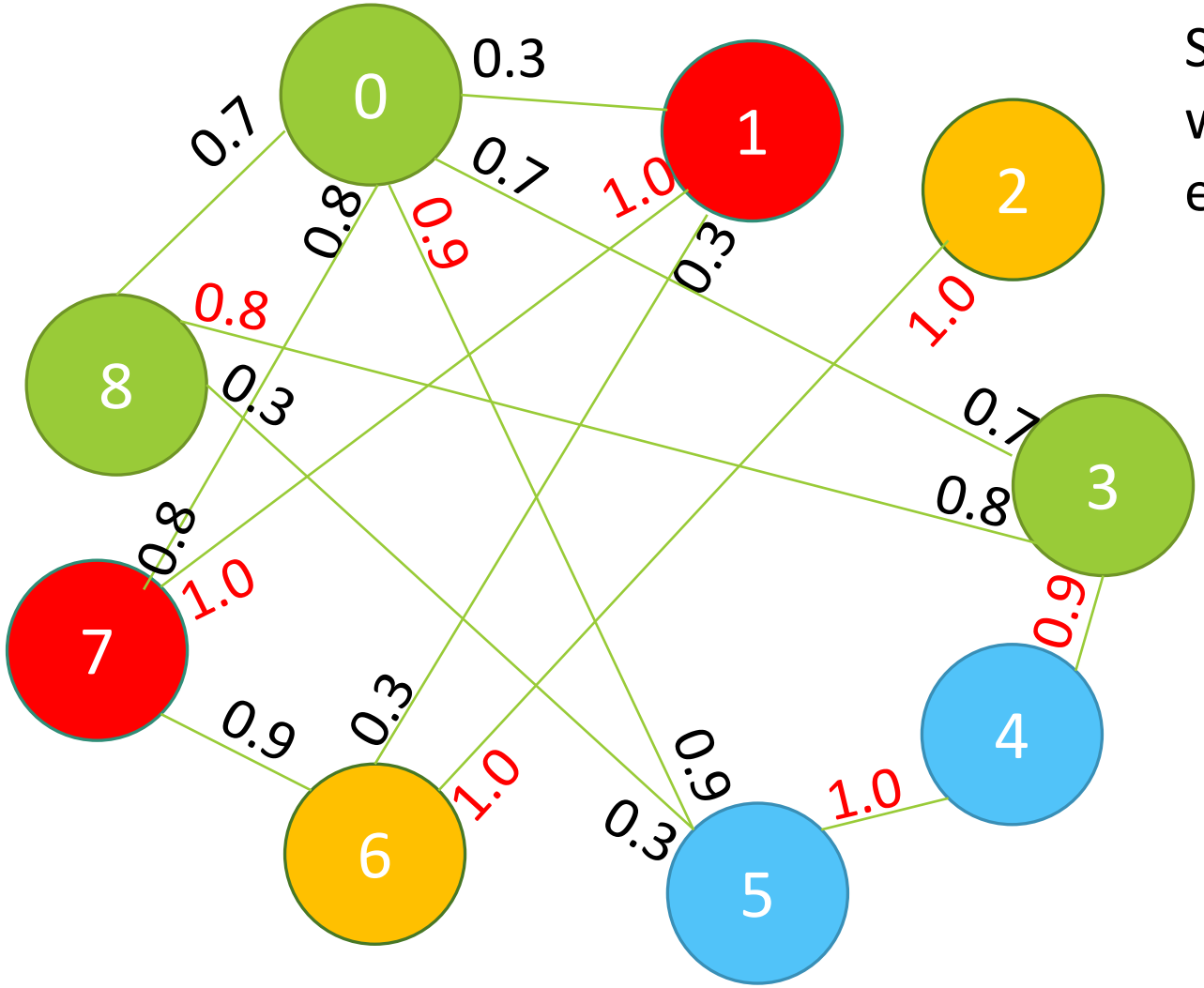
Step 1: Generate edge scores between 0 and 1

Clustering particle example



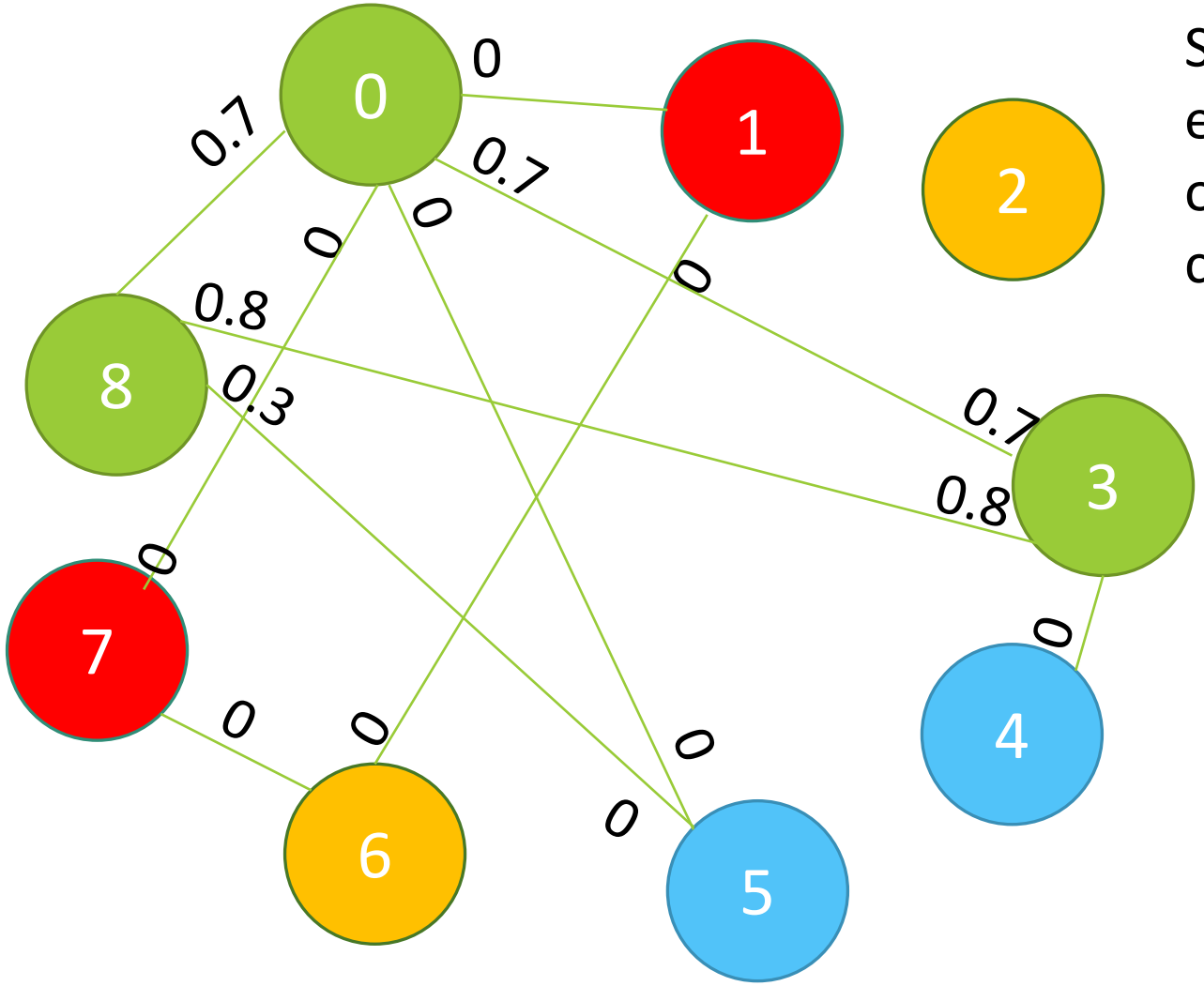
Step 2: Identify highest scoring edge for each node

Clustering particle example



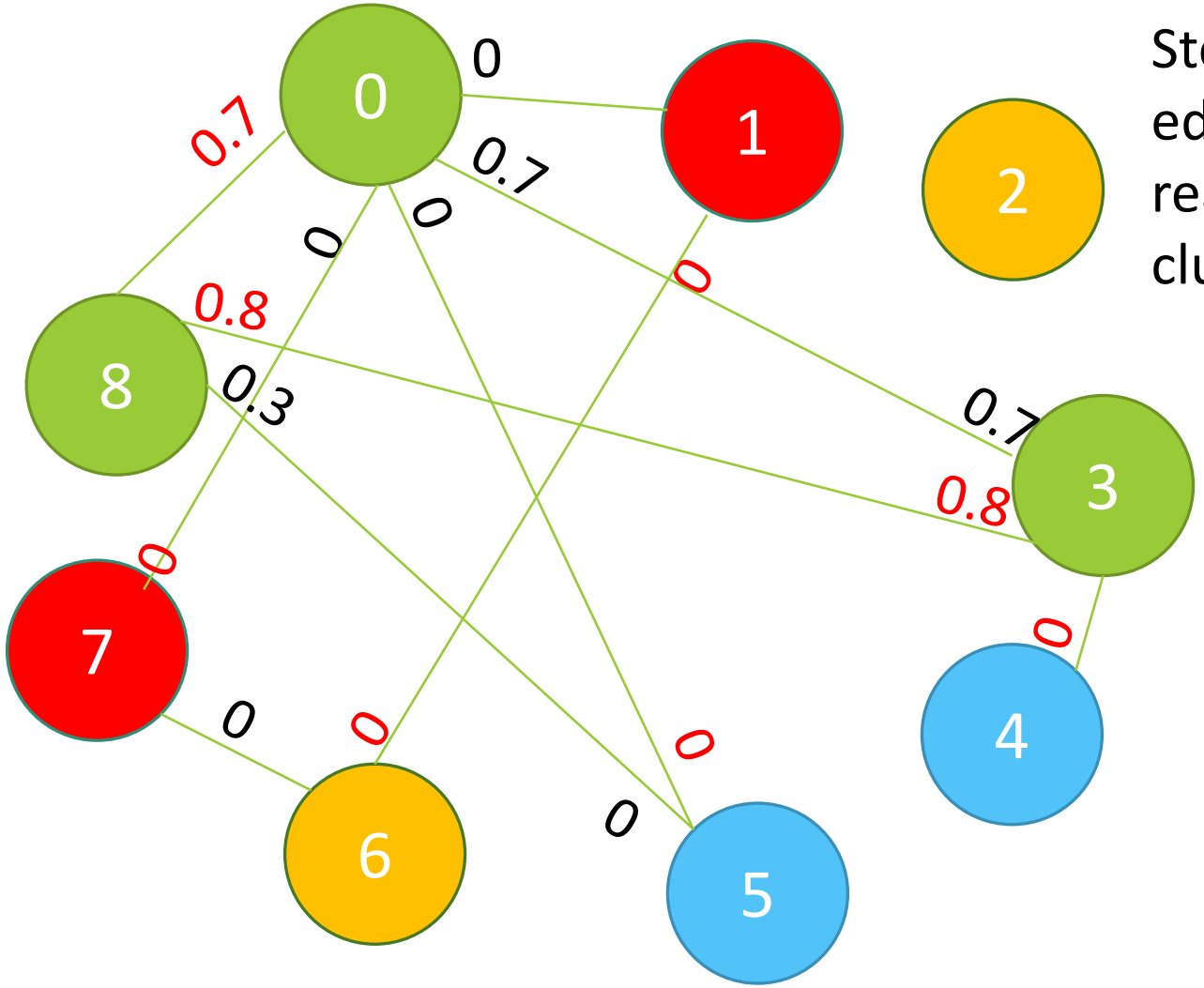
Step 3: Identify pairs of nodes which share a highest scoring edge

Clustering particle example



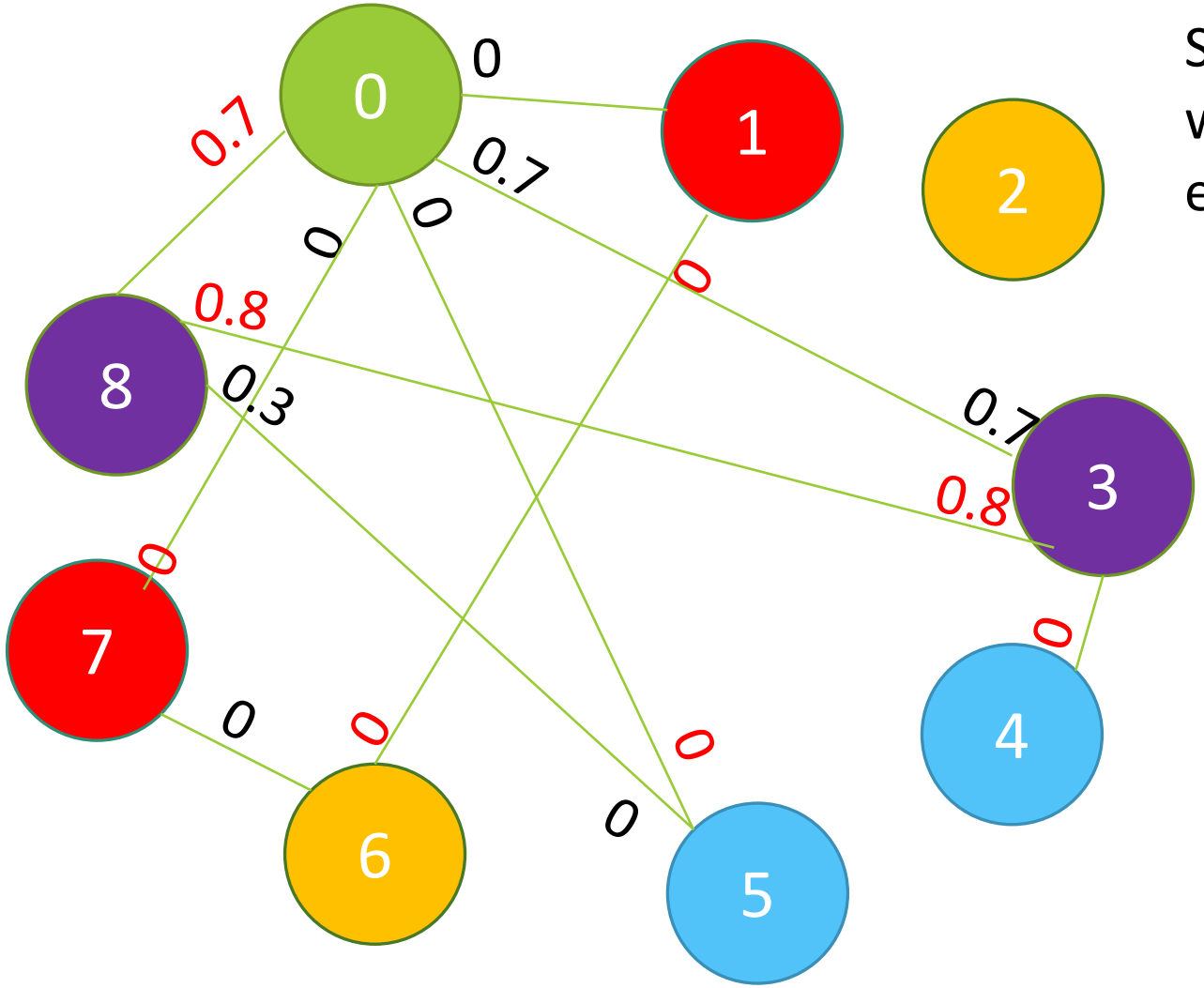
Step 4: Remove clustered edges, and temporarily zero out edges adjacent to clustered nodes

Clustering particle example



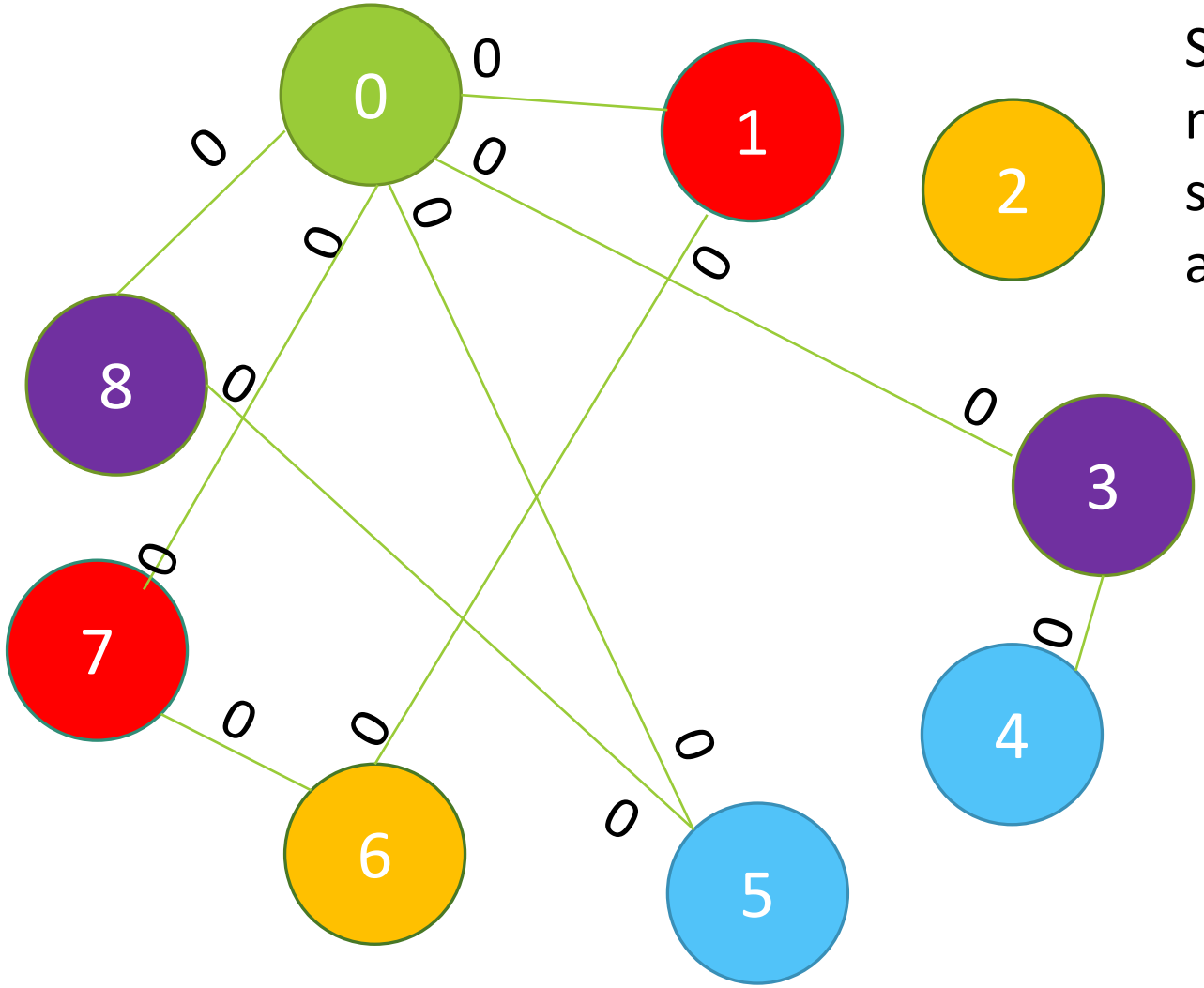
Step 5: Find highest scoring edges for all nodes (but we really only care about un-clustered nodes)

Clustering particle example



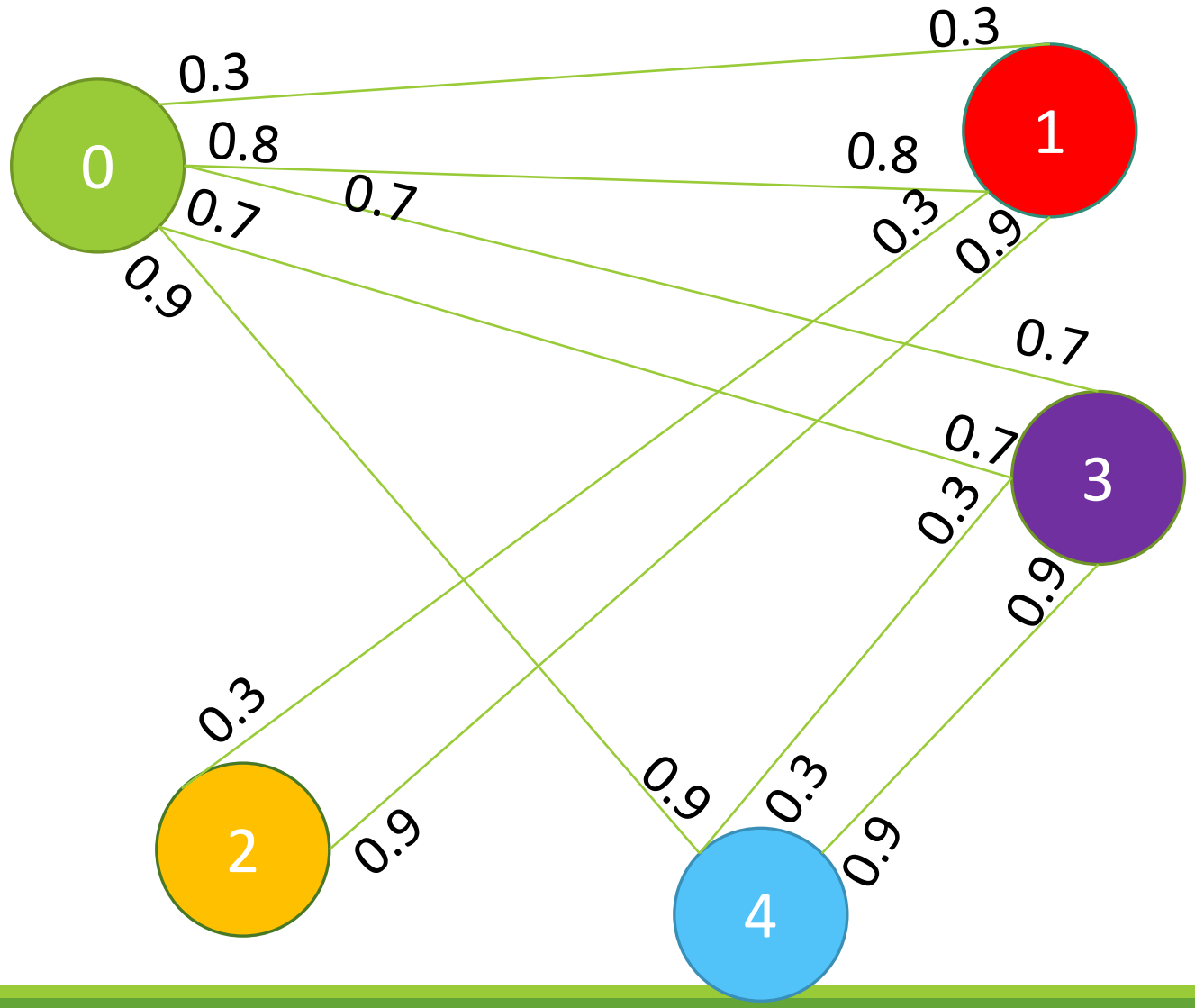
Step 6: Find pairs of nodes which share a highest scoring edge

Clustering particle example



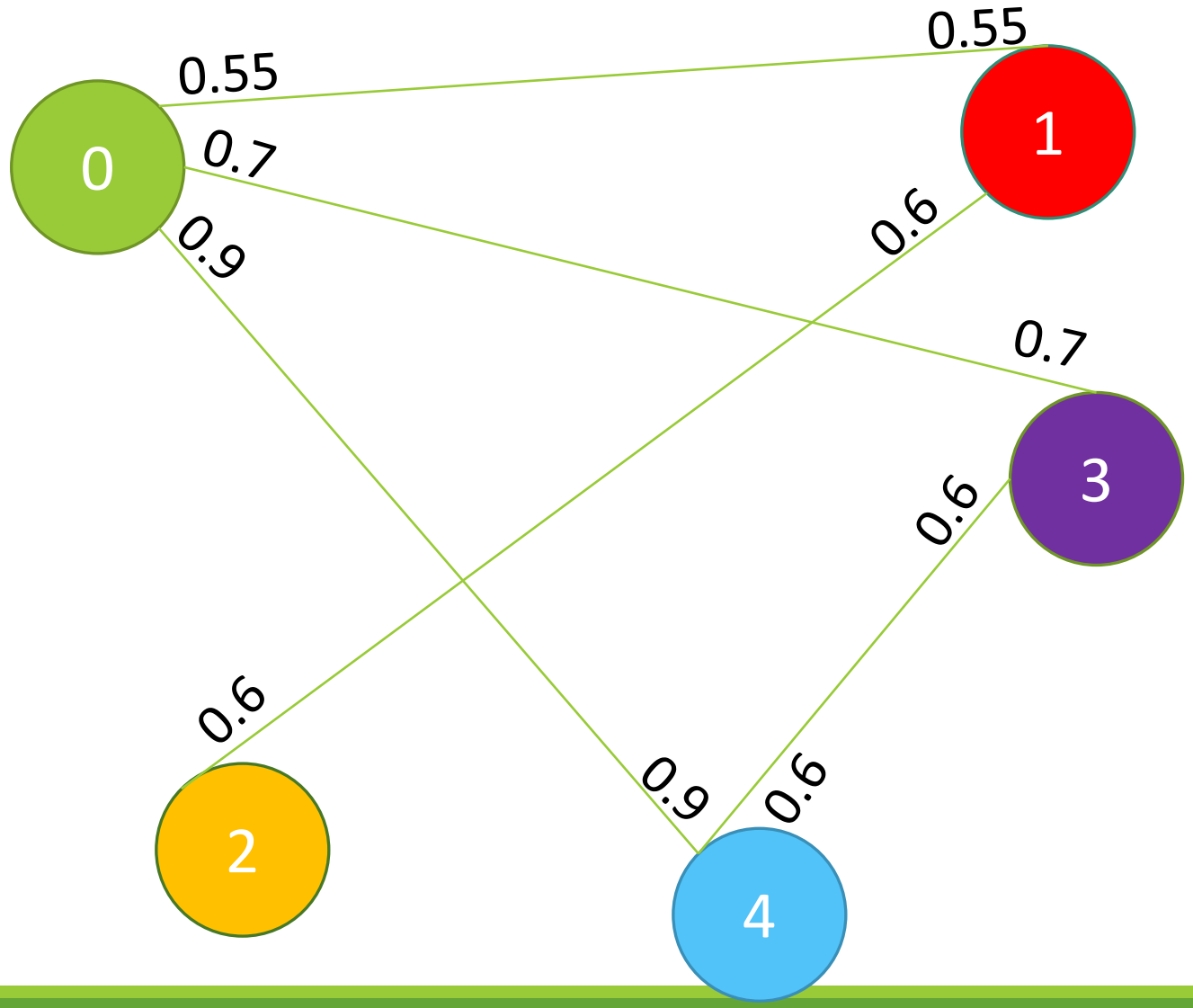
Step 7: Contract the pairs of nodes that share a highest scoring edge, and zero out adjacent edges temporarily.

Clustering particle example



Step 8: Construct a new graph with the clustered nodes.

Clustering particle example



Step 8: Remove duplicate edges and take the mean of the edge scores

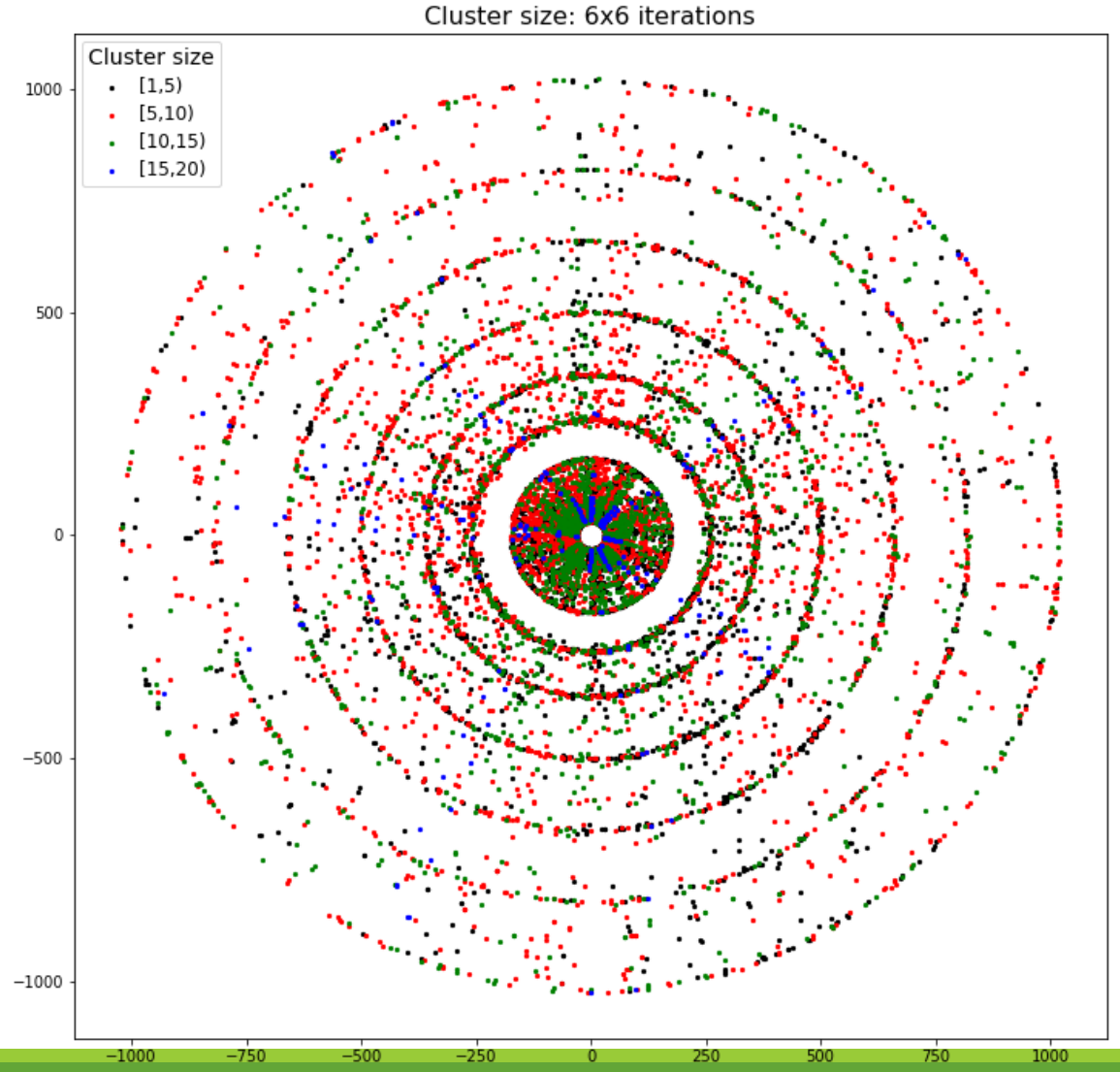
Application to Particle Tracking

- Applied to GNN pipeline with TrackML dataset
- Performed several rounds of contraction, as well as track parameter regression on the clustered nodes
- Achieved 60% efficiency, 60% purity
  - Efficiency – correct edges contracted / all edges that should be contracted
  - Purity – Correct edges contracted / all edges contracted
- Able to achieve clustering ratio of 10%

Clustering algorithm

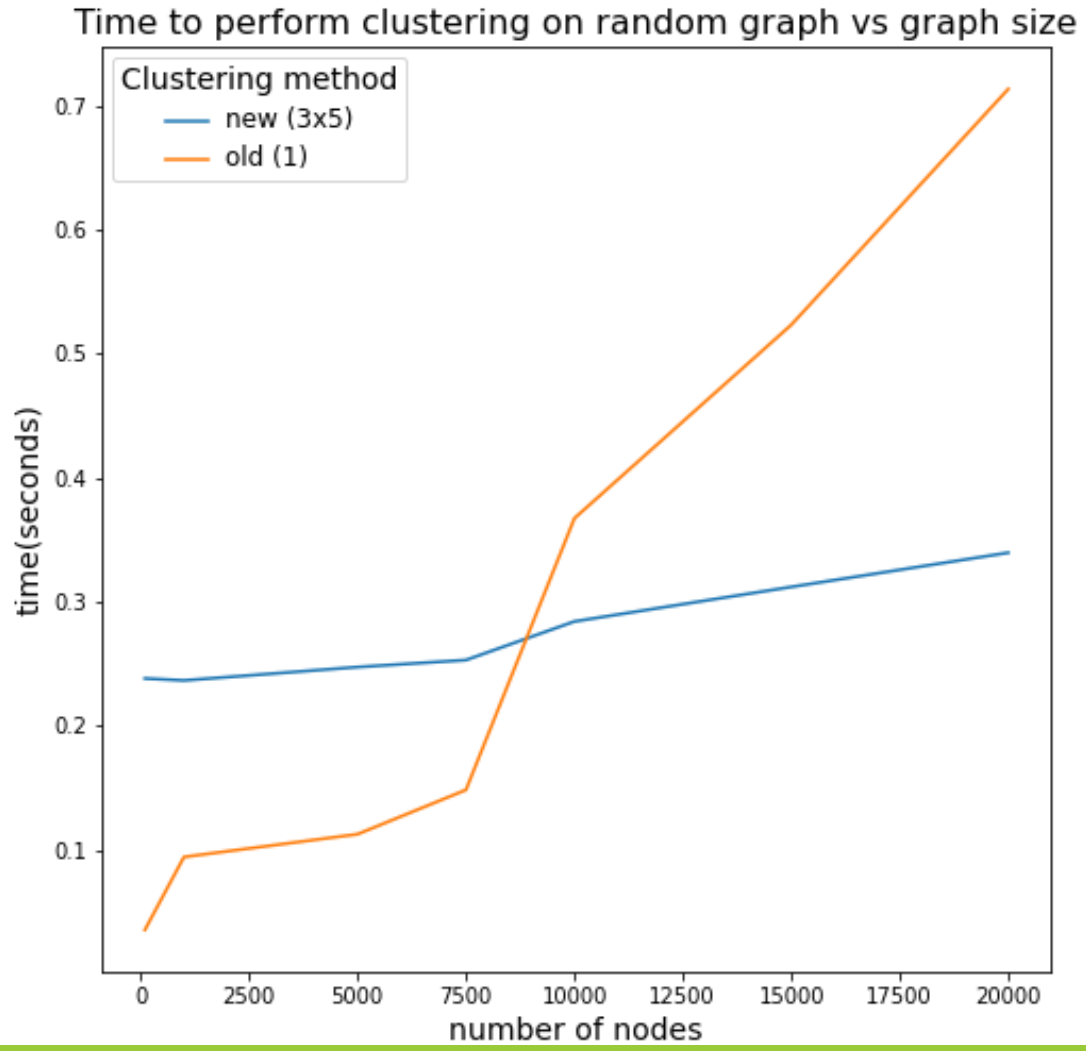
- Can be performed multiple times – leads to 2 nested loops
  - Inner loop-algorithm performed in example
  - Outer loop-performing the inner loop over multiple rounds of clustering
  - Notation in plots: Outer loop x inner loop iterations

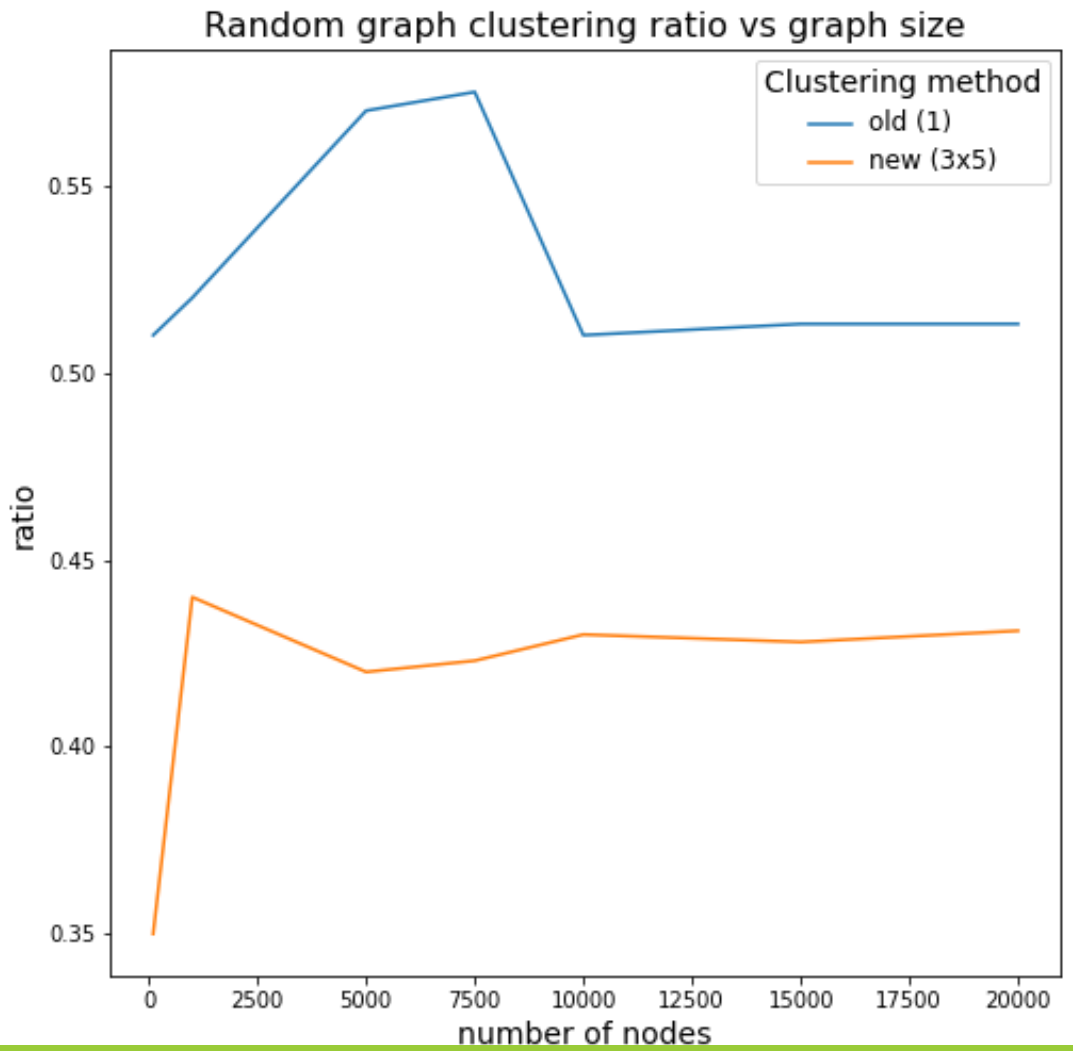
Example clustering results (TrackML dataset)



- Most tracks have 10-20 hits
- Green and blue would be ideal range in this case
- Red and black are smaller clusters that are likely not capturing full track

Comparison between implementations





Summary

- Vectorized clustering algorithm performs well compared to existing implementation on time
  - Specific time multiplier may vary based on inner and outer loop iterations, but growth rate is much better
  - Previous plot was time to apply trained model, so includes the rest of the GNN calculations
- Future work needed:
  - Application to particle tracking, improve training to allow for more accurate clustering
  - Track parameter regression is the ultimate goal of this application
  - Add new vectorized implementation to Pytorch Geometric library

- Thanks to my mentor Daniel Murnane for his support