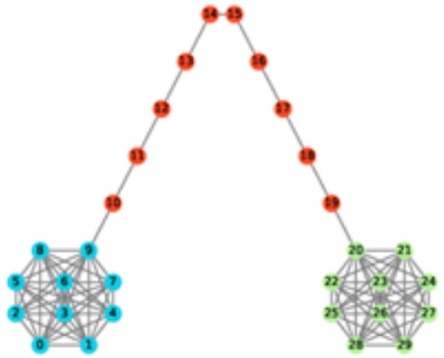
LightNE: A Lightweight Graph Processing System for Network Embedding

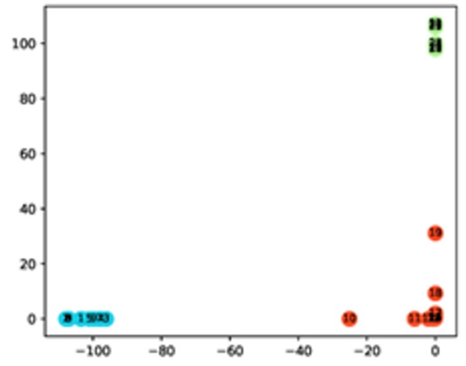
Jiezhong Qiu, Laxman Dhulipala, Jie Tang, Richard Peng, Chi Wang

Steven Raphael

What Is Network Embedding?

- Take a graph and turn every node into a low-dimensional vector.
- Embedding should approximately preserve graph structure.


Original Graph

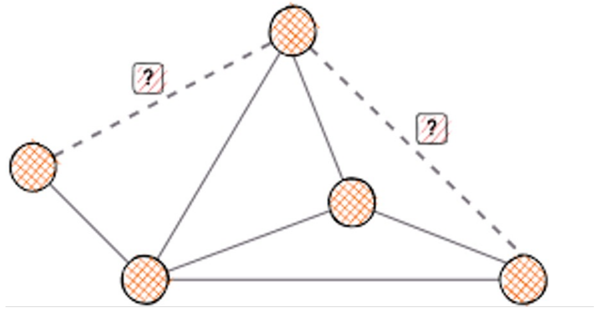


Network Embedding in 2D space

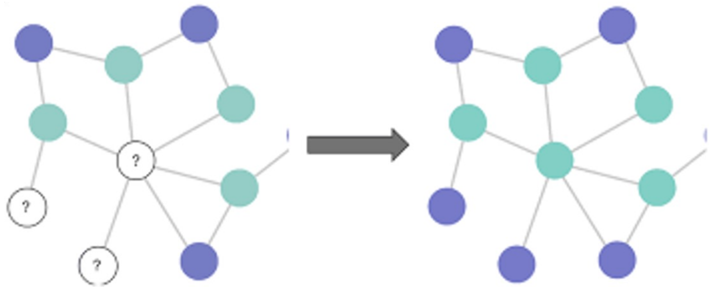
Benefits of Network Embedding

Certain tasks can be completed with regression or ML models trained on the embedding.

- Link prediction: determining whether two nodes are connected.



- Node Classification: determining the label of a node.



Goals of the Paper

The paper develops and implements a lightweight, scalable, accurate system for network embedding.

- Lightweight: process graphs with up to 100 billion edges using under $100 worth of hardware costs.
- Scalable: process a graph with a billion edges in 1.5 hours.
- Accurate: achieve higher accuracy than current state-of-the-art systems using similar time and budget.

Results Summary

Competing systems: NetSMF, GraphVite, PyTorch-BigGraph (PBG), ProNE+.

- The authors created ProNE+ by creating an optimized implementation of an existing system, ProNE.

LightNE achieves higher accuracy than NetSMF, GraphVite, and PBG while processing the graphs at least 8x faster.

LightNE can get slightly better accuracy and speed than ProNE+, and can get even better performance with more processing time.

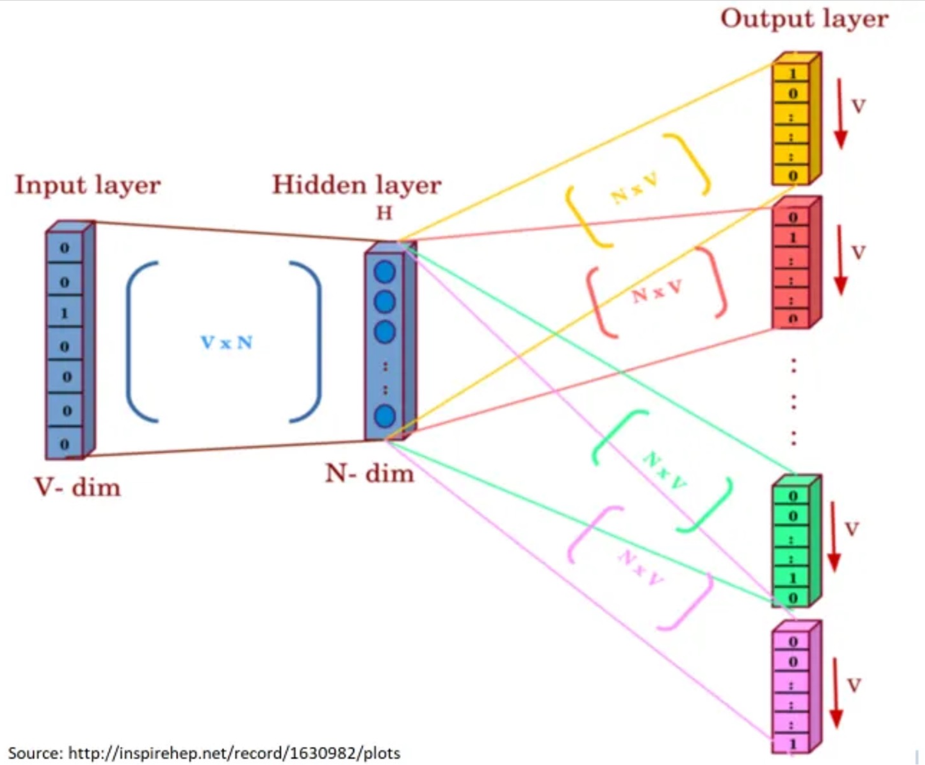
Background

- DeepWalk
- Matrix Factorization
- DeepWalk Matrix Representation (NetMF)

DeepWalk Algorithm for Network Embedding

- Sample a bunch of random walks.
- Skip-Gram algorithm: trains a model to predict the neighboring vertices (context windows) in a random walk given a center vertex.
- “Hidden layer” of model converts vertices to low-dimensional embeddings.
    - Extract embeddings from hidden layer of fully trained model.
- Training time optimization: Negative sampling.
    - Only consider a few randomly selected words during the update step, instead of the loss w.r.t. all words.

Skip-Gram Visualization



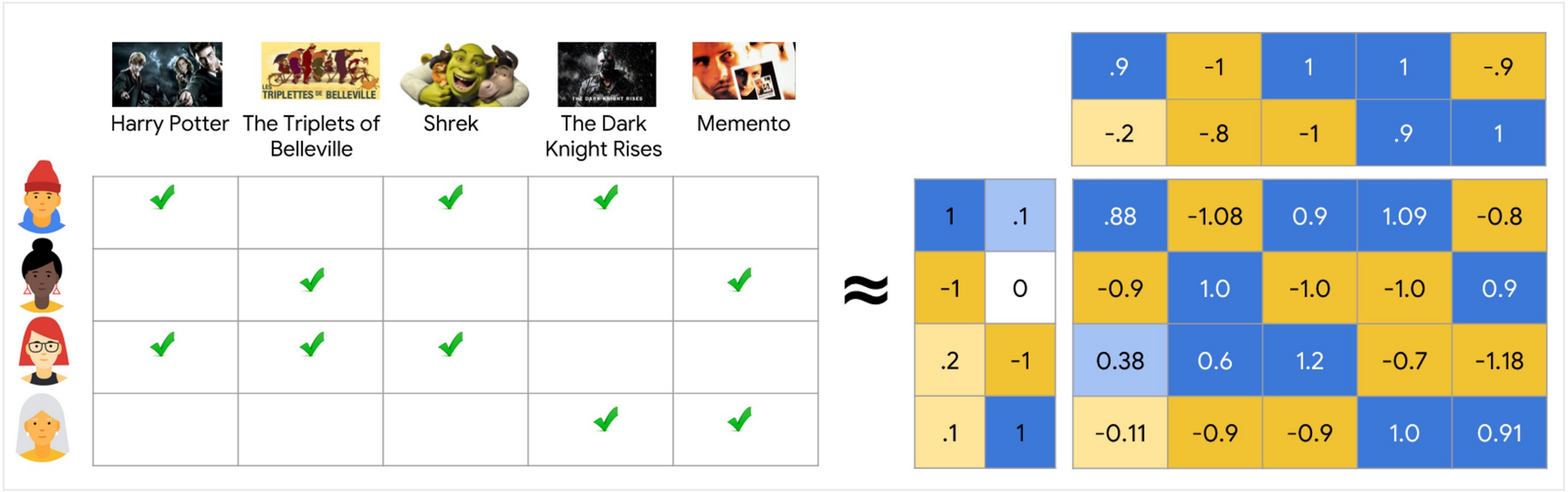
Source: http://inspirehep.net/record/1630982/plots

Matrix Factorization

- Many network embedding algorithms can be replicated by factoring a chosen matrix.
- A VxV matrix can be approximately factored into a Vxd and dxV matrix.
- Entries in the matrix are dot products of vector embeddings and context embeddings.
- Paper uses singular value decomposition (SVD) for factoring.

Matrix Factorization Example

The example constructs a 2-dimensional embedding.



Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
✓		✓	✓	
	✓			✓
✓	✓	✓		
			✓	✓

≈

1	.1
-1	0
.2	-1
.1	1

.9	-1	1	1	-.9
-.2	-.8	-1	.9	1

.88	-1.08	0.9	1.09	-0.8
-0.9	1.0	-1.0	-1.0	0.9
0.38	0.6	1.2	-0.7	-1.18
-0.11	-0.9	-0.9	1.0	0.91

DeepWalk Matrix Representation (NetMF)

- Goal: Create a matrix that represents the trained model.
- Given that a certain vertex v_n is in the context window of center vertex v in a random walk, we can explicitly find the probability that $v=v_i$ for each v_i.
    - The reverse of the Skip-Gram model in some sense.

- Let A be the adjacency matrix, and let D be the diagonal matrix with the degree of each vertex on the diagonal.
- $(D^{-1}A)^r$ is the transition matrix for moving r steps.

DeepWalk Matrix Representation (NetMF)

- Take average of all transition matrices $(D^{-1}A)^r$ in the context window.
- Multiply by D^{-1} to account for prior probability.
- Multiply by constant and take truncated logarithm (max of log(n) and 0).

$$M \triangleq \text{trunc\_log}^{\circ}\left(\frac{\text{vol}(G)}{b}\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r D^{-1}\right)$$

vol(G): number of edges

b: number of negative samples

- Computing the transition matrices is expensive!

Design and Implementation

- High-Level Optimizations
- Spectral Propagation
- System Construction

System Construction Visualized

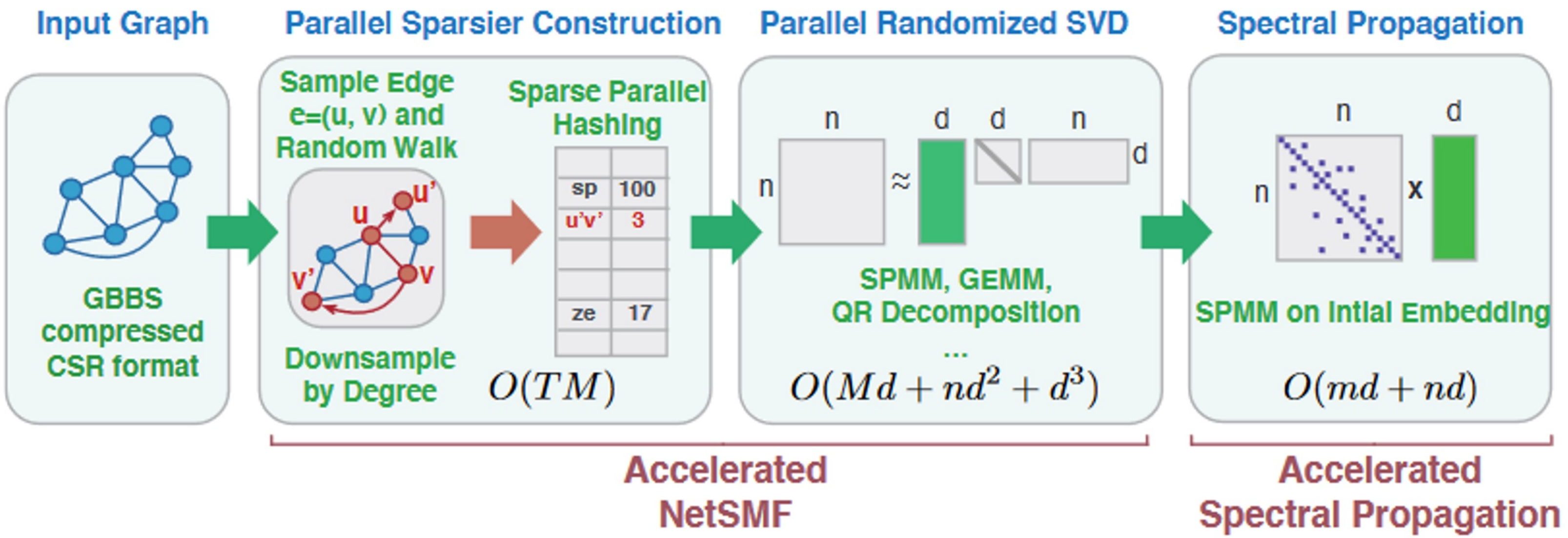


Figure 1: System overview of LightNE.

High-Level Optimizations

- NetSMF: Approximates transition matrices by sampling random walks.
  - $\Theta(E)$ samples are sufficient for a good approximation.
  - Resulting matrices are sparse, so sparse matrix construction can be used.
- Edge downsampling (novel contribution): Exclude edges in random walks from the matrices with edge-weighted probability.
  - Reduces costs of creating sparsified matrix.

Spectral Propagation

- After factorization, adjust the resulting matrix to get a better embedding.
- Takes step from ProNE: Multiply resulting matrix by a polynomial of the normalized Laplacian matrix $I - D^{-1}A$.
    - It appears that this results in better accuracy.

System Construction

- Uses Graph-based Benchmark Suite (GBBS).
    - First network embedding implementation that uses GBBS!
- Graph compression: neighbors of a vertex stored in differential-encoded blocks.
- Choosing first edge in random walk sampling: Choose every edge a random number of times.
    - Avoids decompression and binary search.
    - Choosing later edges in random walk is efficient with small differential-encoded blocks.

System Construction

- Matrix construction: count edges with sparse parallel hashing.
- Matrix factorization: Randomized SVD.
    - New algorithm based on Intel Math Kernel Library.

System Construction Visualized

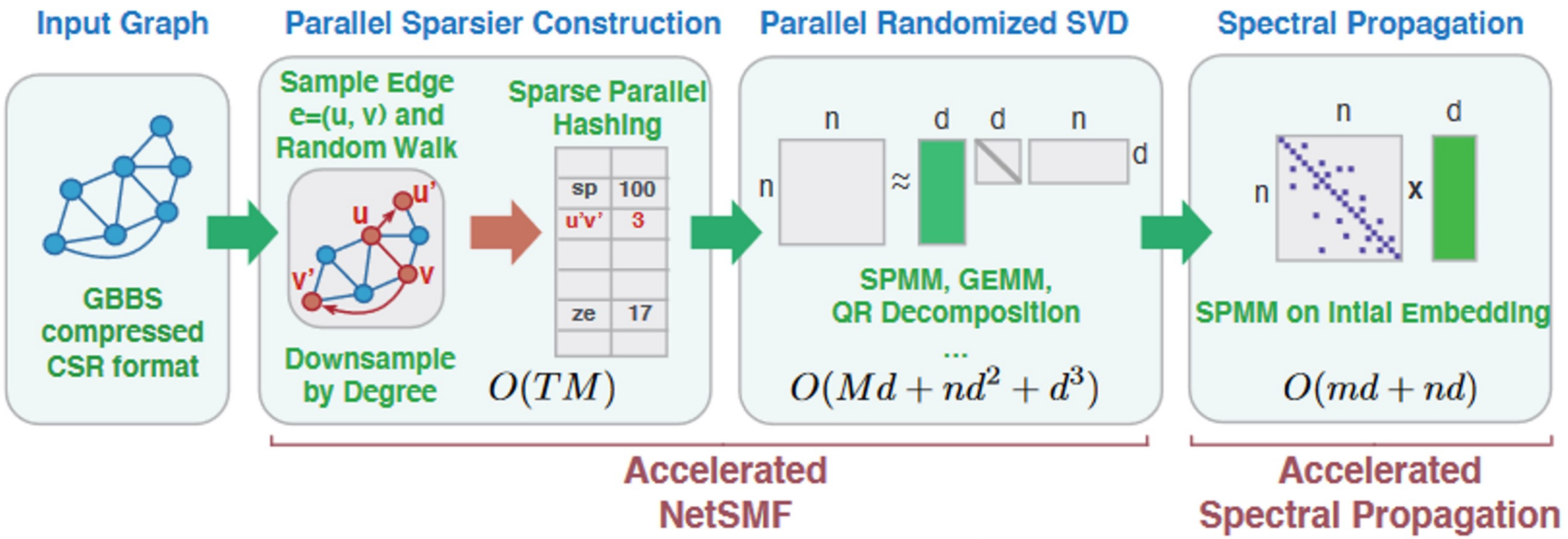


Figure 1: System overview of LIGHTNE.

Evaluation

Ran on server with two Intel Xeon E5-2699 v4 CPUs.

Comparison algorithms designed for big graphs: PyTorch-BigGraph (PBG), GraphVite, and NetSMF.

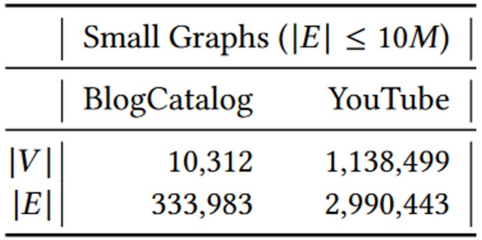
- Unclear whether ProNE is designed for big graphs

Evaluation tasks:

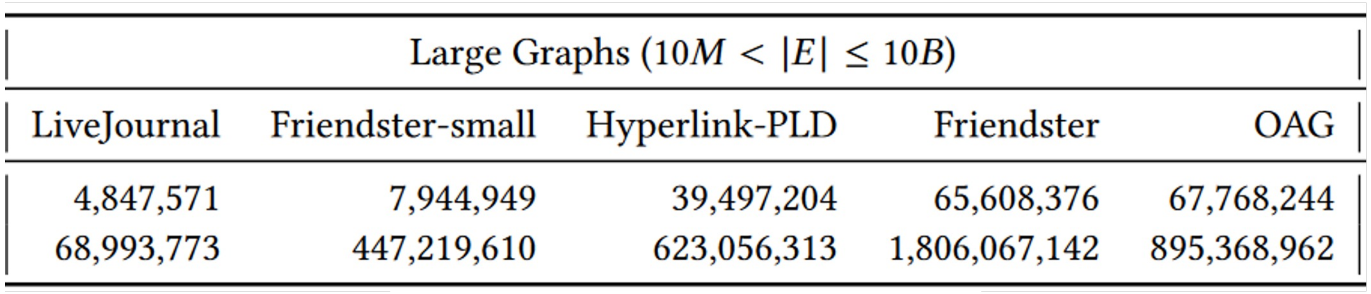
- Link prediction: determining whether two nodes are connected.
- Node classification: determining the label of a node.

It’s unclear what algorithms are used on the embeddings to complete the tasks.

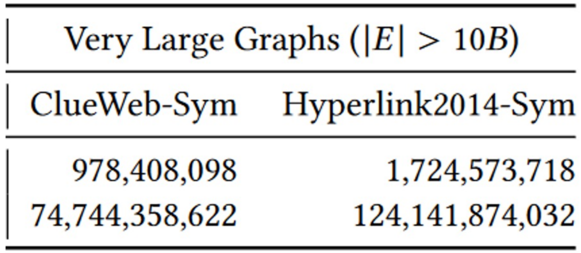
Evaluation Datasets



	Small Graphs ($\lvert E\rvert \leq 10M$)	
	BlogCatalog	YouTube
$\lvert V\rvert$	10,312	1,138,499
$\lvert E\rvert$	333,983	2,990,443



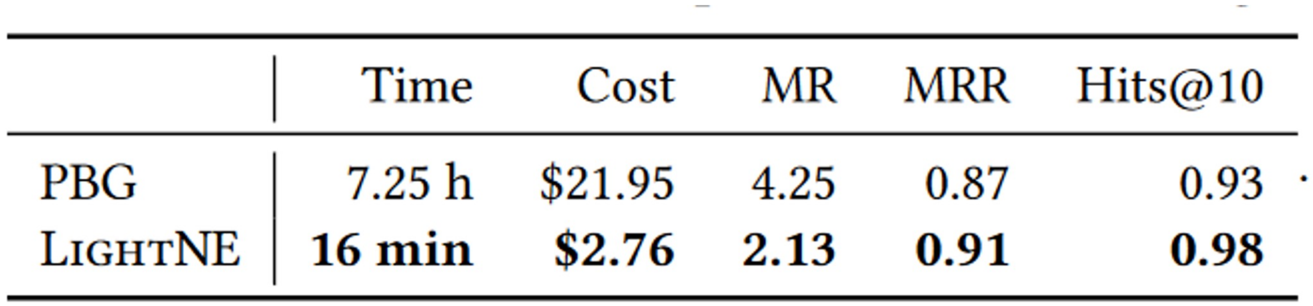
Large Graphs ($10M < \lvert E\rvert \leq 10B$)				
LiveJournal	Friendster-small	Hyperlink-PLD	Friendster	OAG
4,847,571	7,944,949	39,497,204	65,608,376	67,768,244
68,993,773	447,219,610	623,056,313	1,806,067,142	895,368,962



Very Large Graphs ($\lvert E\rvert > 10B$)	
ClueWeb-Sym	Hyperlink2014-Sym
978,408,098	1,724,573,718
74,744,358,622	124,141,874,032

Results

LightNE vs. PBG on Livejournal dataset:

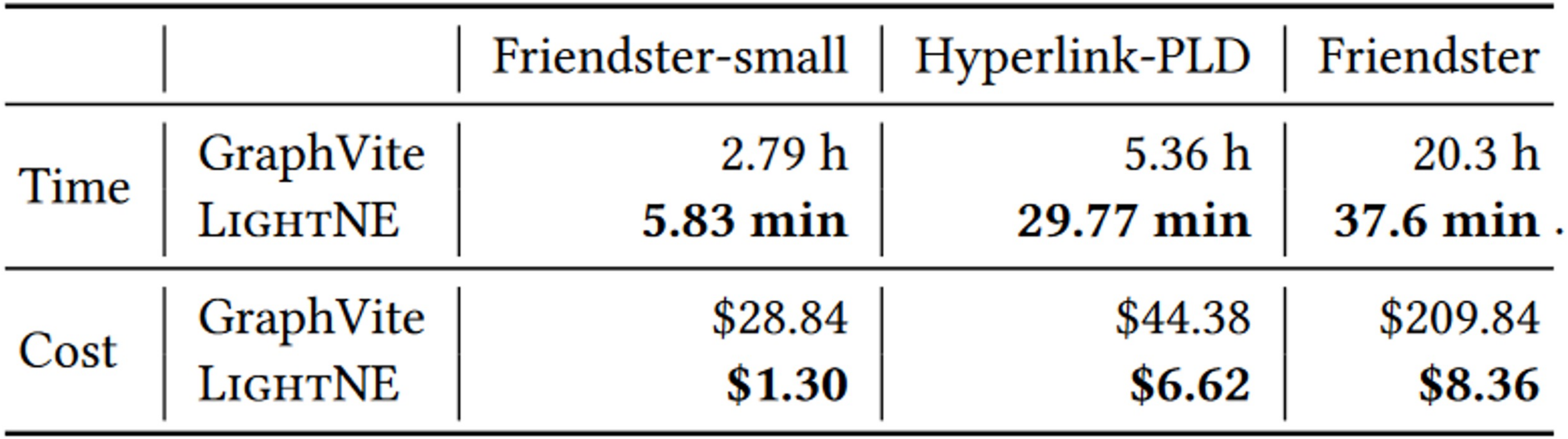


	Time	Cost	MR	MRR	Hits@10
PBG	7.25 h	$21.95	4.25	0.87	0.93
LightNE	**16 min**	**$2.76**	**2.13**	**0.91**	**0.98**

LightNE achieves better accuracy on all metrics.

Results

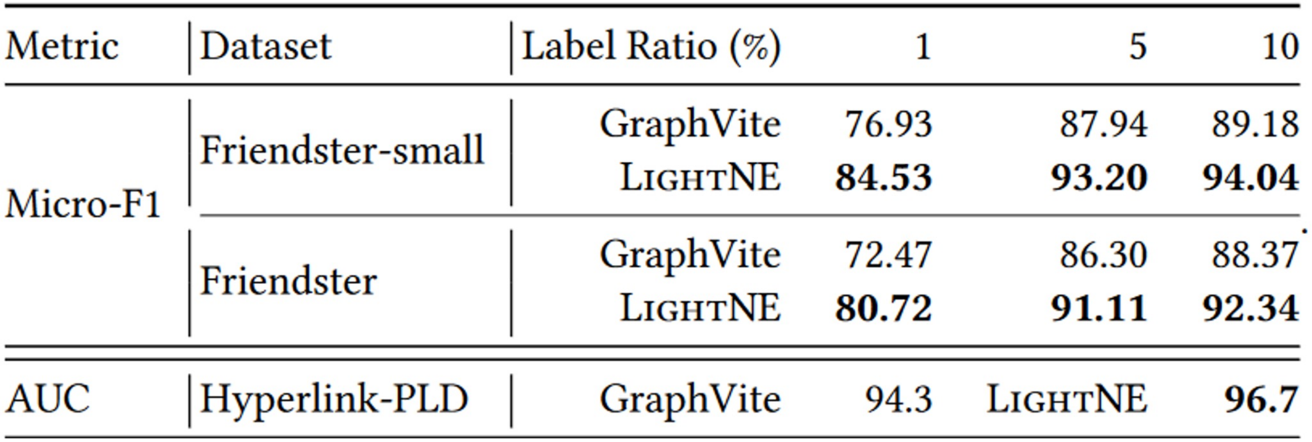
LightNE vs GraphVite:



		Friendster-small	Hyperlink-PLD	Friendster
Time	GraphVite	2.79 h	5.36 h	20.3 h
	LightNE	**5.83 min**	**29.77 min**	**37.6 min** .
Cost	GraphVite	$28.84	$44.38	$209.84
	LightNE	**$1.30**	**$6.62**	**$8.36**

Results

LightNE vs GraphVite:



Metric	Dataset	Label Ratio (%)	1	5	10
Micro-F1	Friendster-small	GraphVite	76.93	87.94	89.18
		LightNE	**84.53**	**93.20**	**94.04**
	Friendster	GraphVite	72.47	86.30	88.37
		LightNE	**80.72**	**91.11**	**92.34**
AUC	Hyperlink-PLD	GraphVite	94.3	LightNE	**96.7**

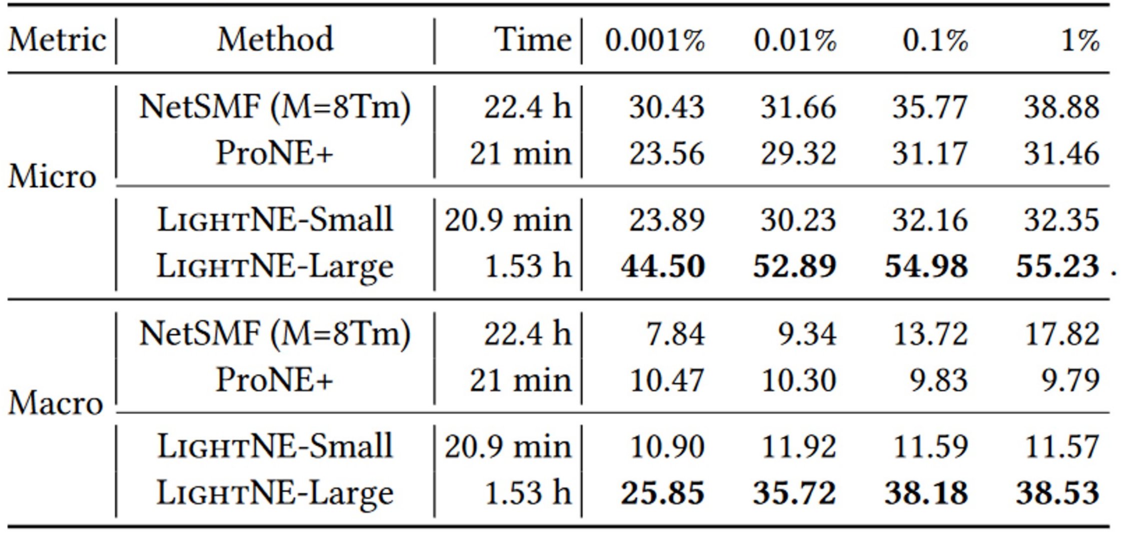
LightNE again achieves better accuracy on all metrics.

Results

LightNE vs. NetSMF and ProNE+:

This part uses two versions of LightNE: LightNE-Small is faster than LightNE-Large but has poorer accuracy.

Table 4: Comparison on OAG with label ratio 0.001%, 0.01%, 0.1% and 1%.



Metric	Method	Time	0.001%	0.01%	0.1%	1%
Micro	NetSMF (M=8Tm)	22.4 h	30.43	31.66	35.77	38.88
	ProNE+	21 min	23.56	29.32	31.17	31.46
	LightNE-Small	20.9 min	23.89	30.23	32.16	32.35
	LightNE-Large	1.53 h	**44.50**	**52.89**	**54.98**	**55.23** .
Macro	NetSMF (M=8Tm)	22.4 h	7.84	9.34	13.72	17.82
	ProNE+	21 min	10.47	10.30	9.83	9.79
	LightNE-Small	20.9 min	10.90	11.92	11.59	11.57
	LightNE-Large	1.53 h	**25.85**	**35.72**	**38.18**	**38.53**

Results Summary

LightNE gets a large speedup and accuracy gain over NetSMF, GraphVite, PBG.

LightNE-Small is a slight improvement over ProNE+ in speed and accuracy.

LightNE-Large gets better accuracy than ProNE+ but slower speed.

LightNE Parameters

It's possible to improve speed by taking fewer edge samples (LightNE-Small vs. LightNE-Large).

There is a tradeoff between speed and evaluation scores

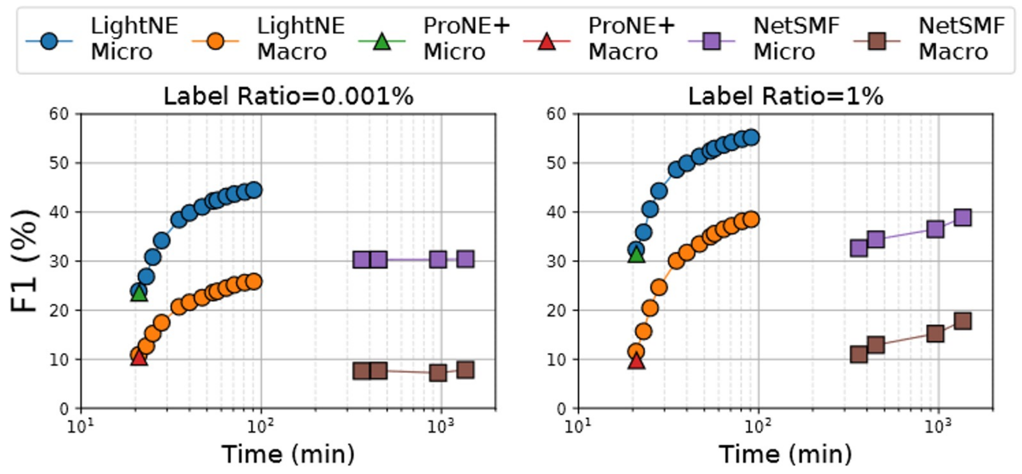


Figure 2: Efficiency-effectiveness trade-off curve of LIGHTNE.

Ablation Studies

How much do the optimizations help performance?

- By removing downsampling, the authors determined that downsampling increased the maximum number of samples by 60%.
- By comparing to NetSMF, sparse parallel hashing increased the maximum number of samples by 56%.

Testing on Large Graphs

- ClueWeb-Sym: 74 billion edges
- Hyperlink2014-Sym: 64 billion edges
- Matrix sparsifier only requires O(n log n) space (n is number of vertices)
- Not enough memory for spectral propagation
- Training time under 2 hours

Any questions?