Distributed Community Detection on Edge-labeled Graphs using Spark

San-Chuan Hung
Carnegie Mellon University
Pittsburgh, USA
c2016.tw@gmail.com

Miguel Araujo
Carnegie Mellon University &
INESC TEC
Porto, Portugal
maraujo@cs.cmu.edu

Christos Faloutsos
Carnegie Mellon University
Pittsburgh, USA
christos@cs.cmu.edu

ABSTRACT

How can we detect communities in graphs with edge-labels, such as time-evolving networks or edge-colored graphs? Unlike classical graphs, edge-labels contain additional information about the type of edges, e.g., when two people got connected, or which company hosts the air route between two cities. We model community detection on edge-labeled graphs as a tensor decomposition problem and propose TeraCom, a distributed system that is able to scale in order to solve this problem on 10x larger graphs. By carefully designing our algorithm and leveraging the Spark framework, we show how to achieve better accuracy (in terms of recovering ground-truth communities) when compared to PARAFAC methods - up to 30% increase in NMI. We also present interesting clusters discovered by our system in a flights network.

CCS Concepts

•**Information systems** → *Data mining;* •**Theory of computation** → *MapReduce algorithms;*

Keywords

Community Detection, Tensor Factorization, Tensor Decomposition, Distributed Computing

1. INTRODUCTION

How can we detect communities in graphs with edge-labels, such as time-evolving networks or edge-colored graphs? For example, social network edges connecting two users might carry information regarding the time at which they became friends, in which case the edge creation time is the label of the edge. Airlines also form edge-labeled graphs, as each air route connecting two cities is hosted by a specific airline company. Finally, english sentences can also be described as edge-labeled graphs: a node might represent a subject or an object phrase, while a subject-verb-object sentence might be represented as an edge between two nodes whose verb is the edge label.

There are many community detection methods designed for classical graphs [12, 11, 13], but they do not leverage the additional information provided by these labels. Detecting communities in edge-labeled graphs is still a developing research topic [5, 31].

We provide a scalable and fast solution for this problem by developing TeraCom, a system based on the Spark framework. We carefully design an algorithm that is able to iteratively reduce the problem size in order to speed-up each subsequent iteration and, according to our experiments, our system is able to find communities in larger graphs while keeping a competitive accuracy when compared to standard tensor decomposition methods.

Figure 1 illustrates our method's ability to find meaningful communities in a real airports dataset. Edge-labels correspond to airlines and have been omitted for clarity.

Our main contributions can be summarized as follows:

- **Scalability**. Our system is able to detect communities in edge-labeled graphs 10X bigger when compared to existing tools. Furthermore, our system speeds-up near linearly when enough machines are provided.
- **Effective Algorithm**. We carefully reorder operations in order to reduce the problem size after each iteration, enabling faster computation.
- **Discoveries**. Our system discovers spatially affine groups in an airline network, shown in Figure 1. For example, our system is able to distinguish low-cost airlines and standard airlines in Europe.

2. BACKGROUND

2.1 Edge-labeled Graphs and Tensors

Let $\mathbf{G} = (\mathbf{V}, \mathbf{L}, \mathbf{E})$ be an edge-labeled graph, where $\mathbf{V}$ is the set of nodes, $\mathbf{L}$ is the set of categorical labels and $\mathbf{E}$ is the set of edges $\{(v_1, v_2, l) \in \mathbf{E} | v_1 \in \mathbf{V}, v_2 \in \mathbf{V}, l \in \mathbf{L}\}$. We can model this edge-labeled graph as a binary three-mode tensor.

A N-mode tensor $\mathbf{X} \in \mathbf{R}^{I_1 \times I_2 \times I_3 \ldots \times I_N}$ is a n-dimensional array used to describe inter-relational data. A one-mode tensor is an array, a two-mode tensor is a matrix, etc. We will be describing three-mode tensors in the rest of the paper.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

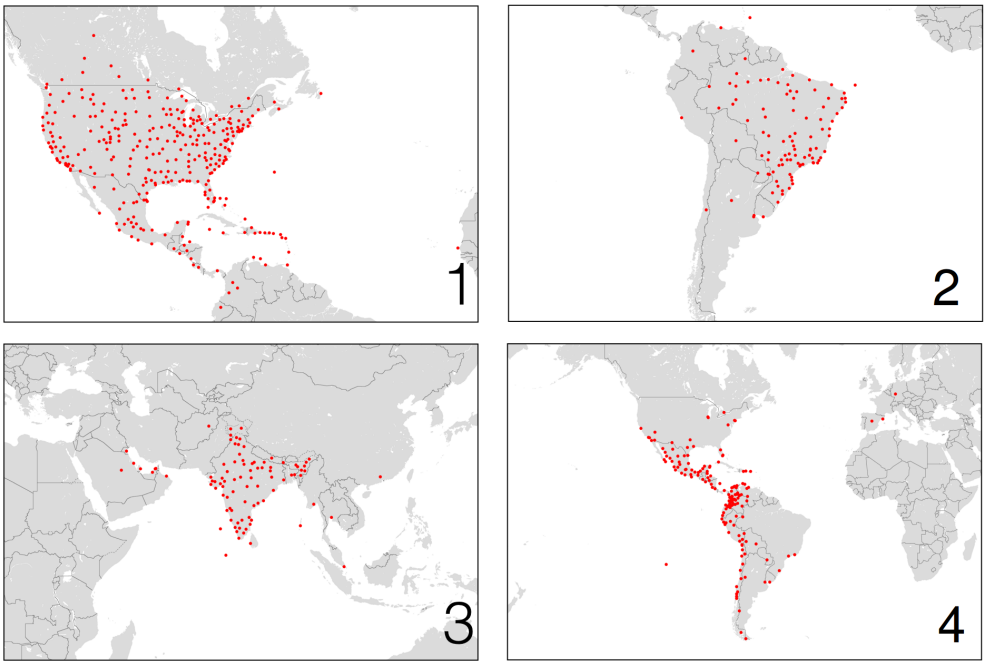

Figure 1: Communities detected in flights dataset: (1) America, (2) Brazil, (3) India, (4) Latin America. Each red dot represents an airport.

2.2 Tensor Decomposition and Community Detection

An edge-labeled graph may contain sets of highly interconnected nodes, which we typically call communities. For instance, in an airline graph, the airports in America may form a community because they are connected by multiple air routes. Furthermore, these flights are hosted by a subset of the many airline companies. Therefore, a community in our edge-labeled airports network would be represented by two sets of airports (sources and destinations) and one set of airlines.

When representing edge-labeled graphs as tensors, tensor-decomposition methods are known to aid in the detection of communities. PARAFAC [7] is a tensor decomposition method that approximates the original tensor through a combination of multiple low-dimensional factors. For example, an original tensor X can be decomposed into R factors $(\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}, \lambda_1), (\mathbf{a_2}, \mathbf{b_2}, \mathbf{c_2}, \lambda_2)...(\mathbf{a_R}, \mathbf{b_R}, \mathbf{c_R}, \lambda_R)$, where $\mathbf{a_1}$, ... $\mathbf{a_R}$, $\mathbf{b_1}$ ... $\mathbf{b_R}$, $\mathbf{c_1}$ ... $\mathbf{c_R}$ are unit vectors, and λ_1, λ_2... λ_R are scalars. The $x_{i,j,k}$ value in the original tensor X can be reconstructed by summing up all factors:

$$\bar{x}_{i,j,k} = \sum_{r=1...R} \lambda_r \cdot \mathbf{a_{r,i}} \cdot \mathbf{b_{r,j}} \cdot \mathbf{c_{r,k}} \qquad (1)$$

If a tensor has clustering structure, the decomposed factors are known to capture different clusters in the tensor. Figure 2 shows the ideal case: each factor captures a different cuboid in the tensor and the decomposed factors can be used to obtain the edge-labeled communities of our network. In practice, due to noise and to the non-uniform structure that communities exhibit [3], factors matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ won't represent cuboids and different techniques are necessary to obtain binary membership.

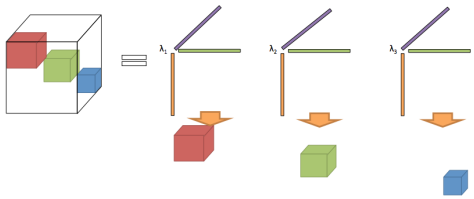

Figure 2: Community Detection with PARAFAC, if elements in the factors could be easily clustered.

3. RELATED WORK

Community Detection on Graphs Classical algorithms find communities in unlabeled graphs through the analysis of the network topology. In general, they look for groups of nodes which achieve high internal connectivity while having few connections to the rest of the graph. There are multiple approaches which can be classified in different categories. The assumption behind graph-partition methods [13] is that groups are connected by bridge edges which will emerge if we cut them; therefore, these methods first define edge centrality and then remove edges repeatedly to form clusters. On the other hand, cluster optimization approaches transform community detection into optimization problems; they define what a "good" cluster is and then apply a strategy to optimize the goodness metric. Modularity-based methods [25, 24, 11], Weighted Community Clustering methods [29, 30], propinquity methods [10] can be classified in this approach. RWR-based methods [8] group nodes by RWR score, i.e. the probability of landing on a node given random walks from a set of seed nodes. Label propagation methods [15] assume that each node has a label which indicates to which group it belongs and continuously update each node's label based

Symbol	Definition
G	a edge-labeled graph
V	set of nodes
L	set of labels
E	set of labeled edges
a, **b**, **c**	column vectors
A, **B**, **C**	factor matrices
λ	scalar weight
m	a community
M	set of communities
R	the rank of the decomposition
T	the number of inner-round of the rank-1 decomposition

Table 1: Symbol Table

on the majority label of neighboring nodes.

Community Detection on Edge-labeled Graphs Aforementioned methods rely on network topology information to detect communities, ignoring the information carried by edge-labels.

Com^2 [4, 5] and TimeCrunch [31] are community detection algorithms for edge-labeled graphs. The basic ideas are similar: they find patterns that minimize the description length (MDL) of the tensor, i.e. a cost for encoding both patterns and errors. The assumption is that patterns identified in this minimization correspond to communities and can be easily interpretable: the patterns are simple, and the errors are identified. Com^2 starts by performing candidate selection through a rank-1 approximation to decompose the tensor in 3 vectors that a hill-climbing algorithm optimizing MDL will use to guide the search. TimeCrunch also finds patterns by minimizing the MDL, but it focuses on explainable time-evolving patterns instead. It first slices the time-evolving graph into static graphs with different time stamps and then uses VoG [21] to mine patterns in these static graphs. It then stitch the patterns in different time periods, building a pattern-node matrix, and uses matrix factorization to cluster the patterns in different time steps.

Probabilistic Approach for Dynamic Graphs Many models [32, 35, 33, 14, 28] detect hidden structures (e.g. community assignment [32], group evolving paths [35], or hidden class interactions in different time periods [33]) inside dynamic graphs through variations of the Dynamic Stochastic Block Model: they extend it by adding different hidden structural assumptions with varying probability distributions, and then use the available data to infer the most likely parameters. In our work, the hidden class assignment are captured as factor vectors and computed by tensor decomposition. Besides, our work does not assume the input graphs should be temporal, but it can be applied to graphs with any categorical labels on the edges.

Tensor Analysis and Tools Multiple tensor analysis toolboxes implementing PARAFAC have been developed. While these systems implement the standard PARAFAC [7] decomposition, our work is focused on *binary tensors* and we use iterated rank-1 decompositions to speed-up the algorithm. Besides, while some of these systems are distributed on Hadoop, our system is based on Spark [36] which allows efficient memory-based operations, not requiring data to be spilled to disk.

Tensor Toolbox [6] is a well-known MATLAB implementation of many tensor decomposition algorithms, such as nonnegative PARAFAC with alternative Poisson regression [16] and PARAFAC via optimization [2]. ParCube [26] introduced a sampling process before the tensor decomposition algorithm so that bigger problems could be tackled.

The difference between our system and these two tools is that (1) these tools are designed for single machine use, while our system runs on a distributed framework and (2) our system is able to cluster the most relevant elements on each mode, instead of scoring elements individually.

GigaTensor [18] developed a large-scale PARAFAC [19] algorithm for tensor decomposition based on Hadoop. Their idea is that by decoupling the product terms they are able to avoid the data explosion known to be generated by intermediate matrices, and by storing the small matrices in the distributed cache they are able to speedup matrix multiplication. HaTen2 [17] is also a Hadoop-based PARAFAC system from the same authors. Similar to GigaTensor, they both aim to avoid the intermediate data explosion problem. The authors claims that HaTen2 is an improvement over GigaTensor.

Apache Spark Spark [36] is an in-memory MapReduce-like general-purpose distributed computation platform which provide a high-level interface for users to build applications. Unlike previous MapReduce frameworks like Hadoop, Spark mainly stores intermediate data in memory, effectively reducing the number of disk Input/Output operations. The main abstraction in Spark is called a Resilient Distributed Dataset (RDD), which represents a collection of read-only objects. RDDs can be constructed from files, vectors or derived from other RDDs, in which case the operation is called a *transformation* (typical transformations are *map*, *reduce*, *flatmap* of *filter*). RDDs aren't always saved to disk and they are computed only when needed.

Spark allows us to use multiple machines and multiple cores. Perhaps more importantly, our algorithms are able to manipulate more memory, which means that our system can be applied to larger datasets.

4. PROPOSED SYSTEM

4.1 Rank-1 Tensor Decomposition

Given a three mode tensor $\mathbf{X}$, we want to find vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ which approximate tensor $\mathbf{X}$ with minimal error.

$$Er = \sum_i \sum_j \sum_k (x_{i,j,k} - \lambda \cdot \mathbf{a}_i \cdot \mathbf{b}_j \cdot \mathbf{c}_k)^2 \quad (2)$$

$$\mathbf{a}, \mathbf{b}, \mathbf{c} = \min_{\mathbf{a},\mathbf{b},\mathbf{c}} Er \quad (3)$$

$\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ can be solved by Alternative Least Squares (ALS): we start by randomly initializing $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and then alternatingly update one of the vectors while assuming that the others are fixed. T

LEMMA 1. *Using Alternating Least Squares (ALS), the updates of vectors* $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ *are given by:*

$$\mathbf{a}_i = \frac{1}{Z_a} \sum_j \sum_k \mathbf{b}_j \cdot \mathbf{c}_k \cdot x_{i,j,k} \quad (4)$$

$$\mathbf{b}_j = \frac{1}{Z_b} \sum_j \sum_k \mathbf{a}_i \cdot \mathbf{c}_k \cdot x_{i,j,k} \quad (5)$$

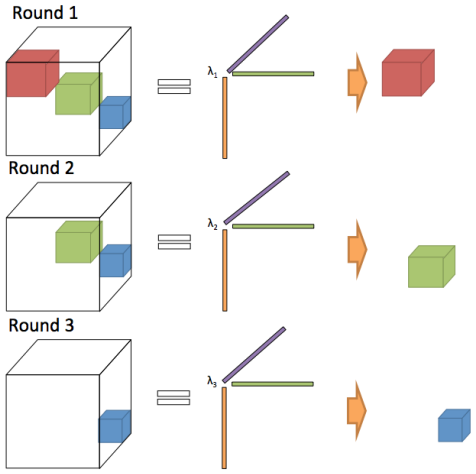

Figure 3: Community detection with our system

$$\mathbf{c}_k = \frac{1}{Z_c} \sum_j \sum_k \mathbf{a}_i \cdot \mathbf{b}_j \cdot x_{i,j,k} \tag{6}$$

where Z_a, Z_b and Z_c are normalization constants that guarantee unit-norm, and $\lambda = Z_a \cdot Z_b \cdot Z_c$.

A sketch of the proof can be found in the appendix, it follows from the general ALS method [9].

4.2 Algorithm Design

4.2.1 *Community Detection*

Algorithm 1 TeraCom

Input: $\mathbf{E}, R$
Output: $\mathbf{M}$

for r in 1...R **do**
  // factorization step
  $\mathbf{a}, \mathbf{b}, \mathbf{c} = rank1Decomposition(E)$
  for e in $\mathbf{E}$ **do**
    // thresholding step
    if $\mathbf{a}_{e.i} \geq 1/|\mathbf{a}|$ and $\mathbf{b}_{e.j} \geq 1/|\mathbf{b}|$ and $\mathbf{c}_{e.k} \geq 1/|\mathbf{c}|$ **then**
      // tensor deflation step
      Add e into m_r
      Remove e from $\mathbf{E}$
    end if
  end for
end for
return $\mathbf{M} = \{m_1, m_2, m_3, ..., m_k\}$

Instead of simultaneously decomposing the tensor into R factors like PARAFAC, we propose a distributed algorithm based on consecutive rank-1 decompositions that extracts communities one by one.

As shown in Figure 3 and Algorithm 1, one round of our method is composed of three steps:

1. The **Factorization step** uses a rank-1 decomposition to extract one set of factor vectors **a**, **b**, and **c**.

2. The **Thresholding step** will establish which elements belong to the current community. It marks an entry $e_{i,j,k}$ to be part of the r-th community if its corresponding factor scores $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$ are larger than the average score of the respective factor vectors.

3. The **Tensor deflation step** saves the marked entries as a community and then removes the non-zero elements corresponding to this community from the tensor. After running the process R times, it will generate R distinct communities.

The advantage of this detection-removal process is that its shrinks the tensor after each iteration, speeding up subsequent rounds. As the tensor is sparsely encoded (only the non-zero elements take physical memory), after each iteration some non-zero elements are marked as part of the currently detected community and removed. Therefore, the number of non-zeros of the tensor decreases after each iteration and less data needs to be processed in later stages.

Algorithm 2 shows the rank-1 decomposition process. Vectors **a**, **b**, **c** are first randomly initialized and normalized. In each inner-round, we update **a**, **b**, and **c** individually, considering that the other vectors are fixed, using their most up-to-date version.

This process runs multiple times until we achieve convergence. Parameter T decides the number of inner-rounds of the rank-1 decomposition: more inner-rounds provide higher precision at the cost of a longer execution time.

Algorithm 2 Rank-1 Decomposition

Input: **E**
Output: **a**, **b**, **c**

randomly initialize normalized vectors: **a**, **b**, **c**
for r in 1...T **do**
  for i in 1 ... $|\mathbf{a}| - 1$ **do**
    $\mathbf{a}_i = \sum_{e_{i,j,k} \in \mathbf{E}} \mathbf{b}_j * \mathbf{c}_k$
  end for
  $\mathbf{a} = Normalize(\mathbf{a})$
  for j in 1 ... $|\mathbf{b}| - 1$ **do**
    $\mathbf{b}_j = \sum_{e_{i,j,k} \in \mathbf{E}} \mathbf{a}_i * \mathbf{c}_k$
  end for
  $\mathbf{b} = Normalize(\mathbf{b})$
  for k in 1 ... $|\mathbf{c}| - 1$ **do**
    $\mathbf{c}_k = \sum_{e_{i,j,k} \in \mathbf{E}} \mathbf{a}_i * \mathbf{b}_j$
  end for
  $\mathbf{c} = Normalize(\mathbf{c})$
end for
return **a**, **b**, **c**

4.3 Implementation Design

Our implementation is carefully designed to obtain the best speed-up and scalability:

We **limit the number of groupBy operations** by partitioning the data among machines so that edges of the same element are available at the same physical location, minimizing data shuffling.

We **cache reused RDD in memory**, minimizing disk accesses between consecutive iterations, which would not be possible if using a system like Hadoop to distribute the computation.

Finally, we **encode shared state using broadcast variables**. For each inner-iteration, we broadcast the latest version of vectors **a**, **b** and **c** (which are small) to all machines.

For each dimension, non-zeros are then mapped to the respective product of equation 4, aggregated and normalized.

5. EXPERIMENTS

Our system is evaluated by answering the following questions:

Q1: How accurately does our system detect communities in edge-labeled graph comparing to other tensor decomposition methods?

Q2: How does our system scale with graph size? Can we solve larger problems than other tools?

Q3: How does our system scale with the number of machines? Does it scale linearly?

Q4: What can our system discover in real world edge-labeled graphs?

5.1 Precision

We are interested in analyzing precision in a community detection setting where hard membership constraints are in place: an element either belongs to a community or it doesn't. Therefore, we compare our system with modified versions of standard [7] and non-negative [22] PARAFAC.

5.1.1 Baseline Methods

We consider three distinct baseline approaches which we call *thresholded PARAFAC* (PARAFAC-THRESH), *absolute-valued PARAFAC* (PARAFAC-ABS) and *thresholded Non-Negative PARAFAC* (NN-PARAFAC-THRESH). In all baselines, we start by apply the respective standard PARAFAC or Non-Negative PARAFAC decompositions and then select elements as being part of the community if their score in the factor vector is greater than the average value of the vector. In *absolute-valued PARAFAC*, we take the absolute value of the score before thresholding instead, as it is possible that negative scores in standard PARAFAC affect the community definitions.

We use Matlab's Tensor Toolbox [6] as the default implementation of PARAFAC and Non-Negative PARAFAC.

5.1.2 Data Description

We use three datasets with ground-truth communities to evaluate accuracy: two synthetic datasets with artificial communities and the DBLP dataset [23], where journal names act as community ground-truth labels.

In the first synthetic dataset (Small Cubes), we generated 10 disjoint highly dense cubes in a tensor to form an edge-labeled graph. Each cube represents a trivial community with multiple edge-labels connecting its nodes. For the second dataset, we generated 5 disjoint random-subgraphs (5-DRS). A 5-DRS graph contains 5 different-size random-subgraphs, whose nodes are randomly connected. There are no links between different subgraphs. As each subgraph represents a community, there are 5 trivial communities in the data. Each sub-graph contains twice the number of edges as the previous, with the total numbers of edges of the graphs ranging from 100K to 100M.

The DBLP dataset is a standard co-authorship network with edges labeled with the year(s) in which the authors collaborated. We only use data from the top-100 journals to form the graph.

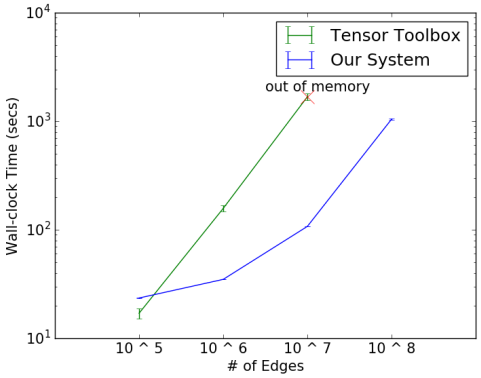

Figure 4: Data Scalability Experiment: Tensor Toolbox (1 machine) v.s. Our System (3 machines)

5.1.3 Experimental Result

We use Normalized Mutual Information (NMI[1]) [34] in order to create a confusion matrix between each factor and each community. We then assign each community to one factor, and calculate the average NMI score. Table 3 shows that our system has better accuracy than competing baselines, achieving 0.9995 NMI in the small cubes dataset and 0.5908 in the DBLP-100 dataset.

Name	Small Cubes	DBLP-100
our system	**0.9995**	**0.5908**
PARAFAC-THRESH	0.9779	0.5209
NN-PARAFAC-THRESH	0.8445	0.5404
PARAFAC-ABS	0.9893	0.4292

Table 3: Precision Experiment

5.2 Scalability

5.2.1 Data Scalability

We are interested in analyzing the speed-up of our system compared to single machine tools. As a baseline, we consider the Matlab's Tensor Toolbox [6] - a tensor decomposition tool which is the state of the art tensor computation package designed for single machine [17].

The numbers of inner-round iterations of the Tensor Toolbox and of our system affect both their speed and accuracy. In order to perform a fair comparison, we first fix the precision of the Tensor Toolbox, then find the required number of inner-iterations for our system to achieve a lower error, and finally compare their running time at this precision level.

We run Tensor Toolbox on a single machine with 15GB of memory and 12 Intel(R) Xeon(R) 3.20GHz CPUs. We build a Spark cluster on 3 AWS m3.xlarge instances; each instance has 15GB memory and 4 Intel Xeon E5-2670 2.6GHz CPUs. We run each experiment three times to compute the average and the deviation of wall-clock time.

Figure 4 shows the experimental results: our system can be applied to 10X larger graphs, while still running faster

[1]NMI values range from 0 to 1 and higher NMI values are better.

Name	|V|	|E|	|L|	|communities|
10-cubes	130	33786	150	10
5-disjointed-subgraph	1K – 1M	100K – 100M	1K – 1M	5
DBLP-top100	2244	5054	45	100
Airline	3162	58443	532	N/A
NELL	74M	144M	26M	N/A

Table 2: Dataset description: 2 synthetic and 3 real-world datasets.

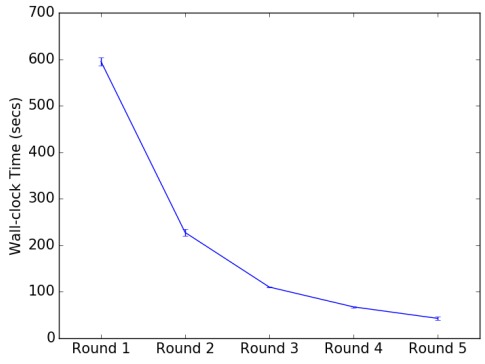

Figure 5: Execution time decreases in each outer-iteration.

in most scenarios. Note that the Tensor Toolbox cannot be executed on the the fourth graph (10^8 edges) because it runs out of memory. Furthermore, please note the logarithmic axis in this figure. For example, in the third graph (10^7 edges), Tensor Toolbox requires 1679.89 seconds on average, but our system only needs 107.40 seconds - a 15.64X improvement.

The speedup comes from two aspects:

1. The Spark framework contributes to the speedup, as it dispatch the tasks to multiple machines and executes the computation in parallel.

2. Our algorithm shrinks the problem size after each outer-iteration: it saves the detected community into a file and removes the edges from the graph.

Figure 5 shows the wall-clock time of each outer-iteration as our system processes the fourth graph (10^8 edges). Note the significant decrease in running time. Furthermore, note that the Tensor Toolbox is faster on small datasets (10^5 edges) than our system, as the Spark framework adds overhead of coordinating jobs and resources across machines.

5.2.2 *Machine Scalability*

The machine scalability experiment tests the speed-up of our system when we increase the number of machines. As the execution time will be determined by the number of inner-rounds of the rank-1 decomposition, we focus on measuring the time of a single iteration.

We test our system on the NELL dataset. NELL is a Subject-Object network: each node represents a subject or an object, and there is a link between a subject and an object if a valid subject-verb-object concept exists. Therefore, edges are labeled with verbs of the English language.

The size of Spark cluster ranges from 3 to 20 AWS r3.xlarge instances. One r3.xlarge instance contains 30GB of memory and 4 Intel Xeon E5-2670 2.60 GHz CPUs. As before, we execute each experiment three times to acquire the average and the deviation of the wall-clock time. We consider the 5-machines wall-clock time (T_5) as the baseline and show relative speedup ratios T_m/T_5 in figure 6.

We note the following: (1) nearly linear speedup from 5 to 10 machines. Our system is 166.71% faster with 8 machines and 193.13% faster with 10 machines, compared to a 5 machines cluster. (2) The 3-machines cluster does not performs as well as expected; although it has more than half of 5 machines, its relative speedup is only 30.95%. The reason is that the small cluster does not have sufficient memory to store the working set of RDDs, causing the JVM to start the garbage collection (GC) often - it spends 91.29% of the running time running the GC. (3) the improvement in speed-up becomes negligible when the number of machines increases from 10 to 15 or 20. Although a 10-machines cluster shows a 193.13% speed-up, it only shows a 209.34% improvement for 15 machines and 210.02% for 20 machines.

There are three reasons explaining this last result. Firstly, the default number of tasks assigned by the Spark system is not large enough to fully leverage parallelism - Spark creates 33 tasks, while the 15 machine cluster has 48 cores available. Secondly, NELL is a NLP dataset: common words contain more links compared to unpopular words, leading the a skewed workload. When we increase the size of the cluster, although the average execution time of each task decreases on average, outlier tasks do not have a proportional speed-up to the increased number of cores. Finally, increasing the number of machines also increases network overhead.

The result shows that our system can provide nearly linear speedup if the cluster has enough memory to load working set RDDs into memory, and if Spark if configured to provide an adequate number of tasks for machines to utilize parallelism fully.

5.3 Discoveries

We apply our system to an Airlines dataset [4]. Each node corresponds to a city and a link represents an air route. Edge label correspond to airlines hosting these routes. We set the number of communities to 10, and the number of inner decomposition rounds to 20.

Our system can find spatially affine communities - the decomposed groups correspond to countries or regions. For example, our system detects the airline cluster of the USA, China, Europe, India, Brazil, Australia, South East Asia, Russia, and Latin America; some examples are shown in Figure 1.

More interestingly, our system is able to use the edge labels to distinguish two airline clusters in Europe, EU1 and

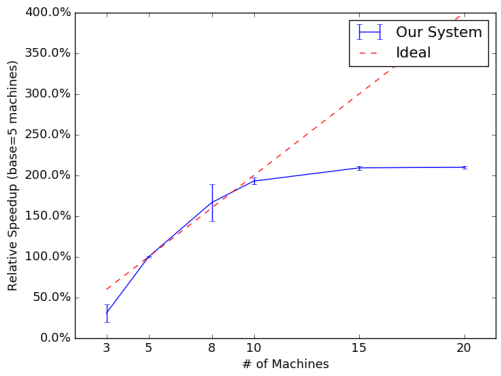

Figure 6: Machine Scalability (Base: 5 machines)

EU2 shown in Figure 7. In EU1, we can see air routes mainly hosted by low-cost airlines, such as Ryanair and easyJet - 74.15% of the flights are hosted by companies on the low-cost airline list [1]. In contrast, only 16.74% of the routes in EU2 are hosted by companies on this list. The results shows that our system fully uses the information of the airline network to detect communities: decisions are not only based on network topology, but also on edge labels.

More results can be found at http://leohung.net/taboo-decomposition-demo/ for interested readers.

6. CONCLUSION

This paper proposes a Spark-based distributed system for finding communities in edge-labeled graphs. Our contribution are summarized as follows:

- **Scalability**. Our system is able to detect communities in edge-labeled graphs 10X bigger when compared to existing tools. Furthermore, our system speeds-up near linearly when enough machines are provided.
- **Effective Algorithm**. We carefully reorder operations in order to reduce the problem size after each iteration, enabling faster computation.
- **Discoveries**. Our system discovers spatially affine groups in an airline network, shown in Figure 1. For example, our system is able to distinguish low-cost airlines and standard airlines in Europe.

Discussion. Performing iterative rank-1 decompositions that are followed by a tensor deflation is a well known technique that multiple algorithms rely on [20, 4], as evidence has been provided that, in sparse settings, it approximates the full-rank decomposition with very high accuracy [27]. We provide empirical evidence that, by avoiding such interactions, this method enables the identification of the underlying structure with less noisy (higher NMI) factors. We expect further research to better quantify this improvement and to analyze when can such techniques be more accurate than the slower full-rank decompositions.

7. ACKNOWLEDGMENTS

Miguel Araujo is partially supported by the COMPETE 2020 program within project POCI-01-0145-FEDER-006961 and by FCT (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program under Grant SFRH/BD/52362/2013 and as part of project UID/EEA/50014/2013.

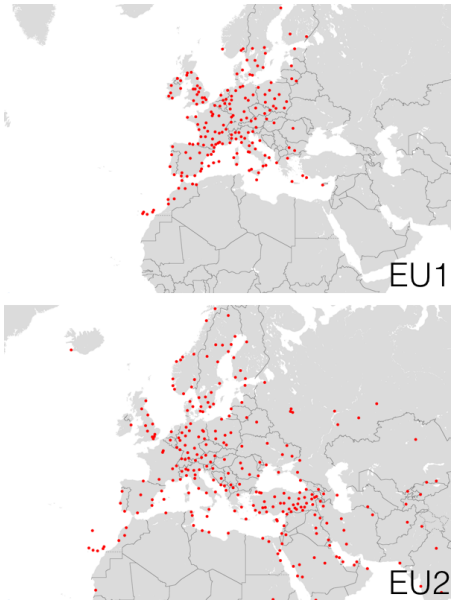

Figure 7: Two clusters of different European airlines. The air routes in EU1 are mainly hosted by low-cost airlines, while routes in EU2 are hosted by regular airlines.

8. REFERENCES

[1] List of low-cost airlines. https://en.wikipedia.org/wiki/List_of_low-cost_airlines.

[2] E. Acar, D. M. Dunlavy, and T. G. Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics*, 25(2):67–86, Jan. 2011.

[3] M. Araujo, S. Günnemann, G. Mateos, and C. Faloutsos. Beyond blocks: Hyperbolic community detection. In *Machine Learning and Knowledge Discovery in Databases*, pages 50–65. Springer, 2014.

[4] M. Araujo, S. Günnemann, S. Papadimitriou, C. Faloutsos, P. Basu, A. Swami, E. Papalexakis, and D. Koutra. Discovery of "comet" communities in temporal and labeled graphs Com^2. *Knowledge and Information Systems*, pages 1–21, 2015.

[5] M. Araujo, S. Papadimitriou, S. Günnemann, C. Faloutsos, P. Basu, A. Swami, E. E. Papalexakis, and D. Koutra. Com2: Fast Automatic Discovery of Temporal ('Comet') Communities. *PAKDD*, 8444(Chapter 23):271–283, 2014.

[6] B. W. Bader, T. G. Kolda, and others. MATLAB Tensor Toolbox Version 2.6. Feb. 2015.

[7] R. Bro. PARAFAC. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 38(2):149–171, 1997.

[8] B. Cai, H. Wang, H. Zheng, and H. Wang. An Improved Random Walk based Clustering Algorithm

for Community Detection in Complex Networks. pages 1–6, July 2011.

[9] J. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[10] Cosmos. Parallel Community Detection on Large Networks with Propinquity Dynamics. pages 1–9, Oct. 2015.

[11] J. Duch and A. Arenas. Community detection in complex networks using Extremal Optimization. *arXiv.org*, (2):027104, Jan. 2005.

[12] S. Fortunato. Community detection in graphs. *arXiv.org*, (3-5):75–174, June 2009.

[13] S. Fortunato and C. Castellano. Community Structure in Graphs. *arXiv.org*, Dec. 2007.

[14] A. Ghasemian, P. Zhang, A. Clauset, C. Moore, and L. Peel. Detectability thresholds and optimal algorithms for community structure in dynamic networks. *arXiv.org*, June 2015.

[15] S. Gregory. Finding overlapping communities in networks by label propagation. *arXiv.org*, (10):103018, Oct. 2009.

[16] S. Hansen, T. Plantenga, and T. G. Kolda. Newton-based optimization for Kullback-Leibler nonnegative tensor factorizations. *Optimization Methods and Software ()*, 30(5):1002–1029, 2015.

[17] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos. HaTen2: Billion-scale tensor decompositions. *ICDE*, pages 1047–1058, 2015.

[18] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos. GigaTensor: scaling tensor analysis up by 100 times - algorithms and discoveries. *KDD*, pages 316–324, 2012.

[19] T. G. Kolda and B. W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500, Aug. 2009.

[20] T. G. Kolda, B. W. Bader, and J. P. Kenny. Higher-order web link analysis using multilinear algebra. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[21] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. VOG: Summarizing and Understanding Large Graphs. *SDM*, pages 91–99, 2014.

[22] D. D. Lee and H. S. Seung. Algorithms for Non-negative Matrix Factorization. *NIPS 2014*, pages 556–562, 2000.

[23] M. Ley. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. *SPIRE*, pages 1–10, 2002.

[24] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *arXiv.org*, (6):066133, Sept. 2003.

[25] M. E. J. Newman. Modularity and community structure in networks. *arXiv.org*, (23):8577–8582, Feb. 2006.

[26] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos. ParCube: Sparse Parallelizable Tensor Decompositions. *ECML/PKDD*, 7523(Chapter 39):521–536, 2012.

[27] E. E. Papalexakis, N. Sidiropoulos, and R. Bro. From k-means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *Signal Processing, IEEE Transactions on*, 61(2):493–506, 2013.

[28] T. P. Peixoto. Inferring the mesoscale structure of layered, edge-valued and time-varying networks. *arXiv.org*, (4):042807, Apr. 2015.

[29] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey. Shaping Communities out of Triangles. *arXiv.org*, July 2012.

[30] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *the 23rd international conference*, pages 225–236, New York, New York, USA, 2014. ACM Press.

[31] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. TimeCrunch: Interpretable Dynamic Graph Summarization. *KDD*, pages 1055–1064, 2015.

[32] X. Tang and C. C. Yang. *Dynamic Community Detection with Temporal Dirichlet Process*. IEEE, 2011.

[33] K. S. Xu and A. O. Hero. Dynamic Stochastic Blockmodels: Statistical Models for Time-Evolving Networks. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 201–210. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[34] J. Yang and J. Leskovec. Defining and Evaluating Network Communities Based on Ground-Truth. *CoRR abs/1204.6078*, cs.SI:745–754, 2012.

[35] T. Yang, Y. Chi, S. Zhu, Y. Gong, and R. Jin. Detecting communities and their evolutions in dynamic social networks–a Bayesian approach. *Machine Learning*, 82(2):157–189, Feb. 2011.

[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. *HotCloud 2010*, 2010.

APPENDIX

A. RANK-1 DECOMPOSITION UPDATES

PROOF. Take updating $\mathbf{a}$ as an example. We want to solve the following optimization problem

$$\min_{\mathbf{a}} \sum_i \sum_j \sum_k (x_{i,j,k} - \lambda \cdot \mathbf{a}_i \cdot \mathbf{b}_j \cdot \mathbf{c}_k)^2 \tag{7}$$

and achieve that by finding the value of $\mathbf{a}$ that sets the gradient to 0.

$$\frac{\partial Er}{\partial \mathbf{a}_i} = 0 \tag{8}$$

$$= \sum_j \sum_k 2 \cdot (x_{i,j,k} - \lambda \cdot \mathbf{a}_i \cdot \mathbf{b}_j \cdot \mathbf{c}_k)(\lambda \cdot \mathbf{b}_j \cdot \mathbf{c}_k) \tag{9}$$

$$\mathbf{a}_i = \frac{\sum_j \sum_k \mathbf{b}_j \cdot \mathbf{c}_k \cdot x_{i,j,k}}{\sum_j \sum_k \lambda \cdot \mathbf{b}_j^2 \cdot \mathbf{c}_k^2} \tag{10}$$

Note that the denominator of Equation 10 is constant for all i. Furthermore, as $\mathbf{a}$ is a unit vector, it follows that the denominator must be the normalization constant Z_a. □