# Communities and Anomaly Detection in Large Edge-Labeled Graphs

Miguel Araujo

CMU-CS-17-110

May 2017

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Computer Science Department
Faculty of Sciences
University of Porto
4169-007 Porto, Portugal

**Thesis Committee:**
Christos Faloutsos, co-chair
William Cohen
Aarti Singh
Pedro Ribeiro, co-chair, University of Porto
Tina Eliassi-Rad, Northeastern University
Beatriz Santos, University of Aveiro
Alexandre Francisco, University of Lisbon

*Submitted in partial fulfillment of the requirements*
*for the degree of Doctor of Philosophy.*

This research was sponsored by "Fundação para a Ciência e Tecnologia" (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program under grant BD/52362/2013 and the Information and Communication Technology Institute at Carnegie Mellon University. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

**Keywords:** Graph Mining, Time-Evolving Networks, Community Detection, Anomaly Detection, Tensor Factorization

*To my loving family.*

# Abstract

The identification of anomalies and communities of nodes in real-world graphs has applications in widespread domains, from the automatic categorization of wikipedia articles or websites to bank fraud detection. While recent and ongoing research is supplying tools for the analysis of simple unlabeled data, it is still a challenge to find patterns and anomalies in large labeled datasets such as time evolving networks. What do real communities identified in big networks look like? How can we find sources of infections in bipartite networks? Can we predict who is most likely to join an online discussion on a given topic?

We model interaction networks using appropriate matrix and tensor representations in order to gain insights into these questions. We incorporate edge attributes, such as timestamps in phone-call networks or airline codes in flight networks, and complex node side-information, such as who-retweets-whom in order to understand who uses a given hashtag on Twitter. We provide three major contributions:

1. *Hyperbolic communities:* Our exploration of real communities provides novel theoretical ideas regarding their structure, and we show how typical long-tail degree distributions can be leveraged to create efficient algorithms on problems that seem to suffer from quadratic explosion.
2. *Anomaly detection:* Novel distributed algorithms that identify problematic nodes whose influence can only be detected on their neighbors, validated through the analysis of data breaches in bank transactions.
3. *Forecasting:* New techniques that forecast network evolution by incorporating contextual side-information and the evolution of independent community structures.

Leveraging these techniques, we explore massive datasets such as networks with *billions* of credit card transactions, Twitter graphs with over *300 million* interactions and phone-call networks with over *50 million* phone-calls.

# Resumo

A identificação de anomalias e comunidades em grafos tem aplicações em variados domínios, desde a categorização automática de artigos na Wikipedia à detecção de fraude bancária. Apesar de investigação recente fornecer ferramentas para a análise de dados não-anotados, ainda é um desafio encontrar padrões e anomalias em grandes volumes de dados anotados, como redes temporais. Qual é a forma das comunidades identificadas em redes de grande dimensão? Como podemos encontrar a fonte de infecções em grafos bipartidos? Conseguimos prever que utilizador é mais provável juntar-se a uma discussão online num determinado tópico?

Obtemos resposta a estas perguntas através da modelação de redes de interação recorrendo a matrizes e tensores. Incorporamos atributos das arestas, como a data e hora de chamadas em redes telefónicas ou códigos de companhias aéreas em redes de voos, e informação auxiliar relativa aos nós, tal como quem-*retweeta*-quem para compreender quem usar uma determinada *hashtag* no Twitter. Apresentamos três contribuições principais:

1. *Comunidades hiperbólicas:* A exploração de comunidades reais fornece novas ideias teóricas em relação à sua estrutura, e mostramos como as típicas distribuições *heavy-tail* do grau dos nós podem ser usadas para criar algoritmos eficientes para problemas que tipicamente sofrem de explosão quadrática na sua computação.
2. *Detecção de anomalias:* Novos algoritmos distribuídos que identificam nós problemáticos cuja influência só pode ser detectada nos vizinhos, validados através da análise de falhas de segurança em transacções bancárias.
3. *Previsão:* Novas técnicas para a previsão da evolução de redes, incorporando informação contextual e a evolução de várias comunidades de forma independente.

Recorrendo a estas técnicas, explorando conjuntos de dados massivos como redes com *milhares de milhões* de transacções de cartões de crédito, grafos do Twitter com mais de 300 *milhões* de interacções e redes de chamadas telefónicas com mais de 50 *milhões* de chamadas.

# Acknowledgements

This 5-years effort would not have been possible without all the strong people in my life pushing me ahead. Their support, influence and inspiration is part of this thesis and makes it as much theirs as it is mine. The role my advisors played will shape my personality and my life going forward. I want to thank Christos Faloutsos, who is simultaneously the happiest and most hardworking man I know. His advice and pertinent questions continued regardless of physical proximity. He was never too busy, even if that meant short-notice skyping on weekends or a midnight push for a deadline. Pedro's guidance was different, less formal. He made sure to set up the right opportunities, the summer internship, the connection with students, the teaching. Finally, I consider Prof. Fernando Silva my unofficial advisor. He is the person we look up to and his experience improves those around him.

I would like to thank the members of my committee, Aarti Singh, William Cohen, Tina Eliassi-Rad, Beatriz Santos and Alexandre Francisco. I appreciate the meaningful feedback you gave me and the insightful questions during my thesis proposal. In particular, I would like to thank Alexandre for giving me the opportunity of showcasing my work to his students in Lisbon.

A relevant part of this dissertation is based on work I did during my internship at Feedzai. Many companies would not be willing to share their methods and results as openly as they were, so I'm particularly grateful to Pedro Bizarro for that opportunity.

A fundamental part of graduate school are the bonds you create when you are not doing research. I want to thank Danai Koutra, Vagelis Papalexakis and Alex Beutel for showing me the way. I still look up to you. Also, all the younger members of the databases group: Jay-Yoon Lee (thanks for introducing me to Korean barbecue!), Neil Shah, Dhivya Eswaran (thanks for that peculiar skype call!), Hyun-Ah Song, Bryan Hooi and Kijung Shin. Also, the Portuguese gang in Pittsburgh: João and Diana, who lent me their couch for more than one night, Luis, Hugo Pinto, Hugo Gonçalves, João Semedo and Bruno Vavala, the undercover Italian. In Portugal, the 1.18 lunch crew and soccer team: David, Pedro Paredes, Miguel and Catarina, Pedro Ferreira, Chris and Sadiq. Our lunch-time discussions are now countless and invaluable.

All the administrative support I got from Marilyn Walgora and Deb Cavlovich in Pittsburgh still amazes me. How can you know all the answers? In Portugal, Alexandra and Isabel have probably booked me more meeting rooms than for many professors who have been at the department for decades.

Many others have been directly or indirectly responsible and without whom this dissertation would not have been possible: San-Chuan Hung, Rosaldo Rossetti, Stephan

Günnemann, Gonzalo Mateos, Spiros Papadimitriou, Prithwish Basu, Ananthram Swami, Miguel Almeida, João Oliveirinha, Jaime Ferreira, Luis Silva.

Finally, the most important. My family supported me beyond what many would believe possible. Joana. Who else would travel across the ocean to be my side when it mattered the most? My parents and my brother, always there, eager to hear the latest news, answer my skype calls and celebrate my successes. My grandparents. I love you even when you call me in the middle of the night, not realizing I'm in a different time zone. You believe I'm the best, even though you don't understand a thing I do. Lastly, my second family: my parents-in-law. You treat me as your own and I shall only respond in kind.

Nobody knows what the future holds, but success is all but guaranteed when so many people look after you.

# Contents

**1 Introduction** **1**

1.1 Part I: Networks and Matrices . . . 2

1.2 Part II: Labeled Networks and Tensors . . . 5

1.3 Overall Impact . . . 7

**2 Background** **9**

2.1 Networks . . . 9

2.1.1 Common Definitions . . . 9

2.1.2 Properties . . . 10

2.1.3 Sampling . . . 11

2.2 Learning and Other Techniques . . . 12

2.2.1 Factorizations and Objective Functions . . . 12

2.2.2 Bayesian Models . . . 13

2.2.3 Minimum Description Length (MDL) . . . 13

2.3 Matrices and Spectral Methods . . . 14

2.3.1 Singular Value Decomposition . . . 15

2.3.2 Non-Negative Matrix Factorization . . . 15

2.3.3 Co-clustering . . . 16

2.4 Tensors . . . 16

2.4.1 Tensor Operations . . . 17

2.4.2 Tensor Factorizations . . . 18

2.4.3 Rank-1 Decompositions . . . 19

2.4.4 Coupled Factorizations . . . 20

2.5 Notation and Common Symbols . . . 22

**I Networks and Matrices** **23**

I Related Work: Community Detection . . . 25

I.1 Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
I.2 Applications . . . . . . . . . . . . . . . . . . . . . . . . . 30
I.3 Challenges: No good cuts . . . . . . . . . . . . . . . . . . . 30
II Related Work: Decomposition of Boolean Matrices . . . . . . . . . . . . . 32

3 Hyperbolic Communities 33
3.1 Empirical Observations . . . . . . . . . . . . . . . . . . . . . 34
3.2 Hyperbolic Community Model . . . . . . . . . . . . . . . . . . . 37
3.2.1 Community Definition . . . . . . . . . . . . . . . . . . . 37
3.2.2 MDL Description Cost. . . . . . . . . . . . . . . . . . . . 38
3.3 Proposed Method: HyCoM-FIT . . . . . . . . . . . . . . . . . . 40
3.3.1 Fast MDL calculation . . . . . . . . . . . . . . . . . . . 43
3.3.2 Complexity analysis . . . . . . . . . . . . . . . . . . . . 45
3.4 Experiments on Real Data . . . . . . . . . . . . . . . . . . . . 45
3.4.1 Q1 - Model quality . . . . . . . . . . . . . . . . . . . . . 45
3.4.2 Q2 - Scalability . . . . . . . . . . . . . . . . . . . . . . 46
3.4.3 Q3 - Effectiveness . . . . . . . . . . . . . . . . . . . . . 47
3.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

4 Scalable Boolean Matrix Factorization 51
4.1 Problem Definition . . . . . . . . . . . . . . . . . . . . . . . 53
4.1.1 Formal Objective . . . . . . . . . . . . . . . . . . . . . . 53
4.1.2 Step Matrix Decomposition . . . . . . . . . . . . . . . . . 54
4.2 Naive Approach: FastStepNaive . . . . . . . . . . . . . . . . . 54
4.2.1 Stopping Criteria . . . . . . . . . . . . . . . . . . . . . . 55
4.2.2 Complexity . . . . . . . . . . . . . . . . . . . . . . . . . 57
4.3 Proposed Method: FastStep . . . . . . . . . . . . . . . . . . . 57
4.3.1 Fast Gradient Calculation . . . . . . . . . . . . . . . . . 57
4.3.2 Fast Error Function Evaluation . . . . . . . . . . . . . . 58
4.3.3 Complexity . . . . . . . . . . . . . . . . . . . . . . . . . 59
4.3.4 Obtaining clusters from A and B . . . . . . . . . . . . . . 60
4.4 Experimental Evaluation . . . . . . . . . . . . . . . . . . . . 60
4.4.1 Q1 - Scalability . . . . . . . . . . . . . . . . . . . . . . 61
4.4.2 Q2 - Low Reconstruction Error . . . . . . . . . . . . . . . 61
4.4.3 Q3 - Discoveries . . . . . . . . . . . . . . . . . . . . . . 63
4.5 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . 65
4.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

5 Detecting Points-of-Compromise 67
5.1 Problem Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
5.2 POC-detection Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . 70
5.2.1 A POC Hierarchical Model . . . . . . . . . . . . . . . . . . . . 71
5.2.2 From Blames to POC Probabilities . . . . . . . . . . . . . . . . 71
5.2.3 From POC Probabilities to Blames . . . . . . . . . . . . . . . . 73
5.2.4 An Alternating Algorithm . . . . . . . . . . . . . . . . . . . . 73
5.2.5 Convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
5.3 Distributed POC-detection . . . . . . . . . . . . . . . . . . . . . . . . . 75
5.4 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
5.4.1 Experimental Setup . . . . . . . . . . . . . . . . . . . . . . . . 78
5.4.2 Empirical Evidence and Fraud Prevented . . . . . . . . . . . . . 78
5.4.3 Accuracy and Early Detection . . . . . . . . . . . . . . . . . . . 79
5.4.4 Scalability . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
5.4.5 Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
5.5 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
5.5.1 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
5.5.2 Real-time Fraud Detection . . . . . . . . . . . . . . . . . . . . 83
5.5.3 Points-of-Compromise . . . . . . . . . . . . . . . . . . . . . . . 83
5.5.4 Guilt-by-association . . . . . . . . . . . . . . . . . . . . . . . . 84
5.5.5 Vertex Cover . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
5.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85

II Labeled Networks and Tensors 87
I Related Work: Communities in Edge-labeled Networks . . . . . . . . . . . 89
I.1 Categorical Edge-labels . . . . . . . . . . . . . . . . . . . . . . . . 89
I.2 Time-evolving Networks . . . . . . . . . . . . . . . . . . . . . . . . 90

6 Communities in Labeled Networks 93
6.1 Problem Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95
6.2 Algorithmic solution . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
6.2.1 Community candidates . . . . . . . . . . . . . . . . . . . . . . . 98
6.2.2 Community construction . . . . . . . . . . . . . . . . . . . . . . 99
6.2.3 Tensor deflation . . . . . . . . . . . . . . . . . . . . . . . . . . . 101
6.2.4 Complexity Analysis . . . . . . . . . . . . . . . . . . . . . . . . 102
6.2.5 Algorithm parameters . . . . . . . . . . . . . . . . . . . . . . . 102
6.3 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103

6.3.1 Q1 - Community Structure . . . . . . . . . . . . . . . . . . . . . . . 103
6.3.2 Q2 - Scalability . . . . . . . . . . . . . . . . . . . . . . . . . . . . 105
6.3.3 Q3 - Discoveries on edge-labeled graphs . . . . . . . . . . . . . . 105
6.3.4 Q3 - Discoveries on time-labeled graphs . . . . . . . . . . . . . . . 106
6.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110

**7 Distributed Community Detection 113**
7.1 Proposed System . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 114
7.1.1 Factorization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
7.1.2 Thresholding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
7.1.3 Tensor Deflation . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
7.1.4 Implementation Design . . . . . . . . . . . . . . . . . . . . . . . . 117
7.2 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
7.2.1 Q1 - Precision . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
7.2.2 Scalability . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120
7.2.3 Q4 - Discoveries . . . . . . . . . . . . . . . . . . . . . . . . . . . 124
7.3 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125
7.4 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126

**8 Forecasting Communities 127**
8.1 Proposed Method: TensorCast . . . . . . . . . . . . . . . . . . . . . . . 128
8.1.1 Overview. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 129
8.1.2 Non-negative Coupled Factorization . . . . . . . . . . . . . . . . . 130
8.1.3 Forecasting . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131
8.1.4 Tensor Top-K elements . . . . . . . . . . . . . . . . . . . . . . . . 131
8.1.5 Complexity Analysis. . . . . . . . . . . . . . . . . . . . . . . . . . 134
8.2 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135
8.2.1 Q1 - Scalability . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136
8.2.2 Q2 - Effectiveness and Context-awareness . . . . . . . . . . . . . . 136
8.2.3 Q3 - Trend Following . . . . . . . . . . . . . . . . . . . . . . . . . 136
8.2.4 Q4 - Precision over Time . . . . . . . . . . . . . . . . . . . . . . . 138
8.2.5 Discoveries - TensorCast at work . . . . . . . . . . . . . . . . . . 138
8.3 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
8.3.1 Top-K elements in Matrix Products . . . . . . . . . . . . . . . . . 139
8.3.2 Power-laws as building blocks . . . . . . . . . . . . . . . . . . . . 140
8.3.3 Link Prediction . . . . . . . . . . . . . . . . . . . . . . . . . . . . 140
8.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 141

III Conclusions and Future Directions 143

9 Conclusions 145

9.1 Networks and Matrices . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146

9.2 Labeled Networks and Tensors . . . . . . . . . . . . . . . . . . . . . . . . 147

9.3 Overall Impact . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147

10 Vision and Future Directions 149

10.1 Systems: “Database Factorizations” . . . . . . . . . . . . . . . . . . . . . 149

10.2 Theory: Adversarial Anomalies . . . . . . . . . . . . . . . . . . . . . . . 150

Appendices

A BreachRadar - Additional details 153

A.1 Fraud Label Delays . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153

A.2 Multiple Points-of-Compromise . . . . . . . . . . . . . . . . . . . . . . . 153

Bibliography 155

# List of Figures

1.1 FASTSTEP agrees with intuition: community of world airports. . . . . . . . . 3
1.2 BREACHRADAR estimated savings: $2M USD in a 6-weeks period. . . . . . 4
1.3 TENSORCAST's precision when forecasting $(author, venue)$ relations in the DBLP tensor. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.1 An example on minimum description length balance. On the left, a big model without errors. On the right, a model with many errors to be encoded. 14
2.2 A three-mode tensor of Authors, Keywords and Venues. . . . . . . . . . . . . 16
2.3 A simple Coupled Matrix-Tensor Factorization. . . . . . . . . . . . . . . . . . . 20
2.4 Adjacency matrices of classical community definitions. . . . . . . . . . . . . 26
2.5 Algorithm characteristics: methods can be classified by their ability to find overlapping or hierarchical communities. . . . . . . . . . . . . . . . . . . . . 27
2.6 Conductance of ground-truth communities. . . . . . . . . . . . . . . . . . . . 31

3.1 Side-by-side: comparing a ground-truth community and one found by HYCOM-FIT. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.2 Big communities are sparse: number of nodes vs density in the DBLP dataset. 35
3.3 Big ground-truth communities are hyperbolic: community size vs $\alpha$. . . . . 36
3.4 Adjacency Matrix of a synthetic Hyperbolic Community. . . . . . . . . . . 38
3.5 HYCOM requires less bits to encode ground-truth communities. . . . . . . 46
3.6 HYCOM-FIT scales linearly with the number of edges. . . . . . . . . . . . . 47
3.7 Example of anomalous community found by HYCOM-FIT. . . . . . . . . . . 47
3.8 HYCOM-FIT finds meaningful hyperbolic structures. . . . . . . . . . . . . . . 48
3.9 HYCOM-FIT finds bipartite cores and cliques. . . . . . . . . . . . . . . . . . . 48

4.1 FASTSTEP finds realistic hyperbolic communities. . . . . . . . . . . . . . . . 52
4.2 FASTSTEP correctly split the airports of the world by geographic region. . . 52
4.3 FASTSTEP's error function. . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
4.4 A small number of non-zeros approximates the gradient. . . . . . . . . . . . 58
4.5 A small number of samples approximates the error. . . . . . . . . . . . . . . 59

4.6 Scalability: FASTSTEP has a running time linear on the number of non-zeros. 61
4.7 Competitors need to compute the threshold and naively require quadratic time. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
4.8 FASTSTEP successfully groups movies by genre. . . . . . . . . . . . . . . . 63
4.9 European airports are intuitively split between major and secondary. . . . 64
4.10 Communities found follow a power-law distribution. . . . . . . . . . . . . 65

5.1 BREACHRADAR's effectiveness and comparison to other methods. . . . . . 68
5.2 Example of a bipartite network used as input to BREACHRADAR. . . . . . . 69
5.3 Plate notation of the probabilistic graphical model. . . . . . . . . . . . . . 72
5.4 Impact of the prior on the point-of-compromise probability. . . . . . . . . 73
5.5 BREACHRADAR converges exponentially fast. . . . . . . . . . . . . . . . . 75
5.6 BREACHRADAR's prior. . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
5.7 BREACHRADAR's accuracy with varying probability of fraud. . . . . . . . . 80
5.8 BREACHRADAR's speed-up and scalability. . . . . . . . . . . . . . . . . . 81

6.1 COM$^2$ detects competing airlines. . . . . . . . . . . . . . . . . . . . . . 94
6.2 COM$^2$: synthetic datasets used. . . . . . . . . . . . . . . . . . . . . . . 104
6.3 Experiments on synthetic data. . . . . . . . . . . . . . . . . . . . . . . . 105
6.4 Regional communities of competing companies found using flight routes. . 107
6.5 Community in Europe. . . . . . . . . . . . . . . . . . . . . . . . . . . . 108
6.6 Communities activity: weekly periodicity . . . . . . . . . . . . . . . . . . 110
6.7 LBNL community: Com$^2$ detects research group collaborations. . . . . . 111

7.1 Communities detected in flights dataset. . . . . . . . . . . . . . . . . . . 114
7.2 TeraCom: Overview. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
7.3 Data scalability experiment. . . . . . . . . . . . . . . . . . . . . . . . . 121
7.4 Execution time vs outer-iterations. . . . . . . . . . . . . . . . . . . . . . 122
7.5 Machine scalability. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123
7.6 Two clusters of different European airlines. . . . . . . . . . . . . . . . . 124

8.1 TENSORCAST's effectiveness and scalability. . . . . . . . . . . . . . . . . 128
8.2 Overview of TENSORCAST. . . . . . . . . . . . . . . . . . . . . . . . . . 130
8.3 TENSORCAST only checks a linear number of elements of the tensor. . . . 134
8.4 Precision when forecasting novel relations in the TWITTER tensor. . . . . 137
8.5 Forecasting growth and decay. . . . . . . . . . . . . . . . . . . . . . . . 137
8.6 TENSORCAST achieves higher precision at every forecasting horizon. . . . 138
8.7 Groups with similar interest on TWITTER. . . . . . . . . . . . . . . . . . 139
8.8 The evolution and forecast of two example groups. . . . . . . . . . . . . 140

A.1 BreachRadar: label delays over time. . . . . . . . . . . . . . . . . . . . . . 154

# List of Tables

1.1 Thesis Overview. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2.1 Symbols and Notation used throughout this dissertation. . . . . . . . . . . 22

3.1 HYCOM: Summary of real-world networks used. . . . . . . . . . . . . . . 35

4.1 Datasets used to evaluate FASTSTEP. . . . . . . . . . . . . . . . . . . . . . 60
4.2 Number of non-zeros in segments of the `MovieLens10m` dataset. . . . . . 61
4.3 FASTSTEP: comparison of the squared error. . . . . . . . . . . . . . . . . . 62
4.4 Automatic clustering of movies. . . . . . . . . . . . . . . . . . . . . . . . 63
4.5 FASTSTEP: Comparison of decomposition methods. . . . . . . . . . . . . . 66

5.1 Notation, symbols and definitions . . . . . . . . . . . . . . . . . . . . . . 70
5.2 Several Points-of-Compromise identified in one of the datasets have also been mentioned in news reports. . . . . . . . . . . . . . . . . . . . . . . 77
5.3 Overview of the two datasets used. Specific values masked for privacy. . . 78
5.4 Impact of the infection probability on the number of *fraud-cards* and possible Points-of-Compromise. . . . . . . . . . . . . . . . . . . . . . . . . . 81
5.5 Comparison of BREACHRADAR with other methods. Properties are described in section 5.5. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83

6.1 Networks used: two small synthetic networks and three large real networks.103
6.2 Comparison of community detection methods. . . . . . . . . . . . . . . . . 111

7.1 Dataset description: 2 synthetic and 3 real-world datasets. . . . . . . . . . 119
7.2 Precision Experiment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120

8.1 Summary of real-world networks used. . . . . . . . . . . . . . . . . . . . . 135
8.2 TENSORCAST integrates context and time-awareness. . . . . . . . . . . . . 142

# List of Algorithms

1 HyCoM-FIT: Community Construction . . . . . . . . . . . . . . . . . . . . . 42

2 FastStep: Gradient Descent . . . . . . . . . . . . . . . . . . . . . . . . . 56

3 BreachRadar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74

4 Distributed BreachRadar . . . . . . . . . . . . . . . . . . . . . . . . . . 76

5 Com$^2$: Community Construction . . . . . . . . . . . . . . . . . . . . . . . 100

6 TeraCom . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115

7 TensorCast: Top-K Elements . . . . . . . . . . . . . . . . . . . . . . . . 132

# Chapter 1

# Introduction

Graphs are widely used representations for modeling relationships between objects, both concrete and abstract: they model cities and roads, people and their relations, neurons and their synapses and many other interesting phenomena. These relationships aren't random. In real networks, nodes naturally organize into communities exhibiting a degree of cohesiveness, as reported not only in social graphs [WF94b] but also in the World Wide Web [FLG00] and in protein-protein interaction networks [SKJ06]. Anomaly detection in these big graphs often involves the identification of common patterns and the detection of outliers, i.e. situations in which these patterns don't hold. In fact, some of the most interesting patterns and outliers relate to the connections between entities, rather than to their intrinsic properties: dense bipartite cores in user-reviews graphs might indicate fraud, and dense bipartite cores in a social graph might indicate that some people are buying followers. On the other hand, dense subgraphs in protein-protein interaction networks aid in metabolic function identification, while in networks such as Wikipedia they tend to represent articles on the same subject. As they help us describe so many rich scenarios, the detection of domain-specific patterns (e.g., communities) and anomalies helps us understand, explain and control these complex domains.

As our ability to collect data improves and as graphs grow bigger, making sense and understanding these big structures requires new methods and tools that help us extract useful information. In fact, even describing these structures is currently a challenge. What is the typical structure of groups of nodes we consider useful? What do they look like and how can we find them? How can we detect anomalous nodes in power-law networks, such as attackers or sources of infection, when only some of their neighbors show signs of their presence? How do we do any of this when we take into consideration that our networks are dynamic and time-evolving? What time-evolving structures can we find? Can we use this evolution to predict stable relations or to forecast novel interactions?

This work has two main directions: a) the focus on groups and communities and their detection in static and time-evolving networks, using interpretable and scalable methods; and b) anomaly detection and the forecasting of novel relations. We provide new insights and techniques on both areas with impact on real-world applications.

This thesis is organized in two main parts: 1) networks and matrices and; 2) labeled networks and tensors. We summarize the main problems of each part in the form of questions in Table 1.1.

**Table 1.1: Thesis Overview.**

| Part | Research Problem | Chapter |
|---|---|---|
| **I: Networks and Matrices** | **Community Structure:** What is the structure of real-world communities? How can we find them? | 3, 4 |
| | **Points-of-Compromise:** Can we detect sources of data breaches or infections in billion-sized datasets? | 5 |
| **II: Labeled Networks and Tensors** | **Temporal Communities:** What are typical patterns of temporal communities? How can we find them? | 6 |
| | **Forecasting Relationships:** Can we forecast novel relationships when context is available? | 8 |

## 1.1 Part I: Networks and Matrices

We analyze local structures and communities in an effort to better understand individual nodes in a network. What patterns do meaningful groups of nodes follow in static networks? What should we be looking for when we need to find similarly important groups in different graphs? Just as importantly: how can we find anomalous nodes whose influence can only be detected on their neighbors? Our work identifies similarities that important groups share across a variety of domains. We propose scalable methods for their identification and show how specific malicious nodes can be found when we can only detect symptoms on their neighbors.

## Community Structure

*“What is the structure of real-world communities?”*
*“How can we find them?”*

One way of understanding the structure of real-world communities is to analyze their common shape and properties over different networks (Chapter 3). Unlike previously assumed, through the analysis of ground-truth communities we show that meaningful structures do not exhibit near-clique behavior. We propose HyCoM - the Hyperbolic Community Model [AGMF14] - which aims for a more realistic representation of typically found communities.

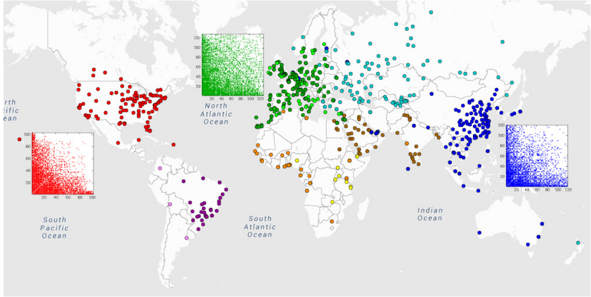


**Figure 1.1: FastStep agrees with intuition: communities of world airports.**

We formalize this problem and present two distinct approaches that identify communities with such structure. Firstly, we pose the problem according to the Minimum Description Length (MDL) principle: good structures should minimize the compression length of the graph (Chapter 3). We then propose FastStep [ARF16], a factorization-based approach which incorporates a binary reconstruction of the data. We show that hyperbolic communities emerge naturally from this procedure, without ever being imposed (Chapter 4).

**Contributions:**

- *Modeling*: We provide empirical evidence that communities in large real-world networks are better modeled using an Hyperbolic Model.
- *Scalability*: FastStep shows that efficient optimizations can be performed by taking into consideration the natural heavy tails of the factorization vectors.
- *Effectiveness*: We show meaningful clusters of movies and users when applying FastStep to a movie ratings dataset with over *10 million* non-zeros.

**Impact:**

- FastStep [ARF16] is the basis of Miguel Duarte’s currently ongoing Masters thesis at the University of Lisbon.

## Points-of-Compromise

*"Can we detect sources of data breaches or infections in billion-sized datasets?"*

In many applications, an anomalous node might only be detectable through the analysis of its neighborhood. For instance, detecting infected files through the analysis of file-machine bipartite graphs and malware symptoms, or detecting a food poisoning source through the analysis of hospital records and who-ate-where data. We call this problem the detection of *Points-of-Compromise*, as some nodes of the graph are able to compromise their neighbors (Chapter 5).

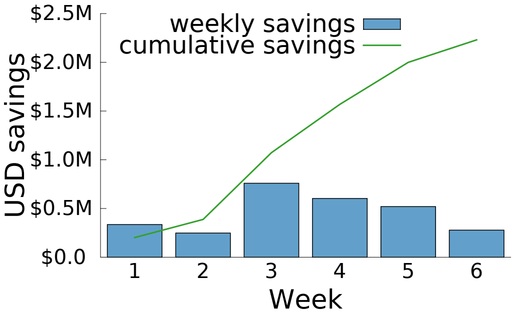


**Figure 1.2: BREACHRADAR estimated savings (2M USD in a 6-weeks period).**

We study a dataset with over a *billion* bank transactions in order to detect data breaches and skimming devices through the analysis of clients who have been victims of fraud. BREACHRADAR [AAF+17] is a distributed algorithm that alternatingly optimizes a graphical model that predicts the probability of a location being the source of a compromise. We show that preemptively reissuing credit cards who have interacted with high probability locations can prevent losses of *millions of dollars* in real fraud.

**Contributions:**

- *Methodology*: We formulate a novel an important Point-of-Compromise detection problem.
- *Scalability:* We designed BREACHRADAR, a distributed algorithm that can be applied to *billion*-sized datasets.
- *Effectiveness*: BREACHRADAR is able to identify Points-of-Compromise with over 90% accuracy when only 10% of the stolen cards have been used in fraud.

**Impact:**

- This work resulted in a worldwide **patent** submission.
- BREACHRADAR is used to detect data breaches and Points-of-Compromise from bank transactions at Feedzai, a fraud and risk detection company.

## 1.2 Part II: Labeled Networks and Tensors

While static networks can represent a myriad of interactions, some problems require the analysis of information that can only be modeled as labels or context of the connections. How do communities evolve over time? What are the typical patterns that they create and how can we find them? As we analyze their growth and death and forecast their evolution, can we predict which connections are more likely to be created? We propose principled methods for finding temporal communities and for the forecasting problem in temporal graphs with contextual side information.

### Temporal Communities

*"What are typical patterns of temporal communities?"*
*"How can we find them?"*

Consider an edge-labeled network, where labels might represent colors, airlines or days of the week. How can we understand its underlying structure? Are there common occurrences and patterns that we should look for? What are the most meaningful communities in a network where edges are labeled? Making sense of a static graph with *millions* of nodes is often difficult due visualization or even memory constraints; how can we understand the structure of edge-labeled graphs (Chapter 6)?

A simple approach would group together nodes that tend to be connected through a subset of the edge labels. We approach the problem from an information theory perspective and present a Minimum Description Length (MDL) approach whose goal is to minimize the total description length (i.e., number of bits) of the labeled network. Com$^2$ [APG+14] is an effective and scalable algorithm that can be applied to any network with categorical edge labels.

**Contributions:**

- *Principled formulation*: We formulate a MDL problem and a scalable algorithm that doesn't require any user-defined parameters.
- *Effectiveness*: The analysis of a communications dataset with over 50 *million* non-zeros shows its effectiveness on finding common patterns.

**Impact:**

- COM$^2$ was selected as one of the best papers at PAKDD'14 [APG+14].
- COM$^2$ appeared in a special issue of the Knowledge and Information Systems journal [AGP+15].

## Forecasting Relationships

*"Can we forecast novel relationships when context is available?"*

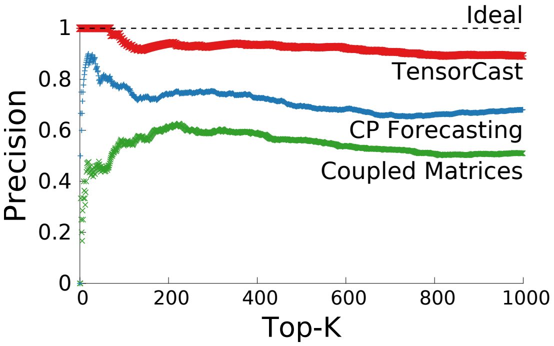


**Figure 1.3: TENSORCAST's precision when forecasting $(author, venue)$ relations in the DBLP tensor.**

Link prediction is a well-studied problem in social networks. However, most methods only consider networks where new links are added and are unable to analyze the networks' evolution. Forecasting relations in a network that we know is time-evolving is important for analyzing churn, designing marketing campaigns and to produce better recommendations. We study the problem of forecasting a network when side-information about the nodes is available, such as demographics or out-of-network connections. Can we predict who is likely to join a political discussion on Twitter or to publish on a given conference next year (Chapter 8)?

We formalize this problem as forecasting a set of coupled tensor and introduce TENSORCAST, a method that 1) factorizes all the input tensors, 2) forecasts the tensor of interest and 3) identifies the most relevant new elements of the forecasted network. Under common network assumptions (i.e., long tails), we show that TENSORCAST doesn't suffer from the quadratic explosion problem and analyze datasets with over *300 million* relations.

**Contributions:**

- *Contextual-awareness Forecasting*: we show how different data sources can be included in a principled way.
- *Scalable Tensor Top-K*: TENSORCAST is able to quickly find the $K$ biggest elements in a tensor factorization under realistic assumptions.
- *Effectiveness*: TENSORCAST achieves over 20% higher precision in top-1000 queries and double the precision when finding new relations than comparable alternatives.

## 1.3 Overall Impact

My thesis work is focused on the development of fast and effective algorithms for the detection of communities and anomalies in multiple settings and applications. We contribute in the cross product of two common themes: on the one hand, we focus on patterns and anomalies, while on the other, we study both static and labeled networks, usually temporal. Hence, this work has broad impact on several domains: recommendation systems, fraud detection, contextual forecasting and graph understanding, and has been used in multiple settings:

- **Impact in industry:** Feedzai detects data breaches and other Points-of-Compromise using BreachRadar [AAF+17], a work which has a patent submitted and early estimates indicate savings of $1M per month.
- **Impact in academia:** At the University of Lisbon, FastStep is the basis of a Masters thesis dissertation.
- **Awards:** Com$^2$ [APG+14] received one of the best paper awards at PAKDD'14.

In the next chapter, we provide a brief overview of the background in this area, introducing useful theoretical notions that aid the understanding of the rest of the dissertation.

# Chapter 2

# Background

We start by introducing the most relevant concepts, definitions and notation in graph theory and linear algebra that are used throughout this dissertation. Please refer to Table 2.1 at the end of this chapter for an easy-to-reference list of operators and symbols.

As common in the literature, we denote vectors by boldface lowercase letters (e.g., $\boldsymbol{v}$), matrices by boldface uppercase letters (e.g., $\boldsymbol{A}$) and tensors by boldface calligraphic letters (e.g., $\boldsymbol{\mathcal{X}}$). For convenience, we refer to the $f$-th column of $\boldsymbol{A}$ as $\boldsymbol{a_f}$ and to the $(i,j,k)$ entry of 3-mode tensor $\boldsymbol{\mathcal{X}}$ as $\boldsymbol{\mathcal{X}}_{ijk}$. Sets and other similar entities are denoted by non-bold calligraphic letters (e.g., $\mathcal{V}$, the set of all nodes).

## 2.1 Networks

### 2.1.1 Common Definitions

**Graph.** A mathematical representation of a set of objects and their relations. We denote a graph $\mathcal{G}$ as an ordered pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ represents the set of objects (also called nodes or vertices) and $\mathcal{E}$ represents the set of relations (also called edges, links or connections).

**Node or vertex.** Together with edges, vertices form the fundamental unities of a graph. Each node corresponds to an object in the finite set $\mathcal{V}$, while $|\mathcal{V}|$ or $N$ will typically represent the size of this set.

**Edge or connection.** Edges represent the relationships between the nodes of a network. The set of all relations is represented by $\mathcal{E}$, while $|\mathcal{E}|$ or $E$ will typically represent the size of this set.

**Degree of a node.** The degree of node $v$, $d_v$, corresponds to the number of relationships in which node $v$ participates. In the case of a directed graph, it can can be divided in $d_v^{in}$ and $d_v^{out}$, the *in-degree* and *out-degree*, respectively.

**Edge-Labeled Graph.** A graph in which edges have *labels* associated. We will restrict this definition to *categorical* edge labels; we call a graph with *numerical* edge labels a **weighted graph**. We represent the set of possible labels by $\mathcal{L}$.

**Directed Graph.** A graph where each edge has a direction. We can represent a directed edge $e$ as an ordered pair $(u, v)$, indicating that edge $e$ establishes a connection from node $v$ to node $u$.

**Bipartite Graph.** A graph where nodes can be split into two disjoint sets ($\mathcal{U}$ and $\mathcal{V}$) and where no connections exist between nodes of the same group. In this scenario, nodes typically represent different entities, such as *users* and *products*.

**Subgraph.** A graph formed from a subset of the nodes and edges of a bigger graph $\mathcal{G}$. An edge is only consider to be part of a subgraph if both endpoints are included in the same subgraph. An **induced subgraph** is one that includes all edges of $\mathcal{G}$ whose endpoints belong to the vertex subset.

**(Near-)clique.** A clique is a subgraph where every pair of vertices is connected. A **near-clique** is an *almost complete* clique.

**(Near-)bipartite core.** A complete bipartite subgraph, i.e., a bipartite subgraph where every two vertices from separate groups are connected. A **near-bipartite core** is an *almost complete* bipartite subgraph.

### 2.1.2 Properties

**Density.** Density measures the proportion of the number of existing to the number of possible edges. It is usually defined as the following ratio:

$$D_{\mathcal{G}} = \frac{|\mathcal{E}|}{|\mathcal{V}|^2} \tag{2.1}$$

**Densification Power-law** Standard graph models, such as the intensively studied preferential attachment model [BA99], normally assume constant average degree during network growth (i.e. they assume that the number of edges is linearly proportional to the number of nodes). Real networks exhibit a relation closer to a power-law ($|\mathcal{E}| \propto |\mathcal{V}|^{\alpha}$) where $\alpha$ is between 1 and 2; 1 would imply constant average degree while 2 would represent a network in which each node has, on average, a degree that is a constant fraction of all nodes.

**Degree power-law.** One of the first relationships identified in real graphs was the power-law distribution of the degree when nodes are sorted in its decreasing order [FFF99]. Lets call $r_v$ the position of node $v$ when nodes are sorted by degree; then the degree $d_v$ is proportional to its rank to the power of a constant:

$$d_v \propto r_v^{\mathcal{R}} \tag{2.2}$$

**Eigenvalues Power-law** The eigenvalues of a graph's adjacency matrix are known to be related to its community structure; intuitively, after a eigen-decomposition, one can see that all the nodes with high score in the same eigenvector will be densely connected to each other. Several studies have analyzed the spectral properties of graphs with power-law degrees and found that the eigenvalues $\lambda_i$ are proportional to their rank $i$ when sorted in decreasing order, up to the power of a constant[FFF99, SFFF03]:

$$\lambda_i \propto i^{\Lambda} \tag{2.3}$$

Later, Mihail and Papadimitriou[MP02] showed that this is a consequence of the degree power-law.

### 2.1.3 Sampling

Networks are often sampled in order to perform efficient computations or to generate smaller but realistic graphs. In Chapter 3, we performed **snowball sampling** [Goo61] in a social network in order to evaluate the scalability of our algorithm. In snowball sampling, one starts from a seed node (or set) and new nodes are included if they have a connection to a node currently in the sample. We used **weighted snowball sampling**, where nodes are sampled with a probability proportional to the number of connections to the current sample, so that community structure could be preserved.

In Chapter 4, two different sampling techniques were included in the algorithm itself. In one instance, we sampled the adjacency matrix uniformly in order to calculate the reconstruction error, while in another instance we sampled a fixed number of the positions with highest reconstruction score in order to approximate the gradient. Further details are included in the appropriate sections.

## 2.2 Learning and Other Techniques

Understanding the structure of networks and identifying communities and anomalies requires new models that fit the data available. We highlight some of the techniques used throughout this dissertation.

### 2.2.1 Factorizations and Objective Functions

Factorizations are the cornerstone of most models developed in this dissertation; in the next section, we overview the most commonly used Matrix and Tensor Factorization techniques. At their basis, Factorization models assume that the data available, represented in matrix or tensor form, can be well approximated using similar but lower-rank matrices or tensors. Intuitively, a representation with fewer degrees of freedom is more general and more interpretable.

The process of fitting a model to the data consists of finding the parameters that optimize some specified criteria. While there are several alternatives, one of the most common approaches requires minimizing some loss function that evaluates how dissimilar the original data and the model are; our goal is for a simple model to capture most of the signal existing in the data. Factorization models are usually fit using least squares, or the entry-wise $\ell_2$ norm, which in the matrix or tensor case is called the Frobenius norm.

Constraints to the model parameters are sometimes included, non-negativity being the most common. On the other hand, one might wish to reduce overfitting or to induce sparsity in order to obtain a model less sensitive to noise. This can be achieved by modifying the model's objective function, often through the addition of $\ell_2$ or $\ell_1$ regularization terms, respectively. For instance, a simple matrix factorization model in which $\ell_1$ penalties are applied to one of the factor matrices and $\ell_2$ penalties to the other would be

$$\underset{\boldsymbol{A},\boldsymbol{B}}{\operatorname{argmin}} ||\boldsymbol{M} - \boldsymbol{A}\boldsymbol{B}^T||_F^2 + \lambda_1||\boldsymbol{A}||_1 + \lambda_2||\boldsymbol{B}||_2$$

where $||\cdot||_F$ is the **Frobenius Norm** defined as

$$||\boldsymbol{M}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}{\boldsymbol{M}_{ij}}^2}$$

and $\lambda_1$ and $\lambda_2$ are meta-parameters. When $\ell_1$ and $\ell_2$ regularizations are applied to the same parameters, we are in the presence of *elastic net regularization* [ZH05].

### 2.2.2 Bayesian Models

As an alternative to specifying the model as a minimization of the difference between it and the data, one can also try to describe the generative process that originated the data. Bayesian models are one of the most popular statistical modeling techniques. In a Bayesian Hierarchical Model, we estimate the parameters of the posterior distribution taking into consideration the prior distribution of these parameters and other dependences of the joint probability model. We operate under the philosophy a) one can model the data under appropriate probability distributions and b) the inclusion of prior knowledge, which is slowly changed as more evidence is considered, creates more robust and natural models.

In Chapter 5, we use a probabilistic graphical model to model the interaction of Points-of-Compromise with credit cards in order to detect the most likely compromised locations.

### 2.2.3 Minimum Description Length (MDL)

The Minimum Description Length (MDL) principle is a formalization of Occam's Razor in which the best hypothesis to explain a given dataset is decided by its ability to compress the data. It was introduced by Jorma Rissanen in 1978 [Ris83].

According to the MDL principle, the best model for describing a set of data is the one for which the sum of the description of the model and the description of the data using the model is minimal; as a consequence, MDL methods naturally avoid overfitting through a trade-off between between model complexity and complexity of the data given the model.

Figure 2.1 illustrates this principle. Suppose one wishes to compress a boolean matrix using a set of smaller "rectangles", i.e. using "rectangles of true regions" as the model. Firstly, one needs to encode all the elements that are necessary to describe the model: the size of the list and the list of rectangles itself. Then, one needs to describe the error of encoding the data this way: some of the rectangles described do not represent only true values, so we need to encode the list of false values inside them, but also some of the true values are not in any rectangle and we also need to encode them, i.e., this is the cost of encoding the data given our model.

Clearly, there are numerous possibilities: at one extreme, you could opt for a rectangle for each one in the matrix (left figure) leading to a very big model, as you have a rectangle per each one in the matrix, but no zero needs to be described; at the other extreme, you could use a single rectangle and then describe every zero in the matrix (right figure). The optimal MDL solution would be the one minimizing the total number of bits required to describe the data.

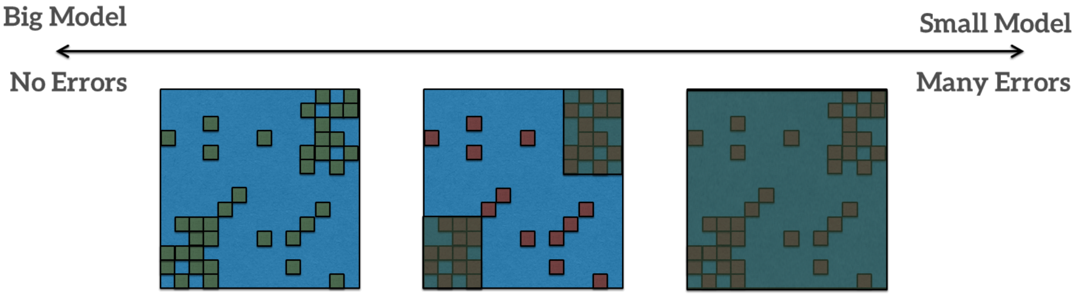


**Figure 2.1: An example on minimum description length balance. On the left, a big model without errors. On the right, a model with many errors to be encoded.**

In this dissertation, we rely on two relatively simple encoding factoids:

1. Representing a number $k$ in the range of $1$ to $N$ requires $\log N$ bits, and this is an optimal code if the probability distribution is *uniform*.
2. In order to represent an unbounded integer $k$, Rissanen proposes

$$\log^*(k) = \log(k) + \log\log(k) + \log\cdots\log(k),$$

a universal code length for integers which assumes that $p(i) \geq p(i+1)$.

## 2.3 Matrices and Spectral Methods

The analysis of graphs from a linear algebra point-of-view follows naturally from their commonly used adjacency or Laplacian matrices. We are led to the analysis of the spectrum of these matrices as it is well understood and does not depend on node orderings or vertex labels. For instance, due to their symmetry, undirected networks only have real eigenvalues.

Additionally, spectral methods lead to a clustering or partitioning of the network. They are related to its community structure, hence increase our understanding of the underlying graph. In the following, we introduce some of the most common matrix factorization techniques.

### 2.3.1 Singular Value Decomposition

In the Singular Value Decomposition (SVD)[GK65] of rank $k$, a $n \times m$ real matrix M is decomposed as

$$\boldsymbol{M} \simeq \boldsymbol{U\Sigma V^T}$$

where $\mathbf{U}$ and $\mathbf{V}$ are real orthogonal $n \times k$ and $m \times k$ matrices, respectively, and $\mathbf{\Sigma}$ is a $k \times k$ non-negative diagonal matrix. The Eckart-Young theorem[EY36] proves this to be the best least squares approximation using regular matrix multiplication and real entries, i.e., this is equivalent to finding:

$$\operatorname*{argmin}_{\boldsymbol{U},\boldsymbol{\Sigma},\boldsymbol{V}} ||\boldsymbol{M} - \boldsymbol{U\Sigma V^T}||_F^2$$

### 2.3.2 Non-Negative Matrix Factorization

In order to interpret the result of a SVD decomposition, one needs to consider that some elements of $\boldsymbol{U}$ and $\boldsymbol{V}$ might be negative. Non-negative decompositions have been shown to be useful in many scenarios such as computer vision and community detection due to their increased interpretability; it is substantially simpler to analyze a process when one is limited to additive composition. For non-negative M, Non-Negative Matrix Factorization (NNMF) methods were developed to overcome this problem. Similarly to SVD, we approximate $\boldsymbol{M}$ using two non-negative matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ such that

$$\begin{gathered}\operatorname*{argmin}_{\boldsymbol{W},\boldsymbol{H}} ||\boldsymbol{M} - \boldsymbol{WH^T}||_F^2 \\ \boldsymbol{W} \geq 0 \\ \boldsymbol{H} \geq 0\end{gathered}$$

While one could apply projected stochastic gradient descent in order to find a solution (i.e., an additive update scheme), Lee and Seung [LS01] developed a multiplicative update method that is proven to converge:

$$\begin{aligned}\boldsymbol{W}_{ik} &\leftarrow \boldsymbol{W}_{ik}\frac{(\boldsymbol{W^T M})_{ik}}{(\boldsymbol{W^T W H^T})_{ik}} \\ \boldsymbol{H}_{jk} &\leftarrow \boldsymbol{H}_{jk}\frac{(\boldsymbol{MH})_{jk}}{(\boldsymbol{WH^T H})_{jk}}\end{aligned} \tag{2.4}$$

Multiplicative updates for other error functions such as the Kullback-Leibler divergence [KL51] have also been proposed.

### 2.3.3 Co-clustering

**Clustering** is a well understood problem: in a clustering task, our goal is to group objects so that instances of the same set are more similar to each other than to instances in other clusters. This grouping is performed taking into consideration the entirety of the feature set, i.e., similarity between two objects is measured taking into consideration everything we know about those two objects.

**Co-clustering** (also called biclustering [Mir98]) is a similar but less common task. The goal is not only to group similar objects together, but also to select a subset of the features which are used to analyze their similarity. Connections to *community detection* in *bipartite graphs* are evident: consider a *user* × *item* dataset, we are looking for a group of users who interact similarly with a given group of items. We are not trying to describe interactions between these group of users and other items.

It is also worth mentioning the distinction between **hard-clustering** and **soft-clustering**. In the first, (co-)clusters are a partition of our elements or features; each element belongs to exactly one cluster. On the other hand, in a soft-clustering setting, we allow elements to belong to multiple clusters and do not impose boolean constraints; we might be looking for a weight between an element and a cluster (e.g., likelihood of belonging to a cluster) [PM10].

## 2.4 Tensors

While matrices are appropriate representations for unlabeled graphs, it is not clear how they can be used to convey edge-labels. Tensors emerge as a natural alternative, as they are multidimensional arrays that generalize the concept of matrices. 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 a vector, a two-mode tensor is a matrix, etc. As a consequence, they are a popular choice when representing time-evolving relations, such as Facebook interactions [PFS12], sensor networks [STF06] or EEG data for detecting the origin of epilepsy seizures [AABB+07].

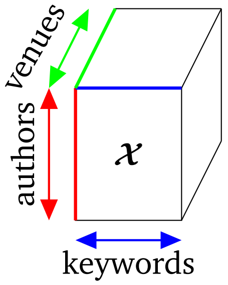


**Figure 2.2: A three-mode tensor of Authors, Keywords and Venues.**

We will consider three-mode tensors, unless noted otherwise. In particular, we can treat an edge-labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{L}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{L}$ is the set of categorical labels and $\mathcal{E}$ is the set of edges $\{(v_1, v_2, l) \in \mathcal{E} | v_1 \in \mathcal{V}, v_2 \in \mathcal{V}, l \in \mathcal{L}\}$ as a three-mode tensor $\boldsymbol{\mathcal{X}} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}| \times |\mathcal{L}|}$.

### 2.4.1 Tensor Operations

With appropriate modifications, most matrix operations can be extended to the tensor scenario. For an in-depth exposition, we forward the reader to a review by Tamara Kolda and Brett Bader [KB09] even though we use slightly different notation. In this subsection, we will briefly mention the most common operations that are used throughout this dissertation. We assume a $N$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$.

**Frobenius Norm** The matrix Frobenius norm can be trivially extended to tensors:

$$||\boldsymbol{\mathcal{X}}||_F = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_n=1}^{I_n} {\boldsymbol{\mathcal{X}}_{i_1 i_2 \cdots i_n}}^2}$$

**Outer Product** The outer product of a n-mode tensor of size $I_1 \times I_2 \times \cdots \times I_n$ with a vector of size $I_{n+1}$ creates a tensor with n+1 modes of size $I_1 \times I_2 \times \cdots \times I_n \times I_{n+1}$. This notation is typically use to represent rank-1 tensors, so if

$$\boldsymbol{\mathcal{X}} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c},$$

then

$$\boldsymbol{\mathcal{X}}_{ijk} = \boldsymbol{a}_i \boldsymbol{b}_j \boldsymbol{c}_k$$

**Hadamard Product** Also known as the entry-wise product. When tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ have the same dimensions, then $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} * \boldsymbol{\mathcal{Y}}$ is equivalent to

$$\boldsymbol{\mathcal{Z}}_{i_1 i_2 \cdots i_n} = \boldsymbol{\mathcal{X}}_{i_1 i_2 \cdots i_n} \cdot \boldsymbol{\mathcal{Y}}_{i_1 i_2 \cdots i_n}$$

We use the **Hadamard Division** operator $\oslash$ analogously.

**Tensor Fiber** Fibers are the higher-order equivalents to matrix rows or columns and they are defined by fixing all but one index. We use the terms rows, columns and tubes to represent mode-1, mode-2 and mode-3 fibers, respectively.

**Tensor Matricization** Matricization is the concept of unfolding a tensor into a matrix. The mode-k matricization of tensor $\boldsymbol{\mathcal{X}}$ is denoted by $\boldsymbol{\mathcal{X}}_{(k)}$ and rearranges the mode-k fibers to be the columns of the resulting matrix, i.e., $\boldsymbol{\mathcal{X}}_{(1)}$ is a matrix of size $I_1 \times I_2 \times \cdots \times I_n$, $\boldsymbol{\mathcal{X}}_{(2)}$ is a matrix of size $I_2 \times I_1 I_3 \cdots I_n$, etc.

**Kronecker Product** The Kronecker product of two matrices $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ and $\boldsymbol{B} \in \mathbb{R}^{K \times L}$ results in a matrix of size $IK \times JL$ defined as

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}_{11}\boldsymbol{B} & \cdots & \boldsymbol{A}_{1J}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{I1}\boldsymbol{B} & \cdots & \boldsymbol{A}_{IJ}\boldsymbol{B} \end{bmatrix}$$

**Khatri-Rao Product** Given matrices $\boldsymbol{A} \in \mathbb{R}^{I \times K}$ and $\boldsymbol{B} \in \mathbb{R}^{J \times K}$ with the same number of columns, their Katri-Rao product is a $\mathbb{R}^{IJ \times K}$ defined as

$$\boldsymbol{A} \odot \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}_1 \otimes \boldsymbol{B}_1 & \cdots & \boldsymbol{A}_K \otimes \boldsymbol{B}_K \end{bmatrix}$$

### 2.4.2 Tensor Factorizations

Tensor factorizations identify the underlying low-dimensional latent structure of the $n$-dimensional data. Latent factors are then used to identify anomalies, to estimate missing values or to understand how the data was generated in the first place.

Among the many tensor factorizations flavors [KB09], the PARAFAC [Har70] (also called CP) decomposition is one of the most popular as it factorizes a tensor into a sum of rank-1 tensors. In three modes, the problem is usually framed as finding factor matrices $\boldsymbol{A}$ , $\boldsymbol{B}$ and $\boldsymbol{C}$ that minimize the squared error between $\boldsymbol{\mathcal{X}}$ and the reconstructed tensor $\hat{\boldsymbol{\mathcal{X}}}$:

$$\min_{\hat{\boldsymbol{\mathcal{X}}}} \left\| \boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}} \right\|_F^2 = \min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} \left\| \boldsymbol{\mathcal{X}} - \sum_f \boldsymbol{A_f} \circ \boldsymbol{B_f} \circ \boldsymbol{C_f} \right\|_F^2 \tag{2.5}$$

The optimization problem in Equation 2.5 is convex when one considers one of the factor matrices while keeping the others fixed. It reduces to a linear least-squares problem whose optimal solution is given by

$$\hat{\boldsymbol{A}} = \boldsymbol{\mathcal{X}}_{(1)} \left[ (\boldsymbol{C} \odot \boldsymbol{B})^T \right]^\dagger \tag{2.6}$$

where $\boldsymbol{A}^\dagger$ represents the Moore-Penrose pseudoinverse [Pen56].

Similarly to Equation 2.4, there are multiplicative updates that enforce non-negativity constraints [WW01]:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} * \frac{\boldsymbol{\mathcal{X}}_{(1)}(\boldsymbol{B} \odot \boldsymbol{C})}{\boldsymbol{A}(\boldsymbol{B} \odot \boldsymbol{C})^T(\boldsymbol{B} \odot \boldsymbol{C})}$$

### 2.4.3 Rank-1 Decompositions

Evidence indicates that, if the factors of the CP/PARAFAC decomposition are *sparse*, then doing the decomposition by extracting a rank-one component each time approximates the *batch*, full rank decomposition with very high accuracy [PSB13]. This factoid is used in multiple algorithms of this dissertation (e.g., Com$^2$, FASTSTEP), so we believe it useful to describe the rank-1 case in more detail.

When $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ are vectors, then the CP/PARAFAC decomposition reduces to:

$$\underset{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}}{\operatorname{argmin}} ||\boldsymbol{\mathcal{X}} - \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c}||_F^2$$

and, if $\boldsymbol{\mathcal{X}}$ is non-negative, then so are $\boldsymbol{a}$,$\boldsymbol{b}$ and $\boldsymbol{c}$.

**Lemma 1** *The Alternating Least Squares method (Equation 2.6) reduces to*

$$\begin{aligned}
\boldsymbol{a}_i &\leftarrow \sum_{j=1,k=1}^{M,K} \boldsymbol{\mathcal{X}}_{ijk}\boldsymbol{b}_j\boldsymbol{c}_k \\
\boldsymbol{b}_j &\leftarrow \sum_{i=1,k=1}^{N,K} \boldsymbol{\mathcal{X}}_{ijk}\boldsymbol{a}_i\boldsymbol{c}_k \\
\boldsymbol{c}_k &\leftarrow \sum_{i=1,j=1}^{N,M} \boldsymbol{\mathcal{X}}_{ijk}\boldsymbol{a}_i\boldsymbol{b}_j
\end{aligned} \tag{2.7}$$

*when we ask for rank-1 results.*

**Proof 1** *According to the Alternating Least Squares method, one fixes matrices* $\mathbf{B}$ *and* $\mathbf{C}$ *and solves for* $\mathbf{A}$ *through the minimization of*

$$\min_{\hat{\mathbf{A}}} ||\boldsymbol{\mathcal{X}}_{(1)} - \hat{\mathbf{A}}(\mathbf{C} \odot \mathbf{B})^T||_F \tag{2.8}$$

*Due to properties of the Moore-Penrose pseudoinverse of the Khatri-Rao Product [KB09]* $\hat{\mathbf{A}}$ *has closed form solution of the form* $\hat{\mathbf{A}} = \boldsymbol{\mathcal{X}}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C^T C} * \mathbf{B^T B})^\dagger$.

*When* $\mathbf{A}$, $\mathbf{B}$ *and* $\mathbf{C}$ *are vectors (*$\mathbf{a}$, $\mathbf{b}$ *and* $\mathbf{c}$*, resp.), the Khatri-Rao product (*$\mathbf{c} \odot \mathbf{b}$*) is equivalent to the Kronecker product (*$\mathbf{c} \otimes \mathbf{b}$*). The inner products* $\mathbf{c}^T\mathbf{c}$ *and* $\mathbf{b}^T\mathbf{b}$ *are scalars and so is* $(\mathbf{c^T c} * \mathbf{b^T b})^\dagger$*. The product of* $\boldsymbol{\mathcal{X}}_{(1)}$*, the* $N \times MK$ *matricization of* $\boldsymbol{\mathcal{X}}$*, and*

$\mathbf{c} \otimes \mathbf{b}$, *the* $MK \times 1$ *column vector, reduces to Equation 2.7. A different proof can be found in [DLDMV00].*

Note that the updates of Equation 2.7 can be performed by iterating over the non-zeros of tensor $\boldsymbol{\mathcal{X}}$.

### 2.4.4 Coupled Factorizations

We are often interested in analyzing real-world tensors when additional information is available from distinct sources. For example, in a simple recommendation task with user×movie ratings, we might have user demographics data available which we wish to incorporate when predicting future ratings.

Coupled Matrix-Tensor Factorizations and Coupled Tensor-Tensor Factorizations are a natural extension to the standard tensor factorization formulation. For instance, the factorization of a third-order tensor $\boldsymbol{\mathcal{X}}$ coupled with a matrix $\boldsymbol{M}$ on its first mode can be obtained by minimizing

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{D}} \left\| \boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}} \right\|_F^2 + \alpha \left\| \boldsymbol{M} - \hat{\boldsymbol{M}} \right\|_F^2 \tag{2.9}$$

where $\alpha$ is a parameter representing the strength of the coupling for this task, i.e., how important $\boldsymbol{M}$ is to improve the prediction of the new ranking or to improve the quality of the factors.

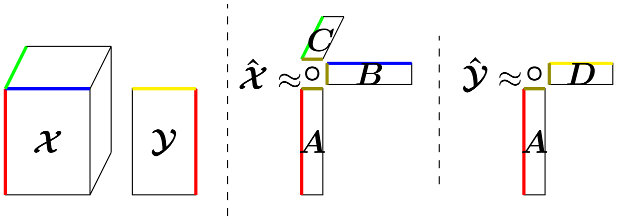

$\boldsymbol{\mathcal{X}}$
$\boldsymbol{\mathcal{Y}}$
$\hat{\boldsymbol{\mathcal{X}}} \approx \circ$
C
B
A
$\hat{\boldsymbol{\mathcal{Y}}} \approx \circ$
D
A

**Figure 2.3: A simple Coupled Matrix-Tensor Factorization.**

The matrix part of the Coupled Matrix-Tensor Factorization depicted in Figure 2.3 is useful to model additional static information about one of the modes of the tensor of interest. Whenever the side information available is dynamic (time-evolving), a model where two tensors are coupled along (at least) one of the dimensions is more appropriate, as the time component can be preserved, e.g.:

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{T}} \left\| \boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}} \right\|_F^2 + \alpha \left\| \boldsymbol{\mathcal{Y}} - \hat{\boldsymbol{\mathcal{Y}}} \right\|_F^2 \tag{2.10}$$

where:

$$\hat{\boldsymbol{\mathcal{X}}} = \sum_f \boldsymbol{A}_f \circ \boldsymbol{B}_f \circ \boldsymbol{T}_f$$

$$\hat{\boldsymbol{\mathcal{Y}}} = \sum_f \boldsymbol{A}_f \circ \boldsymbol{C}_f \circ \boldsymbol{T}_f$$

Many techniques have been proposed to solve this non-negative optimization problem, such as projected Stochastic Gradient Descent (SGD) [BTK+14] (i.e., additive update rules) and multiplicative update rules. Most of this work extends Lee and Seung's multiplicative matrix updates formulae [LS01] for matrices, notably the simple extension for tensors [WW01] and the many coupled extensions, e.g. Generalized Tensor Factorization [Yıl12, ŞECA13]. Update equations for this scenario are also well known:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} * \frac{\boldsymbol{\mathcal{X}}_{(1)}(\boldsymbol{B} \odot \boldsymbol{T}) + \alpha \boldsymbol{\mathcal{Y}}_{(1)}(\boldsymbol{C} \odot \boldsymbol{T})}{\boldsymbol{A}(\boldsymbol{B} \odot \boldsymbol{T})'(\boldsymbol{B} \odot \boldsymbol{T}) + \alpha \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{T})'(\boldsymbol{C} \odot \boldsymbol{T})}$$

$$\boldsymbol{B} \leftarrow \boldsymbol{B} * \frac{\boldsymbol{\mathcal{X}}_{(2)}(\boldsymbol{A} \odot \boldsymbol{T})}{\boldsymbol{B}(\boldsymbol{A} \odot \boldsymbol{T})'(\boldsymbol{A} \odot \boldsymbol{T})}$$

$$\boldsymbol{C} \leftarrow \boldsymbol{C} * \frac{\boldsymbol{\mathcal{Y}}_{(2)}(\boldsymbol{A} \odot \boldsymbol{T})}{\boldsymbol{C}(\boldsymbol{A} \odot \boldsymbol{T})'(\boldsymbol{A} \odot \boldsymbol{T})}$$

$$\boldsymbol{T} \leftarrow \boldsymbol{T} * \frac{\boldsymbol{\mathcal{X}}_{(3)}(\boldsymbol{A} \odot \boldsymbol{B}) + \alpha \boldsymbol{\mathcal{Y}}_{(3)}(\boldsymbol{A} \odot \boldsymbol{C})}{\boldsymbol{T}(\boldsymbol{A} \odot \boldsymbol{B})'(\boldsymbol{A} \odot \boldsymbol{B}) + \alpha \boldsymbol{T}(\boldsymbol{A} \odot \boldsymbol{C})'(\boldsymbol{A} \odot \boldsymbol{C})}$$

The problem is not as well understood when one of the factorizations is symmetric, e.g., $\hat{\boldsymbol{\mathcal{Y}}} = \sum_f \boldsymbol{A}_f \circ \boldsymbol{A}_f \circ \boldsymbol{T}_f$, as this is no longer a linear problem.

Welling and Weber [WW01] note the need for a scaling exponent (for the simple, non-coupled case):

$$\boldsymbol{A} \leftarrow \boldsymbol{A} * \left( \frac{\boldsymbol{\mathcal{X}}_{(1)}(\boldsymbol{A} \odot \boldsymbol{T})}{\boldsymbol{A}(\boldsymbol{A} \odot \boldsymbol{T})'(\boldsymbol{A} \odot \boldsymbol{T})} \right)^{1/d}$$

which should be at least $1/2$ for the matrix case, although no proof is provided. To the best of our knowledge, the best theoretical bound is $1/3$ when the matrix is semi-definite positive [HXZ+11]. Empirical results (for the coupled case) indicate that removing the exponent ($d = 1$) might eliminate the convergence guarantees, but even small perturbations converge (e.g., 0.98 in [ECA13]).

We recommend an exponent of $1/3$, as convergence is exponentially fast in any case.

## 2.5 Notation and Common Symbols

Table 2.1 overviews the most common symbols and notation described in this chapter and used throughout the rest of this dissertation.

| Symbol | Definition |
|---|---|
| $\mathcal{G}$ | a (edge-labeled) graph |
| $\mathcal{V}$ | set of nodes |
| $\mathcal{E}$ | set of (labeled) edges |
| $\mathcal{L}$ | set of labels |
| $\boldsymbol{M'}$ | Matrix transpose |
| $\boldsymbol{M^T}$ | Matrix transpose |
| $\boldsymbol{M}_k$ | Column $k$ of $\boldsymbol{M}$ |
| $\boldsymbol{M} \geq 0$ | $\boldsymbol{M}$ is a non-negative matrix |
| $\boldsymbol{M} \succeq 0$ | $\boldsymbol{M}$ is positive semi-definite |
| $\lVert\boldsymbol{\mathcal{X}}\rVert_F$ | Frobenius norm of tensor $\boldsymbol{\mathcal{X}}$ |
| $\boldsymbol{\mathcal{X}}_{(\boldsymbol{k})}$ | Mode-k matricization |
| $\circ$ | Vector outer product |
| $\otimes$ | Kronecker product |
| $\odot$ | Khatri-rao product |
| $*$ | Hadamard (entrywise) product |
| $\oslash$ | Hadamard (entrywise) division |

**Table 2.1: Symbols and Notation used throughout this dissertation.**

# Part I

# Networks and Matrices

# Overview and Related Work

Nodes in real-world networks organize into communities or clusters, which tend to exhibit a higher degree of 'cohesiveness' with respect to the underlying relational patterns. Group formation is natural in social networks as people organize in families, clubs and political organizations [WF94a]. Communities also emerge in protein-protein interaction or gene-regulatory networks whereby genes associated to a common metabolic function tend to be more densely connected [SKJ06], or in the World Wide Web where hyperlinks between theme-related websites are more prevalent [FLG00].

In Part I, we explore the important problem of identifying these groups from given (unlabeled) graph data. We start with an overview of the related work in community detection in static graphs, with an emphasis on factorization methods of binary matrices. At the end of this overview, we highlight currently unanswered questions, some of which we tackle in this dissertation.

## I Related Work: Community Detection

For an in-depth and thorough discussion of community detection methods, we kindly refer to the excellent surveys of Santo Fortunato [For10, FH16].

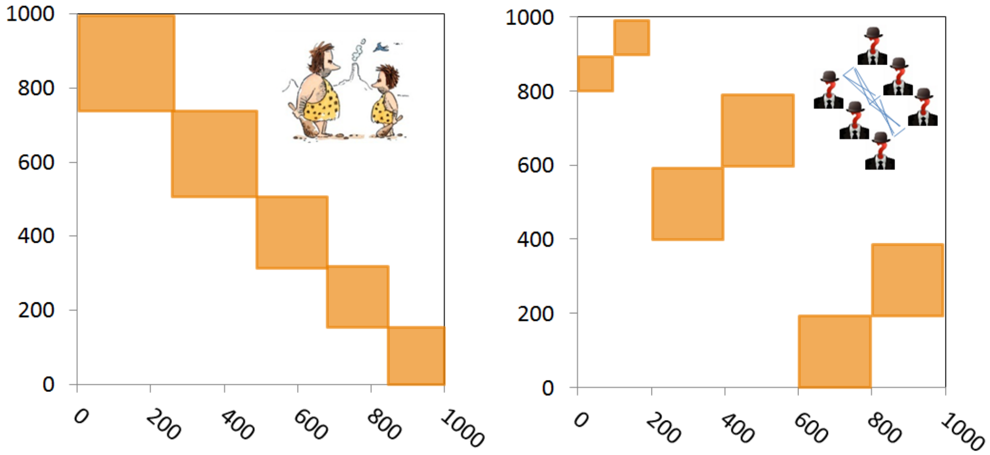
The vast ongoing research in community detection is arguably an indicator of the problem's inherent difficulty. As discussed in [YL12b], the challenges faced by community detection methods are usually threefold:

a) a lack of consensus on the structural definition of network community;

b) the fact that node subset selection overlaid to the combinatorial structure of graphs typically leads to intractable formulations;

c) the lack of ground-truth to carry out an objective validation on real data.

Nevertheless, the widespread notion of cohesiveness used to group nodes has typically reflected that community members are well connected among themselves and relatively well separated from the remaining nodes. The mental image is the *cavemen* graph [WF94a], where the adjacency matrix is block-diagonal, as shown in Figure 2.4a. All graph-cut algorithms assume such structure. As an extension to this definition, the *familiar strangers* [ATMF12] pattern can be applied to unipartite or bipartite graphs. In this pattern, one group of people is connected to a second, forming a bipartite-core; people on each side of the bipartite-core can be grouped together given the similarity of their connections, although they do not know each other. This results in an off-diagonal block in the adjacency matrix (see Figure 2.4b), but one can also consider it as an extension of *cavemen* graphs to "rectangular" adjacency matrices, e.g. user × products data, where the result would resemble a block-diagonal rectangular matrix.



**(a) Cavemen communities.**

**(b) The "familiar strangers" pattern.**

**Figure 2.4: Adjacency matrices of classical community definitions.**

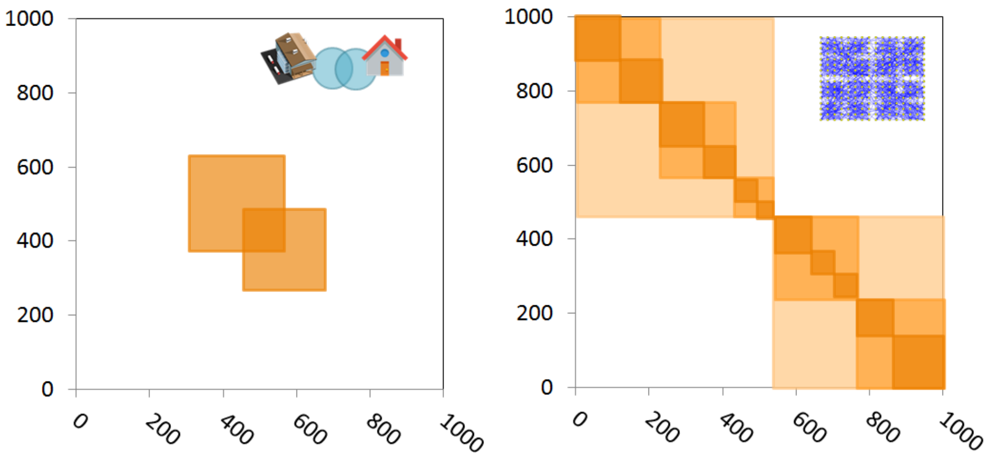
Recent developments have showed that a realistic picture of the community structure is not as simple. Therefore, the definition of community *we consider* is less restrictive: *we consider communities to be sets of nodes that help us compress the graph*. Note that the typical structures of Figure 2.4, when they exist, are very useful from a compression perspective. In the rest of this section, we first discuss common community detection approaches, then describe some of the most common evaluation procedures and, at the end, we discuss key applications of community detection.

## I.1 Methods

While multiple classifications can be created, we consider it useful to characterize methods along two particular dimensions:

a) *Overlapping communities* [YL12a]: methods might acknowledge that one node can participate in multiple communities. For instance, we have friends from high school, friends from college, and colleagues from work; the three groups have little overlap, and few connections between them (see Figure 2.5a);

b) *Hierarchical communities*: methods might take into consideration that there are "no good cuts" and that communities can be analyzed at different granularities (see Figure 2.5b).



**(a) Overlapping communities.**

**(b) Hierarchical communities.**

**Figure 2.5: Algorithm characteristics: methods can be classified by their ability to find overlapping or hierarchical communities.**

**Optimization: Community Cohesiveness Metrics** While most community detection methods can be casted as the optimization of some cost function, a number of metrics used across the literature that are worth describing.

**Conductance or Normalized Cut.** According to this metric, the best communities are densely linked and attached to the rest of the network via few edges. The conductance $\phi$ of a set of nodes $S \subset \mathcal{V}$ is defined as the ratio between the number of edges with one endpoint in $S$ and one in $\bar{S}$, and the sum of degrees of the nodes in $S$. More formally, if **A** is the adjacency matrix of graph $\mathcal{G}$:

$$\phi(S) = \frac{\sum_{i \in S, j \notin S} \boldsymbol{A}_{ij}}{\sum_{i \in S, j \in \mathcal{V}} \boldsymbol{A}_{ij}} \tag{2.11}$$

Conductance [SM00] provides a metric for the quality of the cut and lower conductance is used as a proxy for community quality.

**Modularity.** Introduced by Newman and Girvan[NG04, New06], modularity is based on the idea that, on expectation, a random graph should not have community structure. Edge density in subgraphs is then compared to the expected density one would have if the graph vertices were attached disregarding this structure. In general, the modularity score $Q$ for a subset of nodes $S$ is given by

$$Q(S) = \frac{1}{2E} \sum_{i \in S, j \in S} (\boldsymbol{A}_{ij} - \boldsymbol{P}_{ij}) \tag{2.12}$$

where $\boldsymbol{P}_{ij}$ represents the probability of a connection between nodes $i$ and $j$ according to the chosen *null model*. A simple *null model* could set $\boldsymbol{P}_{ij} = \frac{2E}{N(N-1)}$ but that is uncommon as it does not describe real-networks correctly (most notably, it provides a Poissonian degree distribution). A more common approach is to choose as *null model* a graph with the same degree distribution as the original (in expectation), in which case the probability of an edge between two nodes $i$ and $j$ existing is proportional to their degrees $d_i$ and $d_j$, yielding $\boldsymbol{P}_{ij} = \frac{d_i d_j}{2E}$.

Multiple methods have been proposed to optimize these metrics [New04, DA05] and others, such as Weighted Community Clustering [PPDSBLP12, PPDSLP14], which tries to maximize the number of triangles closed inside $S$, and propinquity methods [ZWWZ09], which slowly modify the graph to better match current estimates of $\boldsymbol{P}_{ij}$.

**Partitioning** In graph-partitioning methods, the goal is to split the vertices in groups of pre-specified size, or to enforce penalties if sizes differ significantly [FC07, Pot97]. The assumption is that groups are connected by bridge edges and will emerge if we cut them. While the result is similar to hard-clustering, the approach is different: methods usually start by defining edge centrality and then removing edges repeatedly to form clusters. Another typical approach involves bisectioning the graph into two groups and repeated application to find groups of similar sizes.

METIS [KK95] is the most commonly used software package to find appropriate graph partitions, as it supports multiple partitioning objectives.

Label propagation methods [Gre10] are also related to graph partitioning, even though they do not enforce groups to have similar sizes. They assume that each node has a label which indicates to which group it belongs, and continuously update each node's label based on the majority label of neighboring nodes.

In general, graph partitioning methods are not considered to be very good for community detection [For10], as they require an explicit definition of the number of groups and their size. Similar-sized communities tend to be unnatural, as most common community definitions do not require them to be of similar size.

**Random-Walk methods** Random Walk-based methods group nodes by their score when doing a Random Walk with Restart [TFP08] (RWR). The idea is similar to PageRank [PBMW99], but instead of computing a global PageRank vector indicating the probability of landing on a given node during a random walk on the graph, RWR computes the probability of landing on a given node $j$ when restarts are made to a given node $i$. Communities can be obtained from RWR by clustering the $\boldsymbol{D}_{ij}$ matrix created [Zho03]. Modifications have been proposed, such as biasing random walk to nodes closer to the origin [ZL04], or to consider overlapping communities [CWZW11].

**Spectral** In Chapter 2, we have seen that the spectral properties of matrices are frequently used to find clusters. Research started with Donath and Hoffman [DH73], who used eigenvalues to partition a graph, but the whole range of spectral approaches have been applied, e.g. NNMF for overlapping community detection [PRES11]. Dhillon [Dhi01] proposed a co-clustering partition of bipartite graphs based on singular vectors of a scaled word-document matrix. [GMZ03] provided one of the first analysis of the AS graph through spectral analysis.

This area has been heavily extended, including spectral clustering methods in the presence of node-attributes [GFBS14, GFRS13].

**Generative Models** Generative models start by representing the network as a group of communities, then rely on inference methods to learn the most appropriate parameters to fit the model to the network. Block modeling [WF94a] is one of the most common of such approaches. Nodes are decomposed into classes with similar properties, usually a mixture of intra-class properties (e.g. density) and inter-class properties, describing connections to nodes in different classes. The reader is referred to [DBF05] for a more detailed analysis.

Yang and Leskovec [YL12a] proposed a community-affiliation model which allows overlapping and hierarchical community definitions. In this model, nodes can be affiliated with several groups and overlapping affiliations increase connection probabilities. Gionis et al. [GMS04] proposed a probabilistic model where hierarchical tiles represent the boolean adjacency matrix.

**Information theory** Orthogonally to the previous approaches, methods often try to incorporate information theory concepts in an effort to avoid overfitting their model to the data. For instance, methods might use the AIC [Aka74] or BIC [S$^{+}$78] criteria when learning the correct number of parameters.

On the other hand, some approaches rely on information theory at their core. Rosvall et al. proposed information-theoretic modules that try to maximize mutual information [RB07]. Many methods leverage MDL to find succinct representations of the original data, such as AutoPart [Cha04] and Cross-associations [CPMF04] which find minimal encodings of square and rectangular matrices, respectively. Motivated by understanding large graphs, VoG [KKVF14] uses MDL to encode easy to understand structures such as stars, chains, cliques, etc.

## I.2 Applications

One of the first applications of community detection methods was image segmentation [SM00], given its relationship to graph partitioning. Nevertheless, most applications nowadays focus on understanding network topology, in particular Social Network analysis [WF94a] in widespread domains, such as the Internet [GMZ03], Wikipedia articles [KKVF14], citation networks and patents [GFBS14]. Blondel et al. [BGLL08] were able to recover Belgium's linguistic split using a phone calls network.

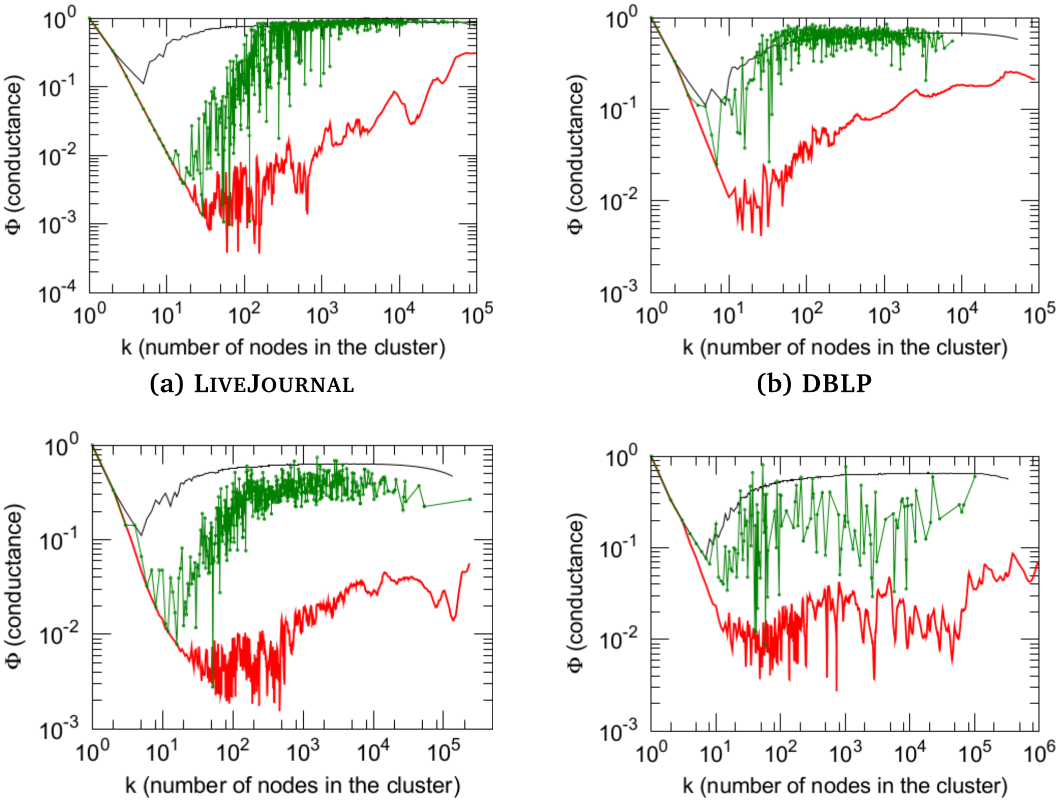
Biological networks have also inspired plenty of applications. Rives and Galitski [RG03] studied the protein-protein interaction network of the *yeast* and were able to identify its modular organization. Gene regulatory networks and gene expression [WH04, GFBS14] also provide strong motivation: gene co-occurrence is biologically motivated by evolution.

## I.3 Challenges: No good cuts

All previous community detection methods and metrics have been either explicitly or implicitly aimed at extracting areas of high and/or uniform density in the adjacency matrix, e.g. near cliques in the corresponding graphs. However, analysis by Leskovec et al.[LLDM08] of what are considered to be ground-truth communities in different datasets[1] has shown that these big communities don't necessarily achieve good "community quality" scores using metrics such as conductance; they developed the Network Community Profile (NCP) plot which measures the quality of the best possible community in that network as a function of the size of the community, i.e. the best conductance for a set of cardinality $k$ in that network:

$$\Phi(k) = \min_{S \subset \mathcal{V}, |S|=k} \phi(S) \tag{2.13}$$

[1] Communities that we would be interested in finding automatically, that have been either manually assessed or self-declared.



**(a) LiveJournal**

**(b) DBLP**

**(c) Amazon**

**(d) IMDB**

**Figure 2.6: NCP plots of four different datasets (red). Conductance values for ground-truth communities (green).** ***Source:*** ***[LLDM08]*****. Note the low community quality achieved using this metric as communities get bigger.**

In Figure 2.6 one can find, in red, the NCP plots of four different datasets. A steady improvement in the "community score" can be seen up to sizes of 10-100 nodes and then, as communities get bigger, no subset produces good "community scores". In green, one can find the conductance values for the ground-truth communities that, as expected, achieve worse scores than those in the NCP plot. These results seem to agree with Dunbar[Dun98] who predicted that 150 is the upper limit on the size of a human community. Leskovec et al.[LLDM08] questioned whether larger communities even exist given that, as they grow larger, they become so diverse that they stop existing in the traditional "network community" sense.

We come to the conclusion that existing models are not appropriate to represent these structures, whose existence is backed by self-reported big communities in multiple domains. In Chapter 3, we analyze their shape and provide new definitions that generalize the cavemen model.

## II Related Work: Decomposition of Boolean Matrices

In Chapter 4, we develop novel factorization approaches to tackle the community detection problem. In this section, we provide a summary of current alternatives for the decomposition of boolean matrices. Throughout this section, we consider a $n \times m$ boolean matrix to be factored.

While SVD or NNMF (see Chapter 2) can be applied to boolean matrices, there are no clear extensions when we need the reconstructed matrix to be boolean. One simple idea is rounding or thresholding the reconstructed matrix, but no guarantee can be given on the reconstruction error. Another possibility is thresholding the factor matrices and using boolean algebra in order to obtain a boolean reconstruction, but selecting the appropriate threshold is a difficult problem as a clear cut-off might not exist.

In fact, Miettinen [MMG$^+$08] showed that Boolean Matrix Factorizations (BMF) methods could achieve lower reconstruction error than SVD in boolean data and proposed an algorithm using association rules (ASSO) which exploits the correlations between columns, but unfortunately it's time complexity is $O(nm^2)$, i.e., it cannot be applied to large scale networks.

Zhang et al.[ZLD$^+$10] proposed two approaches for BMF, one using a penalty in the objective function (BMF-PENALTY) which achieved good results for dense datasets, and an alternative thresholding method (BMF-THRESH) which by thresholding factor matrices is better suited for sparse datasets. None of these methods is scalable and they have the problem of forcing a tiling of the data matrix, as each factor is effectively treated as a block. In particular, the notion of "importance" inside a cluster, which previously existed in NNMF, is now lost and the analysis of the resulting factors is limited.

Tao Li [Li05] proposed an extension of the K-means algorithm to the two-sided case where $\mathbf{M}$ is decomposed into $\mathbf{AXB^T}$ with $\mathbf{A}$ and $\mathbf{B}$ binary, and an alternating least squares method when $\mathbf{X}$ is the identity matrix.

In Logistic PCA (L-PCA), Schein et al.[SSU03] replace PCA's Gaussian assumption with a Bernoulli distribution and fit their new model using an alternating least squares algorithm that maximizes the log-likelihood. Their alternating algorithm has running time $O(nm)$ and it is hard to interpret due to the possibility of negative values in the factors.

Chapter based on work that appeared at ECML PKDD 2014 [AGMF14].

# Chapter 3

# Hyperbolic Communities

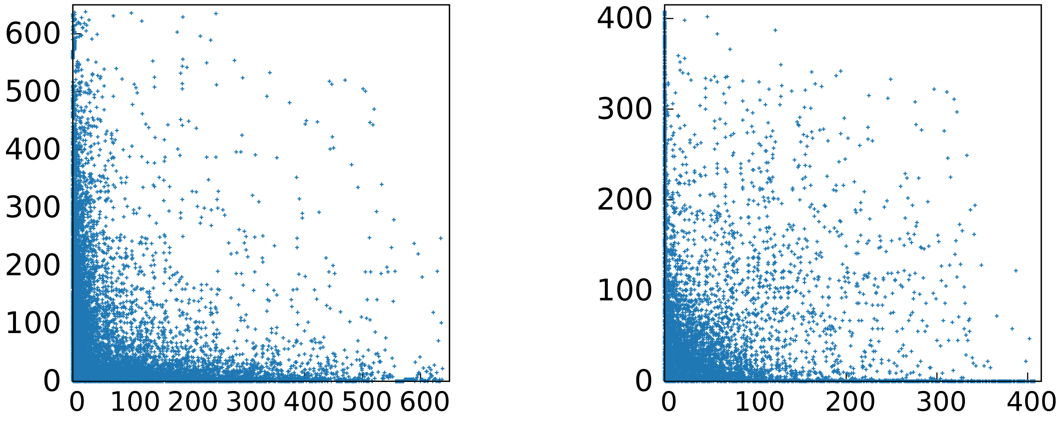
Given a large social network, what do real communities look like? How does their size affect their structure, shape, and density of connections? Are the communities' degree distributions uniform as implied by traditional community detection algorithms that look for quasi-cliques? One would intuitively expect that larger communities exhibit similar relational patterns to the whole graph. Accordingly, do the communities' degree distributions obey power laws? What is the structure of communities in large, real social networks and what are suitable models to describe them? Moreover, how can one find these communities in an effective and scalable way by leveraging this particular structure and without any user-defined parameters?

We analyze four real-world social networks with ground-truth communities and provide empirical evidence that communities exhibit power law degree distributions. As such, they are typically best represented as having a hyperbolic structure in the adjacency matrix, rather than rectangular (uniform) structure. We detail HyCoM - the Hyperbolic Community Model - as a better representation of communities and the relationships between their members, and introduce HyCoM as a scalable algorithm to detect communities with hyperbolic structure. To illustrate our model and algorithm, Figure 3.1a represents the adjacency matrix of a real (ground-truth) community externally provided when nodes are ordered by degree, and Figure 3.1b shows the adjacency matrix of an exemplary community found by our algorithm. Clearly, both communities do not show uniform density. In a nutshell, the main contributions of our work are:

- **Hyperbolic Community Model:** We provide empirical evidence that communities in large, real social graphs are better modeled using a hyperbolic model.
- **Scalability:** We develop HyCoM-FIT, an algorithm for the detection of hyperbolic communities that scales linearly with the number of edges.
- **No user-defined parameters:** HyCoM-FIT detects communities in a parameter-

free fashion, transparent to the end-user.

- **Effectiveness:** We applied HyCoM-FIT on real data where we discovered communities that agree with intuition.
- **Generality:** HyCoM includes uniform block communities used by other algorithms as a special case.



**(a) Motivation for our work: a real ground-truth community from the YouTube dataset.**

**(b) Result of our work: a community found by HyCoM-FIT.**

**Figure 3.1: Side-by-side: comparing a ground-truth community and one found by HyCoM-FIT.**

## 3.1 Empirical Observations

In this section, we provide empirical evidence that real communities are not blocks of uniform density and are best represented as hyperbolic structures. We examined a collection of four real networks (Table 3.1) previously used in the literature [YL12a, YL12b] with significantly different ground-truth definitions, available in the Stanford Network Analysis Project (SNAP) collection. In addition to the underlying graph, these datasets provide *externally defined communities*, which we consider to be ground-truth. We only consider communities with a minimum of 15 nodes. How these communities were defined and their internal structure is totally dependent on the application domain:

- The YouTube and LiveJournal datasets are standard friendship networks. Each node represents a user of the website and friendship relations establish links between

| | Networks with ground-truth communities | | | | Network with node labels |
|---|---|---|---|---|---|
| **Dataset** | Amazon | DBLP | YouTube | LiveJournal | Wikipedia |
| **Nodes** | 334 863 | 317 081 | 1 134 890 | 3 997 962 | 143 508 |
| **Edges** | 1 851 744 | 2 099 732 | 5 975 248 | 34 681 889 | 3 753 156 |

**Table 3.1: Summary of real-world networks used.**

them. In these websites, users are also able to form groups that others can join, which we consider as a ground-truth communities. Therefore, they tend to represent common interests.

- The DBLP dataset is a computer science co-authorship network: two authors (nodes) are connected if they published at least one paper together. Publication venues (i.e. specific journal or conference series like ECML/PKDD) define ground-truth communities. Therefore, ground-truth communities roughly correspond to scientific fields.
- The Amazon dataset was collected by crawling the Amazon website and is based on the "customers who bought this item also bought" feature. Each individual node corresponds to a product and an edge exists if products $i$ and $j$ are frequently co-purchased. Products are organized hierarchically in categories and we view products in the same category as forming a ground-truth community. In this scenario, communities represent product similarity.

Exploring the communities in these networks allows for a better understanding of common community structures.

**Density.** Firstly, in Figure 3.2 we see that community size impacts edge density (here plotted for the DBLP data). While small communities might have any density, big communities are consistently less dense. These simple observations already indicate that blocks of uniform density are not the appropriate representation for a wide range of communities: small communities might go from small stars to full-cliques and big communities are usually not dense enough for a uniform block repre-

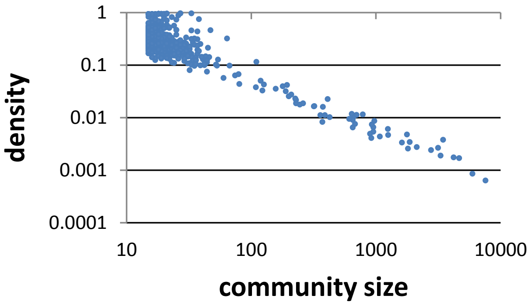
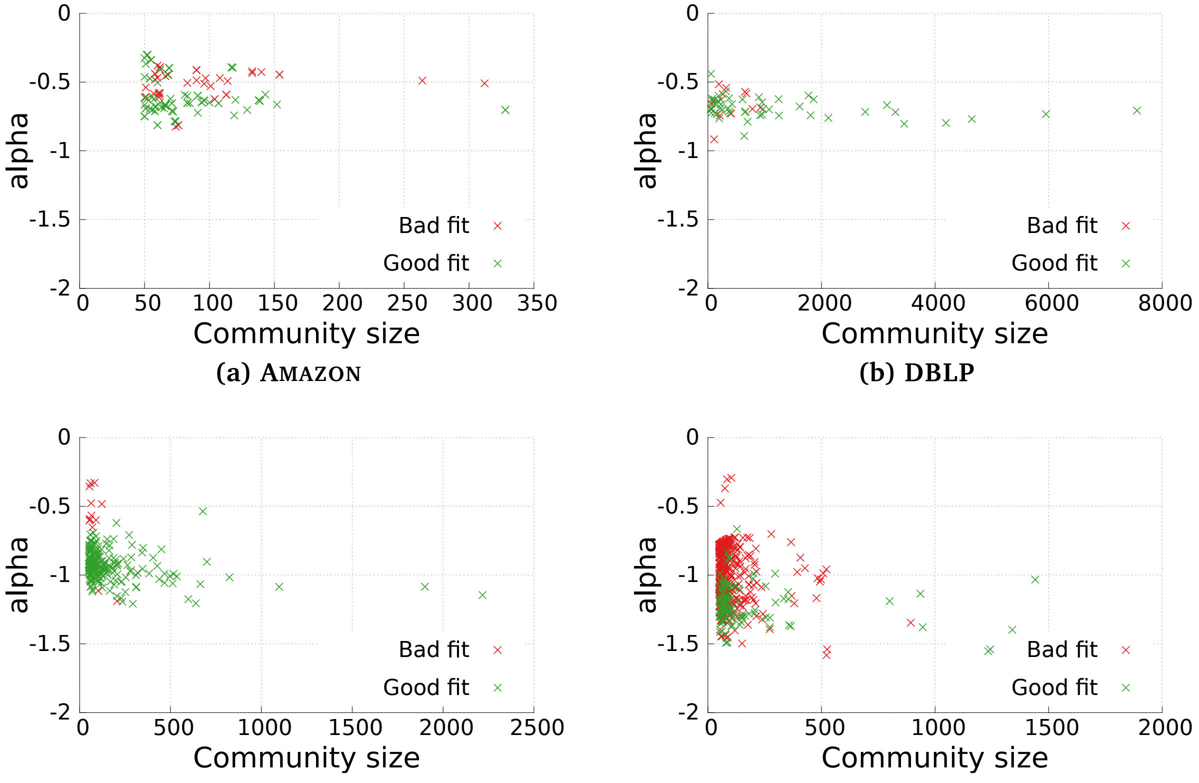


**Figure 3.2: Big communities are sparse: number of nodes vs density in the DBLP dataset.**



**(a) AMAZON** **(b) DBLP**

**(c) YOUTUBE** **(d) LIVEJOURNAL**

**Figure 3.3: Big ground-truth communities are hyperbolic: community size vs power-law exponent ($\alpha$). [Good fit: coefficient of determination ($r^2$) above 0.8.]**

sentation to be the most suitable. We hypothesize that nodes in big communities might play different roles and have different characteristics, in a process analogous to the differences between nodes in the global graph.

**Power-law degrees.** One well documented relationship in real networks is the power-law between the degree of a node and its rank (i.e. position in decreasing order of degree) [FFF99], which means the degree of a node $i$ can be approximated as $d_i = K \cdot p_i^{\alpha}$, where $\alpha$ is the power-law exponent, $K$ is the scaling factor correlated with the number of edges and $p_i$ is the rank of node $i$.

Our first hypothesis is that big communities follow a similar degree distribution. Figure 3.3 shows the calculated $\alpha$ values for different ground-truth communities in the 4 datasets. Communities have been marked according to their coefficient of determination ($r^2$) when we approximate the degree distribution within each community with a power-law. The power-law was approximated using a linear-regression in the log-log data and the coefficient was calculated using the same transformation. It can be seen, agreeing with intuition, that power-law degree distributions represent big communities fairly well. In fact, most of the ground-truth communities do not show uniform degree distribution (which imply $\alpha = 0$) but strongly skewed ones. Interestingly, $\alpha$ appears to decrease with community size (note the differences in the x-axis) and to be between -0.6 and -1.5 for communities with thousands of elements. Furthermore, as the frequently used uniform block model for communities indirectly assumes a uniform degree distribution, the power-law model necessarily achieves a better fit – the uniform model is a special case of the power-law model where $\alpha = 0$.

Some variations between the datasets are yet to be explained but can most likely be attributed to the different community definitions. For example, some communities with uniform degree distribution in the DBLP dataset are due to anomalies such as venues with a single paper creating artificial cliques (e.g. recording errors, conference proceedings with a single entry, workshops, etc.).

Again, we want to highlight that the observations made above are based on the communities which were externally provided for these datasets ("ground-truth communities") – not based on the results of a specific algorithm.

## 3.2 Hyperbolic Community Model

The previous analysis shows that, in order to detect big communities with realistic properties, models must be able to represent non-uniform degree distributions. In this section, we first propose HyCoM, a community model that assumes communities to have a power-law degree distribution. We then detail the MDL-based formalization that will guide the community discovery process and that is used as a metric for community quality.

### 3.2.1 Community Definition

We are given an undirected network consisting of nodes $\mathcal{N}$ and edges $\mathcal{E}$. We represent this network as an adjacency matrix $\mathrm{M} \in \{0,1\}^{|\mathcal{N}|\times|\mathcal{N}|}$. As an abbreviation, we use $N = |\mathcal{N}|$. The goal is to detect Hyperbolic Communities:

**Definition 1.** *Hyperbolic Community*
A hyperbolic community is a triplet $C = (S, \alpha, \tau)$ with $S = [S_1, .., S_{|S|}], S_i \in \mathcal{N}$ and $S_i \neq S_j$ if $i \neq j$, representing an ordered list of nodes, $\alpha \leq 0$ being the exponent when the degree distribution of the nodes is approximated by a power-law, and $0 \leq \tau \leq 1$ a threshold that determines the number of edges represented by the community.

Given the above triplet, and knowing that the nodes in $S$ are sorted by degree in the community, the degree of a node is $d_i \propto i^\alpha$. If we assume conditional independence given the community (i.e. we assume edge independence when we know both nodes belong to the current community), then the probability $p_{i,j}$ that the edge between nodes $i$ and $j$ is part of the community is also proportional to $i^\alpha \cdot j^\alpha$. Therefore, we can define the edges of a hyperbolic community to be the most probable edges given exponent $\alpha$ and threshold $\tau$:

$$E(C) = \{(S_i, S_j) \in S \times S \; : \; i^\alpha \cdot j^\alpha > \tau\}.$$

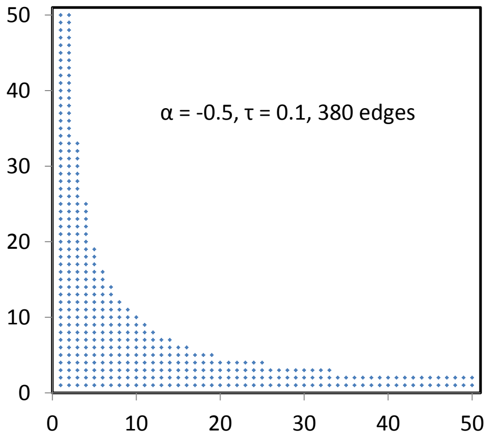


**Figure 3.4: Adjacency Matrix of a synthetic Hyperbolic Community.**

Figure 3.4 illustrates the adjacency matrix induced by the set $E(C)$ given a certain degree distribution and value of $\tau$. Its characteristic shape, a hyperbola, gave name to this model.

We propose to measure the importance of a community via the principle of compression, i.e. by its ability to compress the matrix **M**: if most edges of $E(C)$ are in fact part of **M**, then we can compress this community easily. Finding the most important communities will lead to the best compression of **M**.

More specifically, we use the MDL principle [Grü07]. We aim to minimize the number of bits required to simultaneously encode the communities (i.e. the model) and the data (effects not captured by the model, e.g. missing edges), in a trade off between model complexity and goodness of fit. In the following, we provide details on how to compute the description cost in this setting.

### 3.2.2 MDL Description Cost.

The first part of the description cost accounts for encoding the detected communities $\mathcal{C} = \{C_1, \ldots, C_n\}$ (where $n$ is part of the optimization and not a priori given). Each community $C_i = (S_i, \alpha_i, \tau_i)$ can be described by the list $S_i$, the number of bits used for $\alpha_i$, denoted as $k_{\alpha_i}$ [1], and by the number of edges $|E(C)|$ in the community. Please note that we actually do not need to encode the real-valued variable $\tau$, but it is sufficient to encode

[1] The number of bits does not affect the results as the previous term is significantly bigger. We use 32 bits in our experiments.

the natural number $|E(C)|$. The coding cost for a pattern $C_i$ is

$$L_1(C_i) = \log N + |S_i| \cdot \log N + k_{\alpha_i} + \log(|S_i|^2).$$

The first two terms encode the list of nodes: there are up to $N$ elements in the community and we can encode each element using $\log N$ bits. The third term encodes $\alpha_i$ and the last term encodes the number of edges in the community. Since the number of edges is bounded by $|S_i|^2$, we can encode it with $log(|S_i|^2)$ bits. Similarly, the set of patterns $\mathcal{C} = \{C_1, \ldots, C_l\}$ can be encoded by the following number of bits:

$$L_2(\mathcal{C}) = \log^* |\mathcal{C}| + \sum_{C \in \mathcal{C}} L_1(C).$$

Since the cardinality of $\mathcal{C}$ is not known a priori, we encode it via the function $\log^*$ using the universal code length for integers (see Chapter 2 for more details).

The second part of the description cost accounts for encoding the actual data given the detected communities. Since one might expect to find overlapping communities, we refer to the principle of Boolean Algebra and patterns are combined by a logical disjunction: if an edge occurs in at least one of the patterns, it is also present in the reconstructed data. More formally, we reconstruct the given matrix by:

**Definition 2.** *Matrix reconstruction*
Given a community $C$, we define an indicator matrix $\mathbf{I}^C \in \{0,1\}^{N \times N}$ (using the same ordering of nodes as imposed by $\mathbf{M}$) that represents the edges of the graph encoded by community $C$, i.e.

$$\mathbf{I}^C_{x,y} = 1 \Leftrightarrow (x,y) \in E(C).$$

Given a set of communities $\mathcal{C}$, the reconstructed network $\mathbf{M}^{\mathcal{C}}$ is defined as $\mathbf{M}^{\mathcal{C}} = \bigvee_{C \in \mathcal{C}} \mathbf{I}^C$ where $\vee$ denotes element-wise disjunction.

Since MDL requires a lossless reconstruction of the network and given that the matrix $\mathbf{M}^{\mathcal{C}}$ likely does not perfectly reconstruct the data, the second part of the description cost encodes the data given this model. Here, an 'error' might be either an edge appearing in $\mathbf{M}^{\mathcal{C}}$ but not in $\mathbf{M}$ or vice versa. As we are considering binary matrices, the number of errors can be computed based on the squared Frobenius norm of the residual matrix, i.e. $\left\|\mathbf{M} - \mathbf{M}^{\mathcal{C}}\right\|_F^2$.

Finally, as 'errors' correspond to edges in the graph, the description cost of the data can be computed as

$$L_3(\mathbf{M}|\mathcal{C}) = \log^* \left\|\mathbf{M} - \mathbf{M}^{\mathcal{C}}\right\|_F^2 + 2 \cdot \left\|\mathbf{M} - \mathbf{M}^{\mathcal{C}}\right\|_F^2 \cdot \log N.$$

**Overall model.** Given functions $L_2$ and $L_3$, we are now able to define the communities that minimize the overall number of bits required to describe the model and the data:

**PROBLEM DEFINITION 1. Finding hyperbolic communities**

- **Given** a matrix $\mathbf{M} \in \{0,1\}^{N \times N}$.
- **Find** hyperbolic communities, a set of patterns $\mathcal{C}^* \subseteq (\mathcal{P}(\mathcal{N}) \times \mathbb{R} \times \mathbb{R})^{|\mathcal{C}^*|}$ that **minimize**

$$\min_{\mathcal{C}} [L_2(\mathcal{C}) + L_3(\mathbf{M}|\mathcal{C})].$$

Computing the optimal solution to this problem is NP-hard, given that the column reordering problem in two dimensions is known to be NP-hard as well [JKC+04]. In the next section, we present an approximate but scalable solution based on an iterative processing scheme.

## 3.3 Proposed Method: HYCOM-FIT

We introduce HYCOM-FIT, a scalable and efficient algorithm that fits hyperbolic communities by approximating the optimal solution using an iterative method of sequentially detecting important communities. The general idea is to find in each step a single community $C_i$ that contributes the most to the MDL-compression based on local evaluation. That is, given the already detected communities $\mathcal{C}_{i-1} = \{C_1, \ldots, C_{i-1}\}$, we are interested in finding a novel community $C_i$ which minimizes $L_2(\{C_i\} \cup \mathcal{C}_{i-1}) + L_3(\mathbf{M}|\{C_i\} \cup \mathcal{C}_{i-1})$. Since $\mathcal{C}_{i-1}$ is given, this is equivalent to minimizing

$$L_1(C_i) + L_3(\mathbf{M}|\{C_i\} \cup \mathcal{C}_{i-1}). \tag{3.1}$$

Obviously, enumerating all possible communities is infeasible. Therefore, we perform the following steps in order to detect a single community $C_i$:

1. **Community candidates:** We spot candidate nodes by performing a rank-1 approximation of the matrix $\mathbf{M}$. This step provides a normalized vector with the score of each node.
2. **Community construction:** The scores from the previous step are used in a hill climbing search as a bias for connectivity, while minimizing the MDL costs is used as the objective function for determining the correct community size.
3. **Matrix deflation:** Based on the current community detected, we deflate the matrix so that the rank-1 approximation is steered to find novel communities in later iterations.

In the following, we detail each step of the iterative procedure.

**Community candidates.** As mentioned, exhaustively enumerating all possible communities is infeasible. Therefore we propose to iteratively let the communities grow. The challenge, however, is how to spot nodes which should be added to a community. For this purpose, we refer to the idea of matrix decomposition. Given the matrix $\mathrm{M}$ (or as we will explain in step 3, the deflated matrix $\mathrm{M}^{(i)}$), we compute a vector $\mathbf{a}$ such that $\mathbf{a} \cdot \mathbf{a}^T \approx \mathrm{M}$. The vector $\mathbf{a}$ reflects the community structure in the data and we treat the elements $a_i$ as an indication of the importance of node $i$ to this community.

**Community construction.** Given vector $\mathbf{a}$, Algorithm 1 overviews how a new community is built. We start by selecting an initial seed $\boldsymbol{S} = \{v_1, v_2\}$ of two connected nodes with high score in $\mathbf{a}$.[2] We then let the community grow incrementally: We randomly select a neighbor $v_i$ that is not currently part of the community, where the score vector $\mathbf{a}$ is used as the sampling bias. That is, given the current nodes $\boldsymbol{S}$, we sample according to

$$v_i \propto \begin{cases} a_i & v_i \notin \boldsymbol{S} \wedge \exists v' \in \boldsymbol{S} : (v_i, v') \in \mathcal{E} \\ 0 & else \end{cases} .$$

If the MDL score (cf. Equation 3.1) of the new community, i.e. using the vertices $\boldsymbol{S} \cup \{v_i\}$, is smaller than the MDL score using the previous community definition, vertex $v_i$ is accepted. Otherwise, a new sample is generated. This process is repeated until $\Delta$ consecutive rejections have been observed. Since the probability that an element that should have been included in the community but which was not sampled, i.e. $P$("i not selected"|"i should have been selected") decreases exponentially as a function of $\Delta$ and of its initial score, i.e. it can be bounded by $a_i^{\Delta}$, a small value of $\Delta$ is sufficient. In our experimental analysis, a value of $\Delta = 50$ gave good results for many settings and we recommend it as the default.

After the community growth stage, we try to remove elements from the community, once again minimizing the description cost. This alternating process is repeated until the community stabilizes. This process is guaranteed to converge as the description cost of matrix $\mathrm{M}$ is strictly decreasing.

**Matrix deflation.** While the first two steps build a single community $C_i$, the objective of this step is to transform the matrix so that the process can be iterated in such a way that we don't get the same community repeatedly. In particular, we aim at steering the rank-1 decomposition to novel solutions.

To solve this problem we propose the principle of matrix deflation. Starting with the original matrix $\mathrm{M}^{(1)} := \mathrm{M}$, after each iteration, we remove those edges which are already

[2] We tested different methods with no significant differences found in the results. Selecting the edge $(i, j)$ with highest $min(a_i, a_j)$ provides a good initial seed.

```
input  : $\boldsymbol{a}$ - score vector obtained from the decomposition
output : $\boldsymbol{S}$ - set of nodes of the community
$\boldsymbol{S} \leftarrow$ initialSeed($\boldsymbol{a}$)
repeat
    $t \leftarrow 0$
    while $t < \Delta$ do
        $v_i \leftarrow newBiasedNode(\boldsymbol{S}, \boldsymbol{a})$
        if $MDL(S \cup \{v_i\}) < MDL(S)$ then
            $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{v_i\}$
            $t \leftarrow 0$
        else
            $t \leftarrow t + 1$
        end
    end
    foreach node $n$ in $\boldsymbol{S}$ do
        if $MDL(\boldsymbol{S} \backslash \{n\}) < MDL(\boldsymbol{S})$ then
            $\boldsymbol{S} \leftarrow \boldsymbol{S} \backslash \{n\}$
        end
    end
until $S$ has converged
return $\boldsymbol{S}$
```

**Algorithm 1:** HYCOM-FIT: Community Construction

described by the detected community. That is, we obtain the recursion

$$\mathbf{M}^{(i+1)} := \mathbf{M}^{(i)} - \mathbf{I}^{C_i} \circ \mathbf{M}^{(i)} \qquad [= \mathbf{M} - \mathbf{M}^{\mathcal{C}_i} \circ \mathbf{M}\,]$$

where $\circ$ denotes the Hadamard product. As seen, matrix $\mathbf{M}^{(i+1)}$ incorporates all communities detected so far. Using the deflated matrix, our objective in Equation 3.1 is replaced by

$$L_1(C_i) + L_3(\mathbf{M}^{(i)}|\{C_i\}). \tag{3.2}$$

Overall, the algorithm might either terminate when the matrix is fully deflated, or when a pre-defined number of communities has been found, or when some other measure of community quality (i.e. size) has not been achieved in the most recent communities.

### 3.3.1 Fast MDL calculation

The key task of Algorithm 1 is to compute the MDL score (Equation 3.2) based on the current set of nodes $\boldsymbol{S}$. Besides set $\boldsymbol{S}$, estimating the number of bits requires determining the value of $\alpha$, a specific value for $\tau$ (or, equivalently, $|E(C)|$), and to count the number of errors made by the model. Since the MDL score is computed several times, we propose an efficient approximation for these tasks:

**Approximating the exponent of the degree distribution.** Exhaustive test of different approximation methods is beyond the scope of this work; for an in-depth analysis on power-law exponent estimation from empirical data we refer the reader to the review by Aaron Clauset et al. [CSN09]. The method chosen needs to be both robust in degenerate situations (e.g. uniform distributions) and efficient. We opted for a linear regression of the log-log data, as it not only respects both requirements, but also because it is known to over fit to the tail of the distribution and edges between high degree nodes are already expected under the independence assumption anyway.

**Number of edges and the value of $\tau$.** The value of $|E(C)|$ is defined as the number of edges between the nodes in $\boldsymbol{S}$, i.e. $|(\boldsymbol{S} \times \boldsymbol{S}) \cap \mathcal{E}|$, since this value can efficiently be obtained by an incremental computation each time a node is added or removed from the current community. Efficiency is ensured by indexing the edges in $\mathbf{M}$ by node.

Fixing the value of $|E(C)|$, we need to derive the value of $\tau$ leading to the desired cardinality. For efficiency, we exploit the following approximation:

**Lemma 2** *The value of $|E(C)|$ can be approximated by*

$$|E(C)| \approx (i_{start} - 1) \cdot |S| + \tau^{\frac{1}{\alpha}} \cdot (\log(i_{end}) - \log(i_{start})),$$

*where* $i_{start} := \max\{\lceil \tau^{\frac{1}{\alpha}} \cdot |\boldsymbol{S}|^{-1} \rceil, 1\}$ *and* $i_{end} := \min\{\lfloor \tau^{\frac{1}{\alpha}} \rfloor, |\boldsymbol{S}| + 1\}$.

**Proof 2** *Instead of exactly counting the number of elements* $i^\alpha \cdot j^\alpha > \tau$ *(cf. Figure 3.4), we do a continuous approximation by computing the area under the* $\tau$*-isoline (intuitively: the area shaded in Figure 3.4). More precisely, given a specific* $\tau$ *(and assuming* $\alpha \neq 0$*), we use the isoline derived by*

$$i^\alpha \cdot j^\alpha = \tau \Leftrightarrow j = \tau^{\frac{1}{\alpha}} \cdot i^{-1} =: f(i).$$

*Considering the integral* $\int_1^{|\boldsymbol{S}|+1} f(i)\, di$ *leads to an approximation of* $|E(C)|$*. To achieve a more accurate approximation, we consider two further improvements:*
*(a) For each* $i$ *with* $f(i) < 1$*, no edges are generated. Thus, we also don't need to consider the area under this part of the curve. It holds*

$$f(i) \geq 1 \Rightarrow i \leq \tau^{\frac{1}{\alpha}} \quad \Rightarrow i_{end} := \min\{\lfloor \tau^{\frac{1}{\alpha}} \rfloor, |\boldsymbol{S}| + 1\}.$$

*The integration interval can end at* $i_{end}$*.*
*(b) The number of edges for each node is bounded by* $|\boldsymbol{S}|$*. Thus, for each* $i$ *with* $f(i) > \boldsymbol{S}$*, we can restrict the function value to* $|\boldsymbol{S}|$*. It holds*

$$f(i) \leq |\boldsymbol{S}| \Rightarrow i \geq \tau^{\frac{1}{\alpha}} \cdot |\boldsymbol{S}|^{-1} \quad \Rightarrow i_{start} := \max\{\lceil \tau^{\frac{1}{\alpha}} \cdot |\boldsymbol{S}|^{-1} \rceil, 1\}.$$

*Thus, overall, given a specific* $\tau$*, the value of* $|E(C)|$ *can be approximated by*

$$\int_1^{i_{start}} |\boldsymbol{S}|\, di + \int_{i_{start}}^{i_{end}} f(i)\, di = (i_{start} - 1) \cdot |S| + \tau^{\frac{1}{\alpha}} \cdot (\log(i_{end}) - \log(i_{start})).$$

■

Based on Lemma 2, we find the appropriate $\tau$ by performing a binary search on the value of $\log \tau$ until the given value of $|E(C)|$ is (approximately) obtained. This step can be done in time $O(\log |\boldsymbol{S}|^2)$.

**Calculating the number of errors.** Determining the number of errors can be reduced to the problem of counting the number of existing edges in $\mathbf{I}^C$. In other words, the goal is to determine how many edges $(S_i, S_j) \in (S \times S) \cap \mathcal{E}$ fulfill $i^\alpha \cdot j^\alpha > \tau$. Knowing this number, e.g. denoted as $x$, the number of errors is given by

$$(\mathbf{M}^{(i)} - x) + (|E(C)| - x).$$

We have to encode all edges of $\mathrm{M}^{(i)}$ as errors which are not covered by $C$ (i.e. $\mathrm{M}^{(i)} - x$ many) and we additionally have to encode all non-existing edges which are unnecessarily included in $C$ (i.e. $|E(C)| - x$ many).

Obviously, the value of $x$ can be determined by simply iterating over all edges $(\boldsymbol{S}\times\boldsymbol{S})\cap\mathcal{E}$ of the community, i.e. linear in the number of edges.

### 3.3.2 Complexity analysis

**Lemma 3** HYCOM-FIT *has a runtime complexity of* $O(K\cdot(|\mathcal{E}|+|S|\cdot(\log|\boldsymbol{S}|^2+E)))$*, where* $K$ *is the number of communities we obtain,* $|\mathcal{E}|$ *is the number of edges in the network,* $|\boldsymbol{S}|$ *is the average size of a community and* $E$ *is the number of edges between the elements of* $\boldsymbol{S}$*.*

**Proof 3** *Steps 1 to 3 are repeated* $K$ *times, the number of communities to be obtained. Step 1, the rank-1 approximation, requires* $O(|\mathcal{E}|)$ *time. Step 2, the core of the algorithm, can be executed using* $O(|\boldsymbol{S}|)$ *additions and removals to the community, each with complexity* $O(\log|\boldsymbol{S}|^2+E)$ *as detailed in the previous sub-section. Finally, step 3, the matrix deflation, can be done in* $O(E)$ *with a single pass over the edges of the community.* ■

## 3.4 Experiments on Real Data

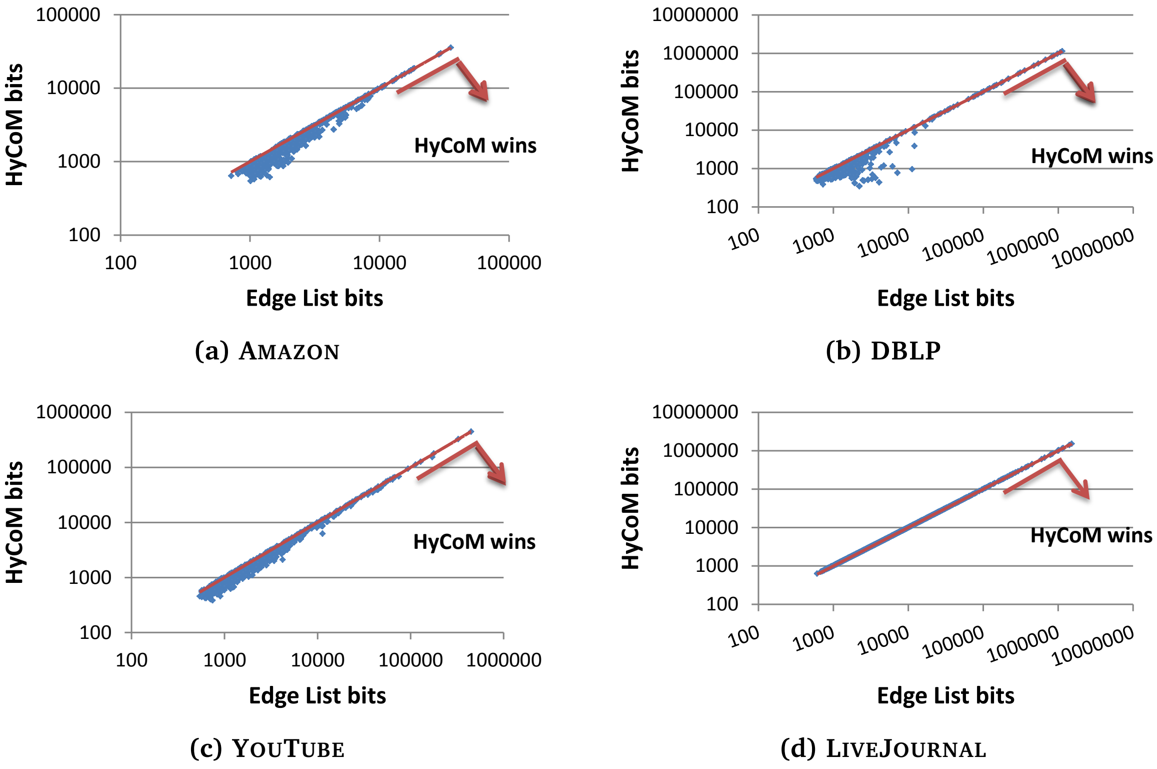
In this section, we start by evaluating the quality of the Hyperbolic Community Model using the datasets of Table 3.1. We subsequently evaluate HYCOM-FIT by studying its scalability and its ability to obtain empirically correct communities through the use of the node-labeled dataset.

We focus on three quality metrics: $Q1$) Model quality, $Q2$) HYCOM-FIT scalability and $Q3$) Effectiveness.

### 3.4.1 Q1 - Model quality

While Section 3.2 describes how to encode hyperbolic communities, it does not show whether this model is preferable over simpler models such as edge lists when encoding real communities. This aspect is not immediately clear because, even though block communities of uniform density are a special case ($\alpha = 0$) of hyperbolic, HYCOM explicitly encodes *missing* edges (i.e. errors made by the model). This observation implies that HYCOM must create dense hyperbolas to ensure that the overall cost of encoding the errors and the model is not higher than the cost of simply encoding all the edges in the graph. Since big communities are usually very sparse, it is not immediately obvious whether better compression can be achieved by this model.

Figure 3.5 shows the number of bits required to encode the ground-truth communities using the hyperbolic model and the edge-list format. In this scenario, the cost of each



(a) AMAZON

(b) DBLP

(c) YOUTUBE

(d) LIVEJOURNAL

**Figure 3.5: Number of bits required to encode ground-truth communities: HYCOM consistently requires less bits**

community using HYCOM can be obtained using the cost function of Problem 1 when setting $|\mathcal{C}| = 1$. As seen, the hyperbolic model consistently requires less bits to represent ground-truth communities. While savings are substantial for the datasets shown in (a)-(c), savings on the LIVEJOURNAL are less strong. In any case, though, compression based on hyperbolic structures is preferable.

### 3.4.2 Q2 - Scalability

We compared HYCOM-FIT to several popular community detection methods found in the literature: the community affiliation graph model [YL12a], clique percolation [APF+06] and cross-associations [CPMF04]. We obtained realistic graphs of different sizes by doing a weighted-snowball sampling[3] in the LIVEJOURNAL dataset.

[3] In this weighted-snowball sampling, weights correspond to the number of connections from a node to the current sample. This was done in an effort to preserve community structure.

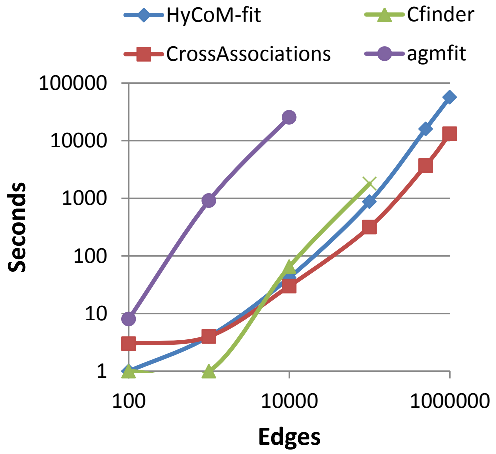


Figure 3.6: HYCOM-FIT scales linearly with the number of edges.

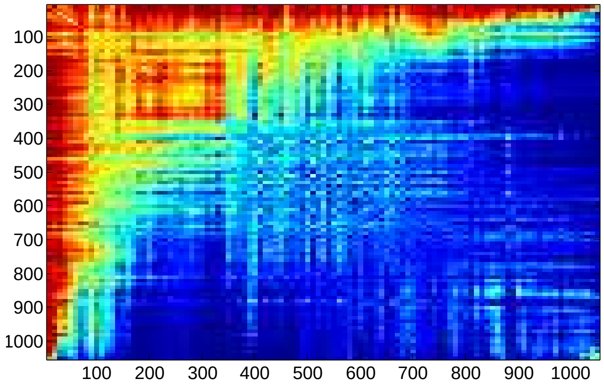


Figure 3.7: Anomalous community found by HYCOM-FIT in the LIVEJOURNAL data.

Figure 3.6 shows the run-time of the different algorithms using their default parameters. Clique percolation ran out of memory on a graph with 100 000 edges. HYCOM-FIT was run without any special stopping criteria (i.e. until the deflation was complete); as a consequence, bigger graphs required more communities to be fully deflated. HYCOM-FIT shows a fully linear run-time when the required number of communities is constant.

### 3.4.3 Q3 - Effectiveness

In addition to the datasets with ground-truth communities previously used, we also applied HYCOM-FIT to a copy of the simple-english Wikipedia pages from March 8, 2014. In this dataset, nodes represent articles and edges represent hyperlinks between them. Unlike previous datasets, we don't consider any ground-truth communities in the Wikipedia data; however, as nodes are labeled, this dataset allows us to assert the effectiveness of HYCOM-FIT.

**Detecting Hyperbolic Communities.** Figure 3.1a and Figure 3.1b presented in the introduction illustrate both a ground-truth community and a community found by HYCOM-FIT in the YOUTUBE dataset. They show not only the existence of hyperbolic communities in real data, but also the ability of our method to successfully find them. Note the similarity in the shape of both communities, which existing methods trying to find communities of uniform density would fail to detect.

**Anomaly Detection.** HYCOM-FIT is also able to detect anomalous structures in data. Figure 3.7 shows a detected community from the LIVEJOURNAL dataset. We

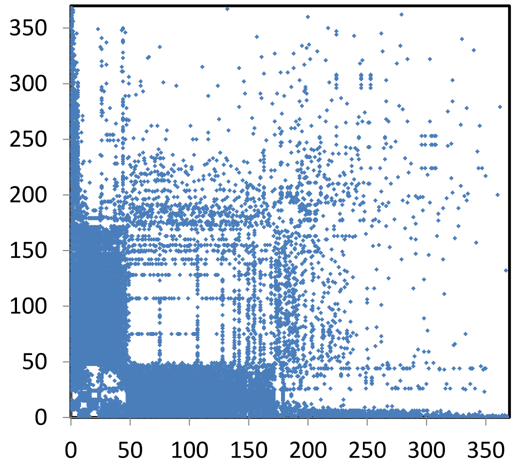


**Figure 3.8: Meaningful hyperbolic structures: community of dates in WIKIPEDIA.**

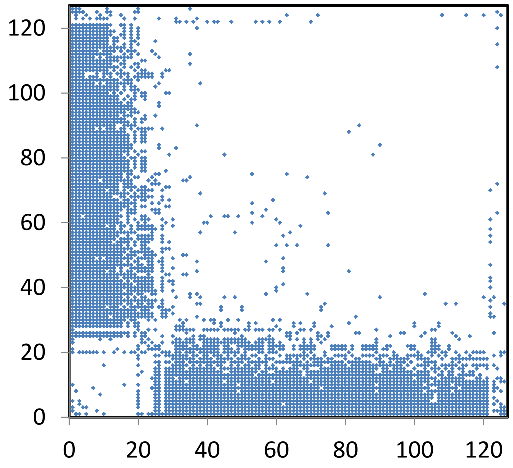


**Figure 3.9: HYCOM-FIT also finds bipartite cores and cliques: community of countries in WIKIPEDIA.**

can see the adjacency matrix (here represented as an heatmap) of suspiciously highly connected accounts. Approximately 1 000 users established over 300 000 friendship relations forming a very dense community (compared to a common distribution as shown in Figure 3.2). Nevertheless, this anomalous community also shows the characteristic shape of a hyperbola.

**Communities in WIKIPEDIA.** Figure 3.8 and Figure 3.9 show two communities detected in the WIKIPEDIA dataset. Figure 3.8 illustrates a hyperbolic community mostly consisting of *temporal* articles. The first 6 articles correspond to countries heavily mentioned in events (e.g. United States, France, Germany, etc.), we then have WIKIPEDIA articles corresponding to months (e.g. April, July), then articles representing individual years (e.g. 2002, 1973) and finally articles corresponding to particular dates (e.g. November 25, May 13).

Figure 3.9 shows HYCOM-FIT's generality and its ability to detect bipartite cores given their close resemblance to hyperbolas. In this community, the approximately 20 articles of highest degree represent articles with lists (e.g. "Country", "List of countries by area", "Members of the United Nations") while the remaining 140 articles are all individual countries.

## 3.5 Summary

We focused on the problem of representing communities in real graph data, and specifically on the resemblance between the structure of the full graph and the structure of big communities. We highlight our main contributions:

- **Hyperbolic Community Model:** We provide empirical evidence that communities in real data are better modeled using a hyperbolic model, termed HYCOM. Our model includes communities of uniform density as used by other approaches as a special case. We also show that this model is better from a compression perspective than previous models.
- **Scalability:** HYCOM-FIT is a scalable algorithm for the detection of communities fitting the HYCOM model. We leverage rank-1 decompositions and the MDL principle to guide the search process.
- **No user-defined parameters**: HYCOM-FIT detects communities in a parameter-free fashion, transparent to the end-user.
- **Effectiveness:** We applied HYCOM-FIT on various real datasets, where we discovered communities that agree with intuition.

HYCOM is available at http://cs.cmu.edu/~maraujo/hycom/.

Chapter based on work that appeared at PAKDD 2016 [ARF16].

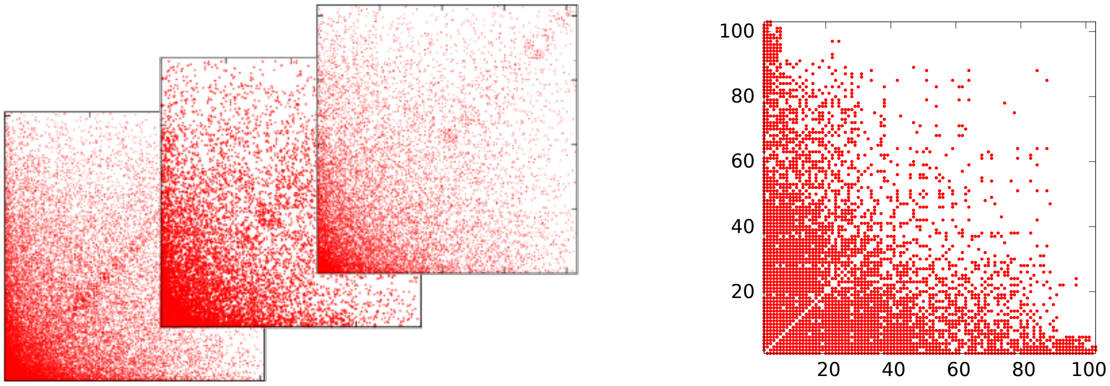
# Chapter 4

# Scalable Boolean Matrix Factorization

Given a boolean who-watched-what matrix, with rows representing users and columns representing movies, how can we find an interpretation of the data with low error? How can we find its underlying structure, helpful for compression, prediction and denoising? Boolean matrices appear naturally in many domains (e.g. user-reviews [BK07] or user-item purchases [VFM+14], graphs, word-document co-occurrences [DMM03] or gene-expression datasets [ZLD+10]) and describing the underlying structure of these datasets is the fundamental problem of community detection [For10] and co-clustering [TSS05] techniques.

We address the problem of finding a low-rank representation of a given $n \times m$ boolean matrix $\boldsymbol{M}$, with small reconstruction error while easily describing $\boldsymbol{M}$'s latent structure. We propose FastStep, a method for finding a non-negative factorization that, unlike commonly used decomposition methods, yields the best interpretability by combining a **boolean reconstruction** with **non-negative factors**. This combination allows FastStep to find structures that go beyond blocks, providing more realistic representations. Figure 4.1a showcases three communities (representing 3 venues) in the DBLP dataset that illustrate the important hyperbolic structures found in real data; compare them to the community found by FastStep in Figure 4.1b representing the American community in the `Airports` dataset.

Using our scalable method, we analyze two datasets of movie ratings and airports flights and show FastStep's interpretability power with intuitively clear and surprising decompositions. As an additional example, Figure 4.2 illustrates an application of FastStep to the task of community detection. Using route information alone, the world airports are decomposed in 10 factors that clearly illustrate geographical proximity. As we explain in more detail in Section 4.4.3, the communities we find have an arbitrary marginal and do not need to follow a block shape.



(a) DBLP real communities - PAKDD, KDD and VLDB.

(b) FastStep community - American airports.

**Figure 4.1: Realistic hyperbolic structure - Adjacency Matrices of real communities in `DBLP` and a community found by FastStep.**

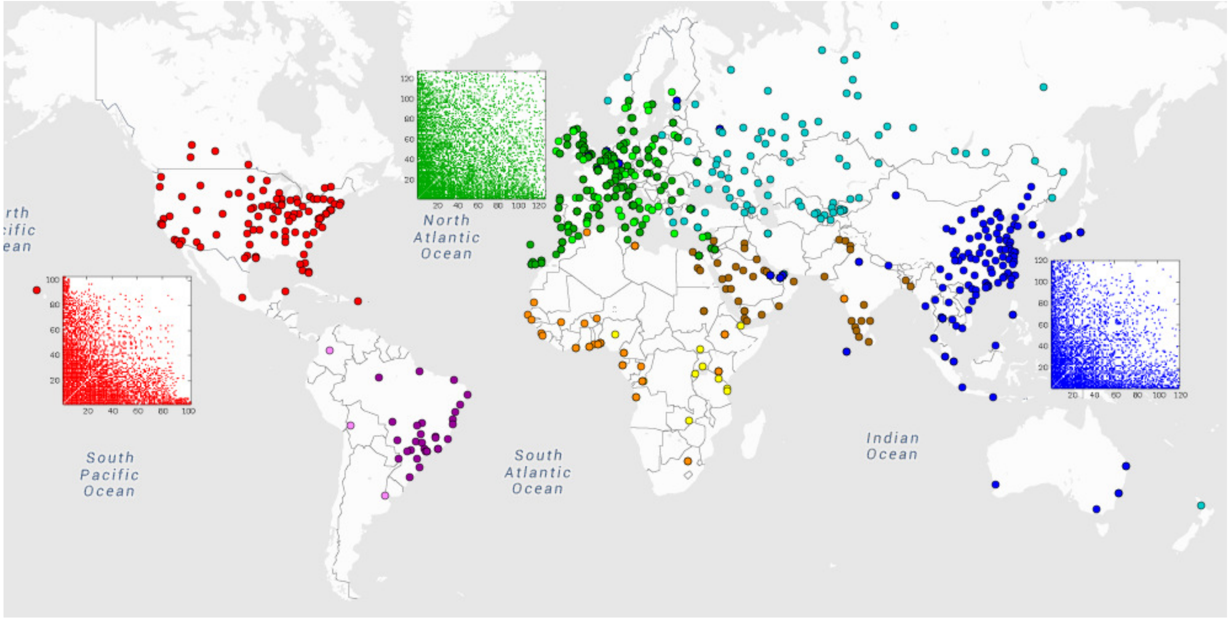


**Figure 4.2: Intuitive non-block communities - Communities automatically found in the `Airports` dataset from flight records. (best viewed in color)**

## 4.1 Problem Definition

There are two aspects for a strong interpretability of a boolean matrix decomposition: boolean reconstruction allows clear predictions and explanations of the non-zeros, while the existence non-negative factors establishes the importance of elements and enable the representation of beyond-block structures. In this section, we introduce a new formulation using a step operator that achieves both goals.

### 4.1.1 Formal Objective

**PROBLEM DEFINITION 2. Step decomposition**

- **Given** a $N \times M$ boolean matrix $\boldsymbol{M}$.
- **Find** a $N \times R$ non-negative matrix $A$ and a $M \times R$ non-negative matrix $B$, **so that** the product $\boldsymbol{AB}^T$ is a good approximation of $\boldsymbol{M}$ after thresholding:

$$\min_{\boldsymbol{A},\boldsymbol{B}} ||\boldsymbol{M} - u_\tau(\boldsymbol{AB}^T)||_F^2 = \sum_{i,j} \Big(\boldsymbol{M}_{ij} - u_\tau(\boldsymbol{AB}^T)_{ij}\Big)^2 \tag{4.1}$$

where $||\cdot||_F$ is the Frobenius norm and $u_\tau(\boldsymbol{X})$ simply applies the standard step function to each element $X_{ij}$:

$$[u_\tau(\boldsymbol{X})]_{ij} = \begin{cases} 1 & \text{if } \boldsymbol{X}_{ij} \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where $\tau$ is a threshold parameter.

As matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ can always be scaled accordingly, note that the choice of $\tau$ does not affect the decomposition.

### 4.1.2 Step Matrix Decomposition

The thresholding operator renders the objective function non-differentiable and akin to a binary programming problem. In order to solve it, we will approximate the objective function of Equation 4.1 by a function with similar objective:

$$\begin{aligned}&\min_{\boldsymbol{A},\boldsymbol{B}} F(\boldsymbol{A},\boldsymbol{B}) = \\ &= \min_{\boldsymbol{A},\boldsymbol{B}} \sum_{i,j} \log\left(1+e^{-\boldsymbol{M}_{ij}*\left(\sum_{k=1}^{r}\boldsymbol{A}_{ik}\boldsymbol{B}_{jk}-\tau\right)}\right)\end{aligned} \qquad (4.3)$$

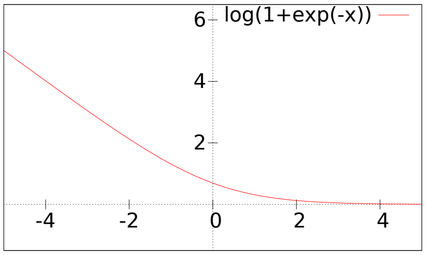


**Figure 4.3: Error function used: the error decreases quickly when non-zeros are correctly represented.**

where $\boldsymbol{M}$ was transformed so that it has values in $\{-1, 1\}$ by replacing all zeros with -1.

Note that $log(1+e^{-x})$ will tend to zero when $x$ is positive and it will increase when $x$ is negative; the intuition is that this error function will be approximately zero when $\boldsymbol{M}_{ij}$ and $(\sum_{k=1}^{r}\boldsymbol{A}_{ik}\boldsymbol{B}_{jk}-\tau)$ have the same sign and a linear penalty is in place whenever their signs differ.

## 4.2 Naive Approach: FASTSTEPNAIVE

Given the above formulation, there are several methods for finding $\boldsymbol{A}$ and $\boldsymbol{B}$ and one possibility is using gradient descent. The gradient is given by

**Lemma 4** *Let $\boldsymbol{S}_{ij} = \sum_{k=1}^{r}\boldsymbol{A}_{ik}\boldsymbol{B}_{jk}$, then the gradient of the objective function in Equation 4.3 is given by:*

$$\frac{\partial F}{\partial \boldsymbol{A}_{ik}} = \sum_{j=1}^{m}\frac{\boldsymbol{B}_{jk}}{1+e^{\tau-\boldsymbol{S}_{ij}}} - \sum_{j\in \boldsymbol{M}_i}\boldsymbol{B}_{jk} \qquad (4.4)$$

***Proof 4***

$$\frac{\partial F}{\partial \boldsymbol{A}_{ik}} = \frac{\partial\left(\sum_j \log\left(1+e^{-\boldsymbol{M}_{ij}*(\boldsymbol{S}_{ij}-\tau)}\right)\right)}{\partial \boldsymbol{A}_{ik}} = -\sum_j \frac{\boldsymbol{M}_{ij}\boldsymbol{B}_{jk}}{1+e^{\boldsymbol{M}_{ij}(\boldsymbol{S}_{ij}-\tau)}} =$$

$$= \sum_{j\notin \boldsymbol{M}_i}\frac{\boldsymbol{B}_{jk}}{1+e^{\tau-\boldsymbol{S}_{ij}}} - \sum_{j\in \boldsymbol{M}_i}\frac{\boldsymbol{B}_{jk}}{1+e^{\boldsymbol{S}_{ij}-\tau}} = \sum_{j=1}^{m}\frac{\boldsymbol{B}_{jk}}{1+e^{\tau-\boldsymbol{S}_{ij}}} - \sum_{j\in \boldsymbol{M}_i}\boldsymbol{B}_{jk}$$

■

The update rules for $\boldsymbol{B}$ are similar, given the symmetry of the problem.

Due to the non-negativity requirement, matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are projected after each iteration - this projection is made to a small value $\epsilon$ instead of directly to 0, as $\boldsymbol{A} = \boldsymbol{B} = 0$ would become stationary point of the objective function and the algorithm wouldn't improve.

Different gradient descent algorithms and small variations can now be tried. Algorithm 2 illustrates what our experiments indicate provides the quickest convergence: stochastic gradient descent with batches corresponding to factors, with an adaptive step size. Our results also indicate that initializing $\boldsymbol{A}$ and $\boldsymbol{B}$ to small random numbers provides the best results.

It should also be noted that, since the introduction of the approximation in Equation 4.3, $\tau$ now impacts the gradient, as the relative error $\left(\frac{\log(1+e^{\tau})}{\log(1+e^{0})}\right)$ of misrepresenting an element increases. However, it is clear that it should be chosen to be the highest possible value in order to improve convergence and to get a sharper decomposition, as long as numerical stability is not compromised. Our implementation uses $\tau = 20$ and our experiments showed that the outcome is not sensitive to the exact value.

### 4.2.1 Stopping Criteria

Intuition about the error function helps in order to decide which stopping criteria to use. With the suggested initialization, the initial error $F(\boldsymbol{A}(\boldsymbol{0}), \boldsymbol{B}(\boldsymbol{0}))$ is approximately $\tau E$, as each non-zero not represented in the reconstruction matrix (i.e. $\boldsymbol{S}_{ij} \approx 0$ when $\boldsymbol{M}_{ij} = 1$) corresponds to an error of approximately $\tau$.

Therefore, the absolute stopping criteria $\Delta$ is related to $\tau$, as $\Delta = c\tau$ corresponds to representing correctly $c$ new positions of the reconstructed matrix. On the other hand, the relative stopping criteria $\beta$ is well understood: $\beta \in [0, 1]$ represents the percentage of improvement and it can be shown that after $t$ successful iterations

$$F(\boldsymbol{A}(\boldsymbol{t}), \boldsymbol{B}(\boldsymbol{t})) \leq \beta^{t} F(\boldsymbol{A}(\boldsymbol{0}), \boldsymbol{B}(\boldsymbol{0})) \tag{4.5}$$

Finally, it is trivial to set the parameters in such a way that only one of them acts as a true stopping criteria. The appropriate value for $\Delta$ must depend on the number of non-zeros in $\boldsymbol{M}$ while a value of $\beta \leq 0.01$ provides a good measure of whether a plateau has been reached. In our experiments, less than $50$ iterations are sufficient when very good approximations are required, while less than $20$ iterations usually guarantees a relative improvement smaller than $1\%$.

```
input : $\boldsymbol{M}$ - $N \times M$ input matrix.
input : $r$ - Rank of the decomposition.
input : $\Delta$ - Absolute improvement threshold.
input : $\beta$ - Relative improvement threshold.
output: $\boldsymbol{A}$ - $N \times R$ factor matrix.
output: $\boldsymbol{B}$ - $M \times R$ factor matrix.

$\boldsymbol{A} : N \times R \leftarrow$ RandomReals(0, 0.1)
$\boldsymbol{B} : M \times R \leftarrow$ RandomReals(0, 0.1)
$\boldsymbol{step} : R \times 1 \leftarrow 1$
repeat
    $oldF \leftarrow F(\boldsymbol{A}, \boldsymbol{B})$
    for $k = 1$ to $R$ do
        $\boldsymbol{gradA} \leftarrow$ GradientA($\boldsymbol{M}$, $\boldsymbol{A}$, $\boldsymbol{B}$, $k$)
        $\boldsymbol{gradB} \leftarrow$ GradientB($\boldsymbol{M}$, $\boldsymbol{A}$, $\boldsymbol{B}$, $k$)
        $tries \leftarrow 0$
        repeat
            $tries \leftarrow tries + 1$
            $\boldsymbol{A'} \leftarrow \boldsymbol{A}$
            $\boldsymbol{B'} \leftarrow \boldsymbol{B}$
            for $i = 1$ to $N$ do
                $A'_{ik} \leftarrow \max(A_{ik} -$ step[k]*gradA[$i$]$, \epsilon)$
            end
            for $j = 1$ to $M$ do
                $B'_{jk} \leftarrow max(B_{jk} -$ step[k]*gradB[$j$]$, \epsilon)$
            end
            $newF \leftarrow F(\boldsymbol{A}, \boldsymbol{B})$
            if $newF \geq oldF$ then
                $step_k \leftarrow step_k/2$
            end
        until $(newF < oldF) \vee (tries \geq 10)$
        $\boldsymbol{A} \leftarrow \boldsymbol{A'}$
        $\boldsymbol{B} \leftarrow \boldsymbol{B'}$
        $step_k \leftarrow step_k * 2$
    end
until $(oldF - newF < \Delta) \vee \left(\frac{oldF - newF}{oldF} < \beta\right)$
return $\boldsymbol{A}, \boldsymbol{B}$
```

**Algorithm 2:** FASTSTEP: Gradient Descent

### 4.2.2 Complexity

A straightforward implementation would take $O(TNMR^2)$ time where $T$ is the number of iterations, $N$ and $M$ are the dimensions of the matrix and $R$ is the rank of the decomposition. However, by using additional $O(NM)$ memory, caching and updating $\boldsymbol{S}$ in each iteration, it can be reduced to $O(TNMR)$.

## 4.3 Proposed Method: FastStep

Unfortunately, the previous algorithm is not adequate for many datasets given its quadratic nature; it grows linearly in $O(NM)$. In many scenarios such as community detection and recommender systems, $\boldsymbol{M}$ is extremely sparse and algorithms must be linear (or quasilinear) in the number of non-zeros ($E$). In the following, we describe how to quickly approximate $F(\boldsymbol{A}, \boldsymbol{B})$ and the respective gradients of the sparse matrix.

### 4.3.1 Fast Gradient Calculation

As shown in Equation 4.4, calculating the gradient exactly requires $O(NM)$ operations per factor because each $\boldsymbol{A}_{ik}$ requires a summation over all elements $\boldsymbol{B}_{jk}$. Furthermore, there is a $\boldsymbol{A}_{ik}\boldsymbol{B}_{jk}$ term in $\boldsymbol{S}_{ij}$, which means that this loop cannot be easily unrolled or reused between elements of $\boldsymbol{A}$. The goal of this subsection is to approximate the gradient of the factor using a number of operations in the order of $O(E)$, the number of non-zeros in the matrix.

Careful analysis of the structure of this summation in the gradient allows us to quickly approximate it. The impact of position $(i, j)$ in factor $k$ is a sigmoid function, scaled by $\boldsymbol{B}_{jk}$ and with parameter $\boldsymbol{S}_{ij}$. This means that only positions with simultaneously high $\boldsymbol{S}_{ij}$ and $\boldsymbol{B}_{jk}$ have a significant impact on the gradient, which implies that we should first consider pairs $(i, j)$ with high $\boldsymbol{A}_{ik}\boldsymbol{B}_{jk}$, as that correlates well with both metrics.

Hence, Equation 4.4 can be approximated as

$$\frac{\partial F}{\partial \boldsymbol{A}_{ik}} \simeq \sum_{(i,j)\in P(t)} \frac{\boldsymbol{B}_{jk}}{1 + e^{\tau - \boldsymbol{S}_{ij}}} - \sum_{j \in \boldsymbol{M}_i} \boldsymbol{B}_{jk} \tag{4.6}$$

where $P(t)$ is the set of elements of $\boldsymbol{M}$ that the decomposition "believes" should be reconstructed, i.e. with high $\boldsymbol{A}_{ik}\boldsymbol{B}_{jk}$ for some $k$. We define $R$ sets of elements $P_k(t)$ that each factor $k$ would like to reconstruct and $P(t) = \bigcup P_k(t)$. The intuition is that, initially, only non-zeros contribute to the gradient so we can quickly calculate it with no error using the second summand of Equation 4.6. As we iterate, the error will gradually move from

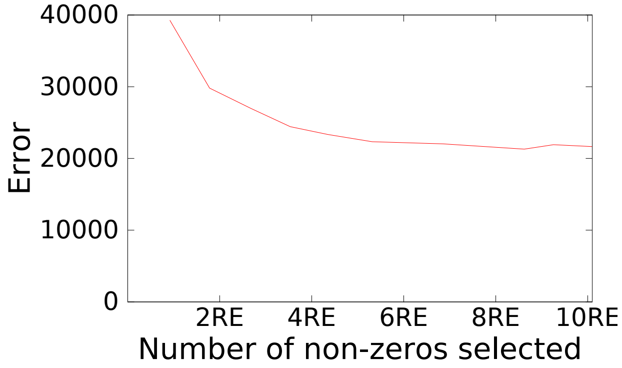


**Figure 4.4: A small number of non-zeros approximates the gradient – quick convergence in the `Airports` dataset.**

the non-zeros of $\boldsymbol{M}$ to some of the zeros. However, given $\boldsymbol{M}$'s sparsity and the symmetry of the error function – the error of misrepresenting a one is the same as misrepresenting a zero – $|P(t)|$ can be kept small and in the order of $O(RE)$; Figure 4.4 shows the error as the size of $P(t)$ increases in the AIRPORTS dataset.

In order to quickly find the top-$T$ pairs $(i, j)$ with highest $\boldsymbol{A}_{ik}\boldsymbol{B}_{jk}$, let $\boldsymbol{a_k}$ and $\boldsymbol{b_k}$ be columns $k$ of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. After sorting $\boldsymbol{a_k}$ and $\boldsymbol{b_k}$, the biggest $\boldsymbol{A}_{ik}\boldsymbol{B}_{jk}$ not currently in $P_k$ can be selected from a very small set of elements along one sort of "diagonal" in the matrix. In particular, it can be shown that element $(x, y)$ should not be added to $P_k$ before both $(x-1, y)$ and $(x, y-1)$ are added, as they would necessarily be at least as big. Therefore, one can keep a priority queue with $O(min(N, M))$ elements and it is possible to select a set of $t$ non-zeros and approximate the gradient of all elements in factor $k$ in $O(T + N\log N + M\log M)$ operations. In Chapter 8, we describe and provide additional theoretic guarantees using a similar algorithm that works in tensors.

### 4.3.2 Fast Error Function Evaluation

Given the method currently used to quickly calculate the gradient, one could consider evaluating the error function at positions $E + P(t)$. Although fast, some positions of the matrix would never be considered and the algorithm would over-fit, thus it cannot be used to detect convergence.

Therefore, in order to detect convergence, after each iteration of the gradient descent procedure (i.e. after all the factor updates are completed), we estimate the error $\tilde{F}(\boldsymbol{A}, \boldsymbol{B})$ by considering all the non-zeros and a uniform sample of the zeros of the matrix, and then scale the error accordingly. Additionally, in order to decrease the probability of underestimating $F(\boldsymbol{A}, \boldsymbol{B})$ and compromising future iterations of the gradient descent, we

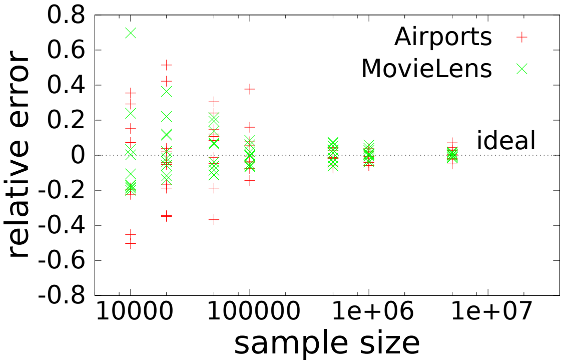


**Figure 4.5: Using less than 1% of the samples produces a very good approximation.**

take the median of 9 simulations.

Note that $\mathbb{E}[\tilde{F}(\boldsymbol{A}, \boldsymbol{B})] = \mathbb{E}[F(\boldsymbol{A}, \boldsymbol{B})]$, as $\tilde{F}(\boldsymbol{A}, \boldsymbol{B})$ is a simple uniform sampling of terms of the error function. It is known that the variance $Var[\tilde{F}(\boldsymbol{A}, \boldsymbol{B})]$ decreases linearly with the number of samples. Figure 4.5 shows the impact of the sample size when approximating $F(\boldsymbol{A}, \boldsymbol{B})$ in the AIRPORTS and MOVIELENS10M_1980 datasets (more details about the datasets in the next section) after decomposing each matrix in 10 factors. Note that, while the second dataset has $41\times$ more non-zeros and its matrix is $12\times$ bigger, the accuracy in respect to the number of samples does not change significantly. This can also be attributed to the fact that the error function is upper-bounded at each position by $\log(1 + e^{\tau}) \approx \tau$, so the variance is small.

### 4.3.3 Complexity

**Lemma 5** FASTSTEP *has a time complexity bounded by the number of nonzeros,*

$$O(TR(E + P\log(\min(N, M)) + N\log N + M\log M + S)),$$

*where, as before, $T$ represents the number of iterations, $N$ and $M$ are the dimensions of the matrix and $R$ is the rank of the decomposition. $P = |P(t)|$, which as we showed can be $O(RE)$, and $S$ represents the number of samples used to check for convergence.*

***Proof 5*** *The gradient descent loop iterates $T$ times. On each iteration, we check for convergence and calculate the gradients:*

- *We check for convergence in $O(R(S + E))$ time, as we evaluate the error function at $S$ distinct zero locations.*
- *We calculate the gradient by, firstly, sorting each factor of $\boldsymbol{A}$ and $\boldsymbol{B}$ in time $O(RN\log N)$ and $O(RM\log M)$, respectively. Then, we update the gradients using the $O(RP)$ biggest*

**Table 4.1: Datasets used to evaluate FastStep.**

| Name | Size | Non-zeros | Description |
|---:|---:|---:|:---|
| `MovieLens100k` | 945×1 684 | 100 000 | User-movie ratings. |
| `MovieLens10m` | 71 568×10 681 | 10 000 055 | User-movie ratings. |
| `Airports` | 7 733×7 733 | 34 660 | Airport to airport flight information. |

*elements of the reconstruction and each one can be obtained in $O(\min(M, N))$ time. Summing the above-mentioned big-$O$ bounds, we get the desired complexity.* ■

Using the same notation as before, the time complexity is now bounded by the number of non-zeros, $P = |P(t)|$ (which as we showed can be $O(rE)$) and the number of samples $S$ to check for convergence. The complexity is now $O(TR(E + P \log(\min(N, M)) + N \log N + M \log M + S))$.

### 4.3.4 Obtaining clusters from A and B

When a binary answer on whether a given element "belongs" to a factor is desired (e.g. community detection), a clear interpretation exists solely based on the principles of the decomposition:

**Definition 3.** *Part of a factor*
A row element $i$ belongs to a factor $k$ if there is a non-zero in the reconstructed matrix at row $i$ and if this factor contributed with a weight above $\frac{\tau}{r}$, i.e.:

$$A_{ik} \geq \frac{\tau}{R \max(\boldsymbol{b_k})} \text{ and } \boldsymbol{S}_{i,\arg\max(\boldsymbol{b_k})} \geq \tau.$$

We show that this method generates empirically correct clusters in the next section.

## 4.4 Experimental Evaluation

FastStep was tested on 2 fairly different real-world datasets, see Table 4.1 for details. `MovieLens100k` and `MovieLens10m` are user-movie ratings datasets made available by `MovieLens` and the `Airports` dataset is a graph made available by `OpenFlights`. Unless otherwise specified, FastStep was run using the default parameters defined in the previous section and 1 000 000 samples.

We answer the following questions:

**Q1.** How **scalable** is FASTSTEP?

**Q2.** How does the **reconstruction error** compare to other methods?

**Q3.** How **effective** and **interpretable** is the FASTSTEP decomposition?

### 4.4.1 Q1 - Scalability

**Table 4.2: Number of non-zeros in the `MovieLens10m` dataset when only considering movies released before a given decade.**

| Year | 1930 | 1940 | 1950 | 1960 | 1970 | 1980 | 1990 |
|---|---|---|---|---|---|---|---|
| Non-zeros | 14 673 | 110 206 | 258 253 | 498 451 | 850 362 | 1 447 301 | 3 077 991 |

The fast approximation of the gradient has a runtime proportional to the number of non-zeros of the matrix. For the runtime to be reproducible, we took different subsets of the `MovieLens10m` dataset by removing all the ratings of movies produced after a given decade. Please note that the matrix was not resized, resulting in columns (and possibly rows) full of zeros.

Figure 4.6 shows the execution time of the decomposition for these different matrices. Notice the sub-quadratic running time.

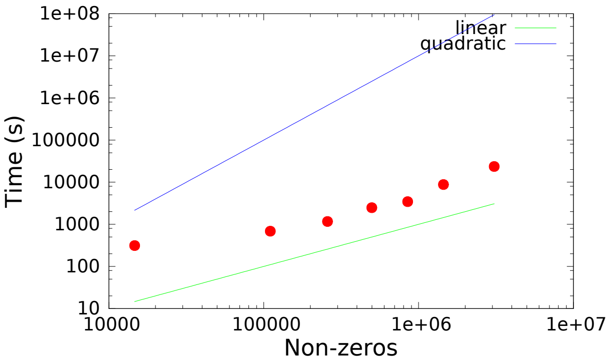


**Figure 4.6: Scalability: the FASTSTEP decomposition has linear running time on the number of non-zeros.**

### 4.4.2 Q2 - Low Reconstruction Error

When considering the same number of factors, a lower reconstruction error implies better compression and potentially enables lower-rank representations of the data. Given the boolean nature of $\boldsymbol{M}$, the error function is intuitively easy to represent. Let $\boldsymbol{M}$ represent the original dataset and $\boldsymbol{R}$ represent the reconstructed matrix, then the error $E$ is given by $E = ||M - R||_F^2$.

We compare FASTSTEP to other methods that are quasilinear in the number of non-zeros. Table 4.3 compares the squared error of FASTSTEP, SVD, NNMF and HYCOM in the `MovieLens100k` and `Airports` datasets when using 10 factors. For SVD and NNMF, as

**Table 4.3: Reconstructions using the FASTSTEP decomposition achieve lower squared error than popular scalable methods.**

| Dataset | FASTSTEP | SVD | NNMF | HYCOM |
|---|---|---|---|---|
| `Airports` | **21 206** | 26 061 | 27 235 | 29 117 |
| `MovieLens100k` | **68 863** | 70 627 | 74 040 | 86 964 |

arbitrary values such as 0.5 do not guarantee the lowest error, we tried all thresholds and considered the optimal (Figure 4.7 illustrates this process). For FASTSTEP, we selected the lowest error from Figure 4.4 and its equivalent in the `MovieLens100k` data (which converged after considering only $2rE$ non-zeros). For HYCOM, we considered as error the sum of the edges not represented and the mistakes made inside each community. Among the state of the art methods, we did not compare with non scalable algorithms (L-PCA, ASSO, BMF-THRESHOLD).

However, while comparing the reconstruction error of these methods might be appropriate given the same number of parameters, their expressiveness is not the same given their different characteristics. In this regard, by allowing negative numbers, SVD is at an advantage when compared to the rest of the methods. Please note that common techniques such as the Bayesian Information Criterion (BIC) [S+78] or the Akaike Information Criterion (AIC) [Aka74] would not provide a fairer comparison, as all methods would keep the same relative *rank* given that the number of parameters is the same. Techniques such as Minimum Description Length (MDL) [Grü07] measure the number of bits required to encode both the error and the model, but it is not clear which method should be used to represent real numbers, especially given that the importance of the bits is not the same - as a result, methods such as HyCoM that rely on integer values would greatly benefit.

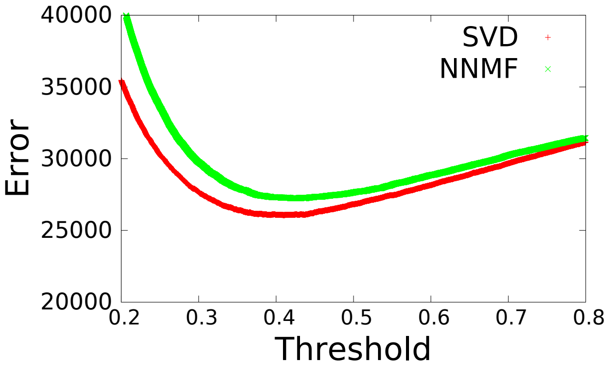


**Figure 4.7: Competitors need to compute the threshold and naively require quadratic time.**

We can see that FASTSTEP is able to simultaneously achieve a lower reconstruction error while maintaining higher interpretability.

**Table 4.4: FASTSTEP is able to automatically group similar movies in the MovieLens dataset. Groups manually labeled according to their highest scoring movies.**

| "Action" | "Romance" | "Drama" |
|---|---|---|
| Raiders of the Lost Ark | Picture Perfect | Titanic |
| The Empire Strikes Back | Addicted to Love | Wag the Dog |
| Terminator 2: Judgment Day | Bed of Roses | L.A. Confidential |
| The Terminator | My Best Friend's Wedding | Jackie Brown |
| Star Trek 3: The Search for Spock | Fly Away Home | Replacement Killers |

### 4.4.3 Q3 - Discoveries

#### MovieLens

The MovieLens100k user-movie dataset was decomposed using a rank-10 decomposition and the factors were clustered as described. Table 4.4 illustrates the top-5 movies (ranked by score) in three of the factors and shows a grouping by movie theme.

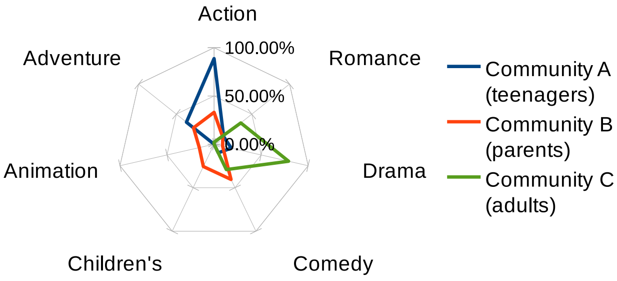


**Figure 4.8: FASTSTEP successfully groups movies by genre.**

Figure 4.8 shows 3 clusters and the percentage of movies in each cluster that correspond to a given genre (movies might have more than one tag, so genres do not sum to 1). We labeled group A as *teenagers* due to the clear prevalence of Action and Adventure movies. In group B, most of the movies rated were in categories of Comedy, Children's, Animation and Adventure; we hypothesize that users rating these movies are parents and labeled the group accordingly. Finally, we labeled group C as *adults* due to the Drama, Comedy and Romance movie genres.

#### Airports

The Airports dataset is an undirected graph where nodes represent airports and edges represent flights connecting them. As both dimensions represent the same entity, we decided to search for communities by enforcing $\boldsymbol{B} = \boldsymbol{A}$. Both the minimization problem $\left(\min_{\boldsymbol{A}} ||\boldsymbol{M} - u(\boldsymbol{A}\boldsymbol{A}^T)||_F^2\right)$ and the gradient are similar.

Figure 4.2 shows a geographical plot of the airports in the different communities;

some big hubs, such as Frankfurt and Heathrow, appear in multiple communities and were coded with a single color to simplify visualization. Even though no geographical information was used to perform this task, there is a very clear distinction between north American airports, Brazilian airports, European airports, previous French colonies in Africa, Russian airports, Middle-Eastern airports and south-east Asia airports. Additionally, in order to illustrate one of the surprising findings, Figure 4.9 highlights the two European communities (in blue and yellow) along with the overlapping airports (in green). While it would initially seem that all these airports should be considered the same community, a quick overview makes us realize that they are in fact divided by "major airports" and "secondary airports", usually operated by low-cost companies. The airports with the highest score in the "major airports" community are Barcelona, Munich and Amsterdam, while the airports with the highest score in the "low-cost" group are Girona (85km from Barcelona), Weeze (70km from Dusseldorf) and Frankfurt-Hahn (120km from Frankfurt). We consider these and other surprising findings to be very strong empirical evidence on FASTSTEP's usefulness for these tasks.

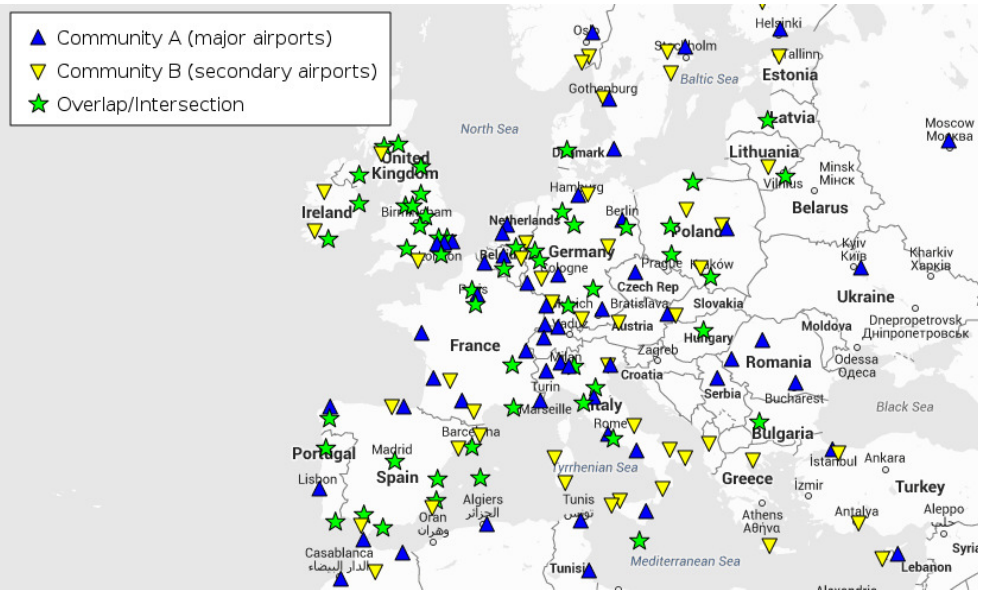


**Figure 4.9: Intuitive split of European airports - FASTSTEP identifies 2 European communities, one with the major international airports (in blue) and the other with secondary (low price) airports (in yellow). Airports that belong to both groups appear in green.**

Another important improvement of the FASTSTEP decomposition is its ability to reconstruct non-block clusters in the data. Figure 4.1b shows the adjacency matrix of the American community found in the previous decomposition. As we have non-negative factors, lets explore the additional information available in matrix $\boldsymbol{A}$. The airports with the highest score correspond to central airports in continental United States with hubs from big airlines: Minneapolis, Denver, Chicago, Dallas, Detroit, Houston, etc. Therefore, using this decomposition alone, measures of centrality can be directly obtained.

Finally, when analyzing the scores of the elements in the communities, we can see that they closely follow a power-law when sorted in decreasing order. The hyperbolic shape of ground-truth communities has been dully described in the previous chapter; nevertheless, note that no power-law bias was introduced in FASTSTEP. We consider this a strong indicator of its ability to detect realistic structures in graph data.

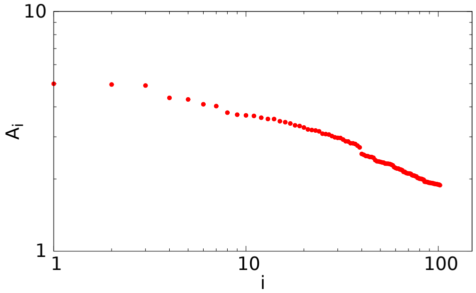


**Figure 4.10: Scores of the airports in the American community - the power-law shape matches ground-truth communities.**

## 4.5 Related Work

An important aspect of fast decomposition methods is their ability to evaluate the reconstruction error $||\boldsymbol{M} - \boldsymbol{R}||_F^2$ in less than quadratic time. Leskovec et al. [LCK$^+$10] approximated the log-likelihood of the fit by exploiting the Kronecker nature of their generator. In the Compact Matrix Decomposition [SXZF07], Jimeng Sun et al. approximated the sum-square-error (SSE) by sampling a set of rows and columns and scaling the error in the submatrix accordingly.

Table 4.5 provides a quick comparison of some of the methods discussed in this section. We characterize as *Beyond-blocks* methods who do not force a rectangular tiling of the data. Methods with the *Boolean Reconstruction* characteristic do not force post-processing of the reconstructed matrix in order to decide whether a non-zero should be considered. *Arbitrary Marginals* refers to a method's ability to represent any marginal in the data (e.g. rectangles, but also triangles or hyperbolic structures). We define *interpretability* as the ability to easily select a subset of elements representing a factor.

## 4.6 Summary

FASTSTEP carefully combines a non-negative decomposition and a boolean reconstruction for the best interpretability of the data. We have shown that it achieves lower recon-

**Table 4.5: Comparison of decomposition methods - FastStep combines interpretability and beyond-block structures for large datasets.**

| | FastStep | SVD | NNMF | ASSO | THRESH | HyCoM | L-PCA |
|---|---|---|---|---|---|---|---|
| **Scalability** | ✔ | ✓ | ✓ | | | ✓ | |
| **Overlapping** | ✔ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Beyond-blocks** | ✔ | ✓ | ✓ | | | ✓ | ✓ |
| **Boolean Rec.** | ✔ | | | ✓ | ✓ | ✓ | ✓ |
| **Arbitrary Marginals** | ✔ | ✓ | ✓ | | | | ✓ |
| **Interpretability** | ✔ | | | ✓ | ✓ | ✓ | |

struction error than similar methods and have provided strong empirical evidence of its ability to find structural patterns in the data. The main contributions of this work are the following:

1. **New formulation and tractable approximation:** We introduce a novel FastStep Decomposition which exploits thresholding of the reconstructed data in order to minimize the reconstruction error.
2. **Scalable:** A very efficient approximation enables a runtime linear in the number of non-zeros.
3. **Low reconstruction error** when compared to standard methods.
4. **Realistic representation** which relates to nodes in clusters or degree inside communities.
5. **Meaningful and interesting discoveries** in real-world datasets.

**Reproducibility** Available at http://cs.cmu.edu/~maraujo/faststep/.

Chapter based on work that appeared at SDM 2017 [AAF+17].

# Chapter 5

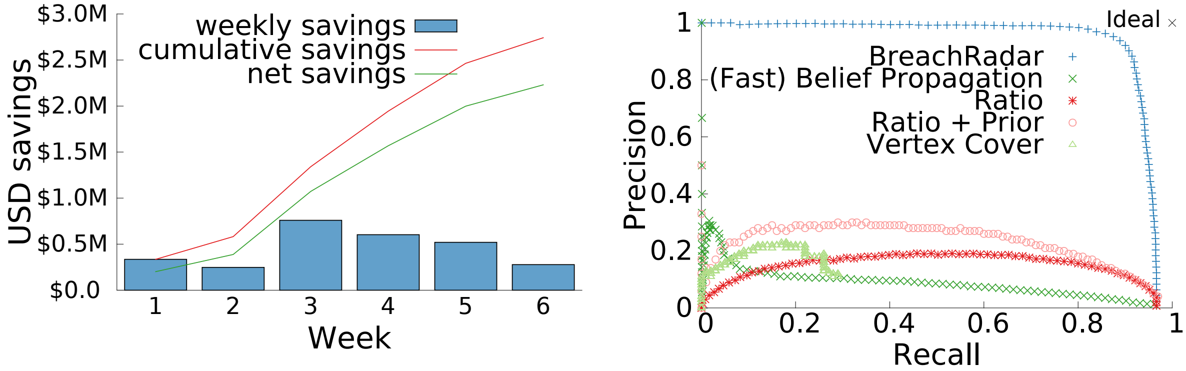
# Detecting Points-of-Compromise

As society turns away from cash as form of payment, the possibility for payment data to be compromised in a transaction increases. A recent Bank of Canada survey using data from 2013 reports that cash transactions comprise only 43.9% by volume and 23.1% by value out of all transactions [BFS15]; Bagnall et al. report similar results for other countries [BBH+14]. FICO estimates European losses to fraud at €1.6 billion in 2014 [Cor15] and global fraud in 2013 was estimated at $13.9 billion [Res15]. If the cost of lost merchandise as well as redistribution costs are included, fraud is estimated to account for 1.32% of the revenue for the average merchant in the US and it's higher if the merchant operates globally [Sol15].

A Point-of-Compromise (POC) is a (physical or virtual) location of the payment network, such as an ATM or a point-of-sales terminal, that processed or collected payment information and that was compromised by fraudsters. In a classical scenario, the victim's card is swiped in a small rogue device (possibly installed in an ATM or vending machine, or used by malicious employees whenever the card leaves the owners' sight at a bar, restaurant or gas station) that reads and stores the magnetic stripe data which is then, e.g., sold and written on a new cloned card. However, this is not the only scenario: data breaches are also common POCs which might occur at the merchant or even payment-processor level.

Given the increase in both the number of data breaches and in the number of cards affected (Target's 2013 data breach alone exposed an estimate of 40 million cards [Kre14]), early and accurate detection of POCs is not only vital for fraud prevention, but could also lead to a decrease in the expected loss from these breaches, reducing their frequency. The timely discovery of a Point-of-Compromise could prevent the fraudulent use of other cards compromised at the same location and early detection could prevent thousands of fraud cases.

As an example, Figure 5.1a illustrates the weekly savings when the proposed BreachRadar algorithm is applied to one of the two real datasets we explore. By automatically canceling cards that were used in locations marked as potentially compromised, and even after assuming a $10 reissue cost per card, our system would be able to prevent over $2 million USD in credit-card fraud in a period of just 6 weeks.



**(a) Significant estimated savings, over a period of 6 weeks (red), even assuming a $10 reissue cost per card (green).**

**(b) Almost perfect precision and recall, significantly higher than competing methods.**

**Figure 5.1: BreachRadar's effectiveness and comparison to other methods.**

The main contributions of this chapter can be summarized as follows:

1. **Point-of-Compromise Problem.** We formulate a novel and important POC detection problem.
2. **Effectiveness.** BreachRadar accurately identifies *Points-of-Compromise*, achieving over 90% precision and recall when only 10% of the stolen cards have been used in fraud (see Figure 5.1b).
3. **Distributed POC-Detection algorithm.** We provide a scalable distributed algorithm for POC detection in big datasets.

**Further Applications.** While we focus on the identification of *Points-of-Compromise* in bank transactions, there are other domains where BreachRadar could help identify malicious or abnormal activity. We invite the reader to consider any situation where individual nodes might trigger abnormal behavior in their neighbors. Consider anti-virus applications and machine-file bipartite graphs: given malware symptoms in some of the machines, a small set of files that exist in common could be formulated as *Point-of-Compromise* detection problem. Similarly, quick identification of food-poisoning sources or of faulty parts in the car industry can be formulated under this setting.

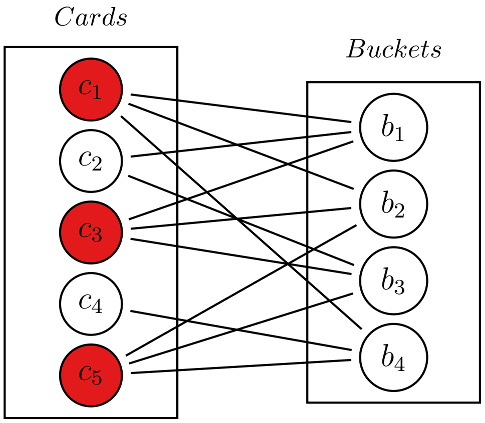


**Figure 5.2: Example of a bipartite network used as input to BreachRadar. Cards in red have been victim of fraud.**

## 5.1 Problem Definition

We assume the point-of-view of a payment network or card issuer who has visibility over the majority of the transactions of a set of cards, some of which have been identified as *fraud-cards* (these are typically canceled and reissued). Loosely speaking, our goal is to automatically identify a small set of locations that many *fraud-cards* have in common.

We represent the set of transactions as a bipartite graph with cards on one side and *possible Points-of-Compromise* on the other.

A *possible Point-of-Compromise* is a feature that a subset of transactions have in common, such as a specific point-of-sale terminal, a store identifier or a merchant name (i.e. all the stores from a corporation). In practice, we would also like to incorporate time as a feature as data breaches and skimming devices temporally correlate transactions: in Section 5.4, we consider *terminal-week* pairs as *possible Points-of-Compromise*, but many other options (or combinations thereof) are admissible. For simplicity, we use the terms *possible Point-of-Compromise* and location interchangeably. Edges connect two nodes if there is a transaction between a given card and a given location.

Let's consider $\mathcal{C}$ to be the set of all cards and $\mathcal{L}$ to be the set of all locations. $\mathcal{G} = (\mathcal{C} \cup \mathcal{L}, E)$ is the bipartite graph and for every edge $(i, j) \in E \Rightarrow i \in \mathcal{C}$ and $j \in \mathcal{L}$.

We will always use index $i$ to represent cards and index $j$ to represent locations. $\mathbf{L}_i$ is the set of neighboring locations of card $i$ and $\mathbf{N}_j$ is the set of neighboring cards of location $j$.

$f : \mathcal{C} \rightarrow \{0, 1\}$, part of our input, is a function indicating whether a given card $c \in \mathcal{C}$ was a victim of fraud or not.

**Table 5.1: Notation, symbols and definitions**

| Symbol | Definition |
|---|---|
| $\mathcal{C}$ | Set of all cards |
| $\mathcal{L}$ | Set of locations (possible POCs) |
| $\mathcal{G}$ | Graph of cards and locations |
| $\mathbf{L}_i$ | Set of locations neighboring card $i$ |
| $\mathbf{N}_j$ | Set of cards neighboring location $j$ |
| $f_i$ | Boolean indicating if $i$ is a fraud-card |
| $\boldsymbol{\theta}$ | Vector of POC probabilities. |
| $\mathbf{B}$ | Blame matrix |
| $\mathbf{b_i}$ | A row of matrix $\mathbf{B}$ |
| $b_{ij}$ | A cell of matrix $\mathbf{B}$ |

$\mathbf{B}$ is a $|\mathcal{C}| \times |\mathcal{L}|$ matrix where $b_{ij}$ is the probability that $j$ is the location responsible for $i$ being a *fraud-card*, or the blame which card $i$ attributes to possible POC $j$.

Using this notation (summarized in Table 5.1), the POC detection problem can be formulated as:

**PROBLEM DEFINITION 3. Spotting Points-of-Compromise**

- **Given:** A graph $\mathcal{G} = (\mathcal{C} \cup \mathcal{L}, E)$ and fraud labels $f : \mathcal{C} \to \{0, 1\}$.
- **Find:**
  - $\boldsymbol{\theta} : \mathcal{L} \to [0, 1]$, where $\theta_j$ is the probability that location $j$ is a *Point-of-Compromise*;
  - $\mathbf{B} : \mathcal{C} \times \mathcal{L} \to [0, 1]$, where $b_{ij}$ is the blame that card $i$ assigns to location $j$.

**that maximize precision** at specific false positive rates.

## 5.2 POC-detection Algorithm

We describe a novel algorithm for the identification of *Points-of-Compromise* following a simple Bayesian inference approach. We adopt the principle that predictions are inherently uncertain and require more evidence in order to increase their confidence, and assume that cards are compromised at a single location and should influence each other towards agreeing on the (locally) most likely mutual *Point(s)-of-Compromise*.

### 5.2.1 A POC Hierarchical Model

We start by assuming that whether a location has been compromised is represented by $p_j$, a Bernoulli random variable whose success probability $\theta_j$ is taken from a Beta prior.

From the card perspective, we assume that each *fraud-card* has an associated variable $r_i$ taken from a categorical distribution of size $|L_i|$ and probability vector $\mathbf{b_i}$, where each element $b_{ij}$ of $\mathbf{b_i}$ is linearly proportional to the respective $\theta_j$ compromise probability. This means we are implicitly assuming that the probability of a card blaming a location is linearly proportional to the probability of it being a POC.

This model can be formally defined as follows:

$$\theta_j \sim Beta(\alpha, \beta) \tag{5.1}$$

$$b_{ij} = \begin{cases} \frac{\theta_j}{\sum_{k \in N_i} \theta_k} & \text{if } f_i = 1 \text{ and } j \in \mathbf{L_i} \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

$$r_i \sim Categorical(|L_i|, \mathbf{b_i}) \tag{5.3}$$

whose corresponding graphical model in plate notation can be seen in Figure 5.3. The only hyper-parameter of this formulation is the Beta distribution which is encoded using $\alpha$ and $\beta$. Intuitively, $\alpha$ and $\beta$ control how much evidence we need to be confident that a location has really been compromised.

The inputs of this model are the sets $\mathbf{N}_i$ (the locations with which each card interacted) and the boolean indicators $f_i$ on whether a card has been a victim. Note the direct relationship between the problem definition and this formulation: the probability that a location $j$ has been compromised can be obtained directly from the expected value of $p_j$ ($E[p_j] = \theta_j$) and the blames attributed by the different cards are encoded in the row-normalized matrix $\mathbf{B}$.

In the following, we will describe an alternating algorithm to simultaneously find $\boldsymbol{\theta}$ and $\mathbf{B}$.

### 5.2.2 From Blames to POC Probabilities

Let's suppose that we knew $\mathbf{B}$, i.e., we knew how much blame each card attributes to each possible POC. Ideally, we would then like to find $\boldsymbol{\theta}$ that maximizes $P[\boldsymbol{\theta}|\mathbf{B}]$, as our model relates the probability of being compromised with the blames attributed.

We defined that $\theta_j$ comes from a $Beta(\alpha, \beta)$ distribution, therefore we know that:

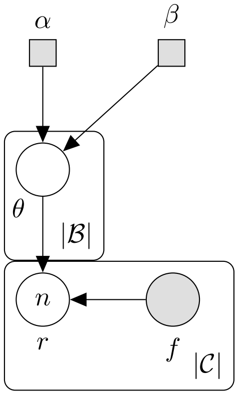


**Figure 5.3: Plate notation of the probabilistic graphical model. Blames $b_{ij}$ are a direct function of $\theta$ and f and are omitted for clarity.**

$$P[\theta_j;\alpha,\beta] = \frac{\theta_j^{\alpha-1}(1-\theta_j)^{\beta-1}}{B(\alpha,\beta)}, \tag{5.4}$$

where the beta function $B(\alpha,\beta)$ is simply the normalization constant that ensures that the total probability integrates to 1.

Let $z_j = \sum_i b_{ij}$ be the sum of all the blames assigned to possible POC $j$. $z_j$ follows a Beta-Binomial distribution and we know that

$$P[z_j|\theta_j;|N_j|] \propto \theta_j^{z_j}(1-\theta_j)^{|N_j|-z_j} \tag{5.5}$$

and, from Bayes' theorem, the posterior distribution equals

$$\begin{gathered} P[\theta_j|z_j;\alpha,\beta,|N_j|] \propto P[z_j|\theta_j;|N_j|]P[\theta_j;\alpha,\beta] \propto \\ \propto \theta_j^{z_j+\alpha-1}(1-\theta_j)^{|N_j|+\beta-z_j-1} \end{gathered} \tag{5.6}$$

This means that the posterior probability distribution of $\theta_j$ is defined as another Beta distribution that can be parametrized as $Beta(z_j+\alpha, |N_j|-z_j+\beta)$ and with expected value

$$E[\theta_j] = \frac{z_j+\alpha}{|N_j|+\alpha+\beta} \tag{5.7}$$

We can think of this expected value as a ratio of the blames ($z_j$) to the total number of cards that used this location ($|N_j|$), with additional terms $\alpha$ and $\beta$ that represent, respectively, "virtual" *fraud-cards* that used this location ($\alpha$) and "virtual" non-*fraud-cards* that used this location ($\beta$). Depending on the prior chosen ($Beta(\alpha,\beta)$), two possible

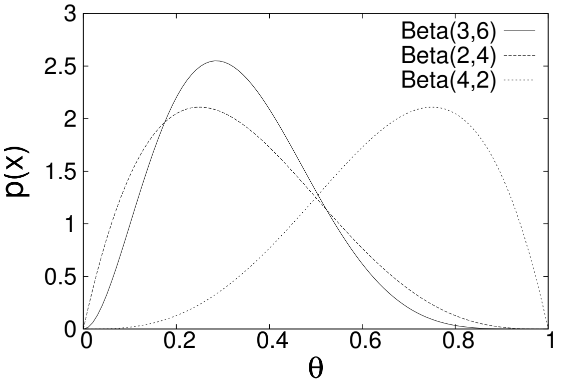


**Figure 5.4: Impact of the prior on the point-of-compromise probability: note how different hyper-parameters place distinct assumptions regarding an unknown location.**

POCs with the same fraud-cards ratio will have their probability adjusted to match our confidence on how far away from the prior distribution they are. The ratio $\frac{\alpha}{\alpha + \beta}$ represents the expected probability that a random location is compromised, while the magnitude of $\alpha$ and $\beta$ encode our confidence on this prior.

### 5.2.3 From POC Probabilities to Blames

Following a similar line of reasoning, let's suppose that we knew $\boldsymbol{\theta}$ and that we would like to find $\mathbf{B}$, i.e. we would like to know the probability that a card will blame each location, given their respective compromise probabilities.

As mentioned in Section 5.2.1, we assume that blaming probabilities are linearly proportional to compromise probabilities. Therefore, the blames matrix $\mathbf{B}$ can be found by directly applying Equation 5.2.

### 5.2.4 An Alternating Algorithm

We previously defined the probability that a location was compromised based on the blames assigned to it, and defined the blame assigned by a card to a possible POC given the compromise probabilities of all the locations in Equation 5.2. This tight coupling suggests an alternating algorithm in which one updates blames and POC probabilities in succession until convergence.

We initialize blames as uniformly distributed across all neighboring possible POCs, and proceed by updating the POC probabilities and blames in sequence. We check for convergence using the $l_1$ norm of the difference of successive POC probability estimations. Algorithm 3 describes this procedure and, as we will see in Section 5.3, one of its advantages is being easily parallelizable.

```
input  : $\boldsymbol{N}$ ($|\mathcal{C}| \times |\mathcal{L}|$) - neighbors of each card
input  : $\boldsymbol{n}$ ($|\mathcal{L}| \times 1$) - number of cards on each possible POC
input  : $\epsilon$ - convergence threshold
input  : $\alpha, \beta$ - prior hyperparameters
output : $\boldsymbol{\theta}$ ($|\mathcal{L}| \times 1$) - POC probabilities
output : $\boldsymbol{B}$ ($|\mathcal{C}| \times |\mathcal{L}|$) - blames matrix
$\boldsymbol{B} \leftarrow$ UniformBlames()
$\boldsymbol{\theta} \leftarrow$ UpdatePOCProbabilities(B, n, $\alpha$, $\beta$)
repeat
    $\boldsymbol{B} \leftarrow$ UpdateBlames($\boldsymbol{\theta}$, $\boldsymbol{N}$)
    $\boldsymbol{\theta_{prev}} \leftarrow \boldsymbol{\theta}$
    $\boldsymbol{\theta} \leftarrow$ UpdatePOCProbabilities(B, n, $\alpha$, $\beta$)
until $||\boldsymbol{\theta} - \boldsymbol{\theta_{prev}}||_1 < \epsilon$
return $\boldsymbol{\theta}$, $\boldsymbol{B}$
Function UpdateBlames($\boldsymbol{\theta}$, $\boldsymbol{N}$)
    foreach card $i$ do
        $sum \leftarrow \sum_{k \in \boldsymbol{N}_i} \boldsymbol{\theta}_k$
        foreach location $j \in \boldsymbol{N}_i$ do
            $\boldsymbol{B}_{ij} \leftarrow \frac{\boldsymbol{\theta}_j}{sum}$
        end
    end
    return $\boldsymbol{B}$
Function UpdatePOCProbabilities($\boldsymbol{B}$, $\boldsymbol{n}$, $\alpha$, $\beta$)
    foreach location $j$ do
        $\boldsymbol{z}_j \leftarrow \sum_i \boldsymbol{B}_{ij}$
        $\boldsymbol{\theta}_j \leftarrow \frac{\boldsymbol{z}_j + \alpha}{\boldsymbol{n}_j + \alpha + \beta}$
    end
    return $\boldsymbol{\theta}$
```

**Algorithm 3:** BreachRadar

### 5.2.5 Convergence

Given its many applications, such as k-means clustering or expectation-maximization methods, the convergence of alternating optimization algorithms is a well studied problem and it is known to work well in a surprising number of cases [BH02a]. In general, one cannot guarantee global convergence but local convergence tends to be very fast. By using the dataset and hyper-parameters described in Section 5.4, we show empirically that our implementation of Algorithm 3 converges exponentially fast in Figure 5.5.

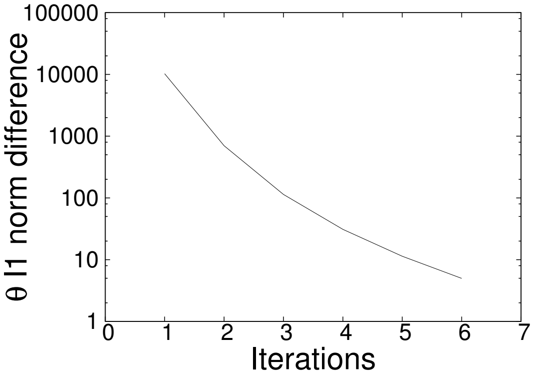


**Figure 5.5: Exponentially fast convergence - notice the log scale in the y-axis.**

## 5.3 Distributed POC-detection

Both stages of the alternating optimization algorithm are parallelizable if we assume a message-passing model of computation, as used in Pregel [MAB$^+$10] and other large-scale graph processing systems. We assume that both nodes and edges can store information which is shared and updated until the whole system converges; in our case, edges contain information relative to blames ($\mathbb{B}$) while location nodes contain their respective *Point-of-Compromise* probability.

Under this new model of computation, the differences to Algorithm 3 can be summarized in Algorithm 4. Succinctly, blames can be updated using a round of message passing from each *potential Point-of-Compromise* to each neighboring card, followed by a mapping of the edges that only requires the data stored in their adjacent nodes. Point-of-Compromise probabilities can be updated in a similar fashion: blames can be propagated to the adjacent *potential POC* which update their internal variables.

In order to obtain an efficient parallel solution, this procedure was implemented using Apache Spark [ZCF$^+$10], a MapReduce engine that enables in-memory computation.

```
input : $\boldsymbol{N}$ ($|\mathcal{C}| \times |\mathcal{L}|$) - neighbors of each card
input : $\boldsymbol{n}$ ($|\mathcal{L}| \times 1$) - number of cards on each possible POC
input : $\epsilon$ - convergence threshold
input : $\alpha, \beta$ - prior hyperparameters
output: $\boldsymbol{\theta}$ ($|\mathcal{L}| \times 1$) - POC probabilities
output: $\boldsymbol{B}$ ($|\mathcal{C}| \times |\mathcal{L}|$) - blames matrix

On G.POCs.NewMessage($blame$)
    $z = z + blame$
    $\theta = \frac{z+\alpha}{n_j+\alpha+\beta}$
On G.Cards.NewMessage($\theta$)
    $sum = sum + \theta$
Function UpdateBlames($\boldsymbol{\theta}$, $\boldsymbol{N}$)
    foreach POC $j$ in G in parallel do
        foreach Card $i$ in $N_j$ in parallel do
            $j$.sendMessage($i$, $\theta$)
        end
    end
    /* After all messages are aggregated.                    */
    foreach Edge $e$ in G in parallel do
        $e.blame = \frac{e.POC.\theta}{e.card.sum}$
    end
Function UpdatePOCProbabilities($\boldsymbol{B}$, $\boldsymbol{n}$, $\alpha$, $\beta$)
    foreach Edge $e$ in G in parallel do
        $e$.sendMessage($e.POC$, $e.blame$)
    end
```

**Algorithm 4:** Distributed BREACHRADAR

**Table 5.2: Several Points-of-Compromise identified in one of the datasets have also been mentioned in news reports.**

| Merchant | Source | Summary |
|---|---|---|
| Schnucks | *ComputerWorld* [Vij15] | A supermarket chain where a breach exposed 2.4M cards. |
| NoMoreRack.com | *Reuters* [Roy13] | An online retailer with over $340 million in sales annually, probes likely card breach. |
| Jetro Cash & Carry | *DataBreaches* [jet11] | A data breach allowed attackers to access private data in cards used over a one month period in this chain. |
| Bashah's Family of Stores | *BankInfoSecurity* [Kit13] | A supermarket chain tied to the compromise of hundreds of cards. |
| Buy.com | *Yahoo Finance - Money Talks* [Bal13] | Hundreds of online shoppers reported fraudulent charges on their credit cards after making a purchase at this online marketplace. |

Spark is well suited for machine learning algorithms as its in-memory model doesn't force sequential stages to synchronize data to disk. In particular, we rely on Spark's GraphX [XGFS13] module which overlays an abstraction for graph-parallel computation that allows message passing and aggregation.

## 5.4 Results

We answer the following questions:

**Q1. Effectiveness -** How accurately and how early can we detect *Points-of-Compromise* in reality? What are the trade-offs between number of *fraud-cards* and accuracy?

**Q2. Scalability -** How does our method scale in terms of the size of the network and in terms of the number of machines available?

**Q3. Fraud-cards precision and recall -** How much of the fraud that is reported can be explained through the identification of *Points-of-Compromise*?

**Q4. Discoveries -** Can we identify real and validated *Points-of-Compromise* in real data?

### 5.4.1 Experimental Setup

BreachRadar was applied to two datasets provided by different sources, each with over one billion transactions, 0.4 million cards and 2 million fraudulent transactions that cover more than one year. The data is very unbalanced with the percentage of transaction fraud in accordance with industry averages, between 0.01% and 0.1% [Eur15]. Due to privacy concerns, results in this section do not indicate the corresponding dataset.

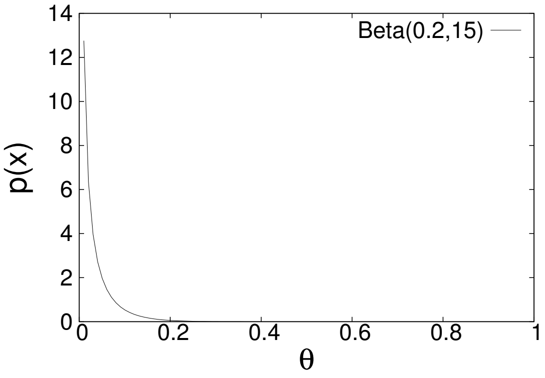


**Figure 5.6: The prior used is biased against considering a location compromised.**

We created possible POCs corresponding to *terminal-week* pairs and removed multi-edges and all possible POCs that interacted with less than 5 *fraud-cards*, as they could not be confidently labeled *Points-of-Compromise* under any circumstance. After this pre-processing stage, there were at least 1.5 million terminal-week pairs that had to be considered in each dataset. Results here reported correspond to a $\alpha = 0.2$ and $\beta = 15$ prior (see Figure 5.6) which provides a significant assumption that a random *terminal-week* is not compromised. Results did not differ significantly with other values of $\alpha$ and $\beta$ we tested.

**Table 5.3: Overview of the two datasets used. Specific values masked for privacy.**

| | **#cards** | **#transactions** | **#locations** | **#fraud transactions** |
|---|---|---|---|---|
| **Dataset 1** | $> 10^5$ | $> 10^9$ | $> 10^6$ | $> 10^6$ |
| **Dataset 2** | $> 10^5$ | $> 10^9$ | $> 10^6$ | $> 10^6$ |

**Reproducibility.** The dataset used in the comparison experiments described in Section 5.4.3 is available upon request.

### 5.4.2 Empirical Evidence and Fraud Prevented

We collected significant empirical evidence demonstrating our ability to find real POCs, both data breaches and terminals that we suspect were victims of skimming, through manual analysis of the POCs reported. The list includes tobacco machines equipped with credit card readers and other general vending machines where the percentage of *fraud-cards* is as high as 80%.

Data breaches are easier to validate as they are often reported in the news; a non-exhaustive list of POCs we automatically detected along with a sample news report can be found in Table 5.2. We were also able to identify POCs whose first report occurred more than 6 months after the last transaction have available in the dataset.

We evaluated the amount of fraud that could be prevented if cards of likely-compromised POCs were automatically reissued. Figure 5.1a shows the gains obtained when BreachRadar is evaluated in a 6 weeks period. Over $2 million USD in savings would be possible when reissuing all cards that interacted with POCs with an expected compromise probability above 10%, even if we assume a $10 reissue cost per card. 17% of the cards reissued would have been victims of fraud and 95% of these would be first-time victims.

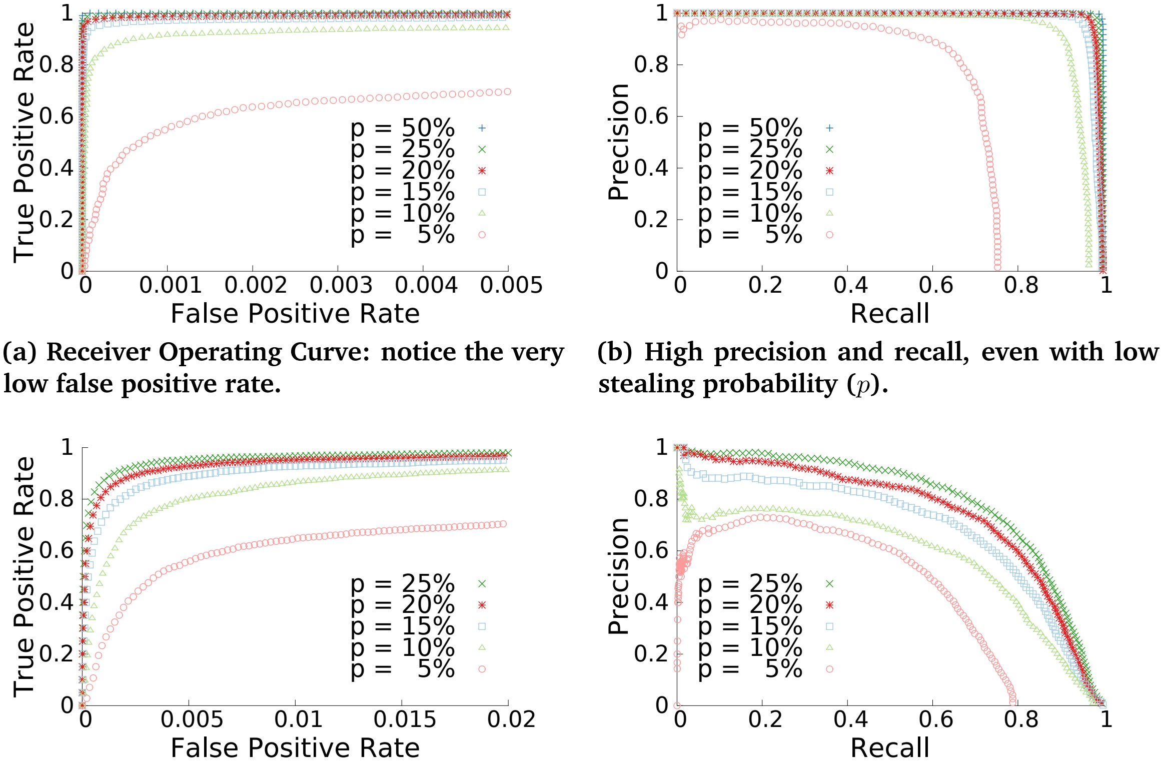
### 5.4.3 Accuracy and Early Detection

Due to the lack of ground-truth regarding which locations are effectively *Points-of-Compromise*, we evaluate the precision and recall of BreachRadar through the injection of synthetic POCs in real data. We evaluate BreachRadar along two dimensions:

1. Probability that a card is a victim of fraud after using a compromised location ($p$). This can be viewed as a proxy for how early our method is able to detect compromised cards, as detection gets naturally easier as the number of victims increases.
2. Presence of noise in the set of *fraud-cards*, i.e., we randomly mark additional cards as victims, although fraud cannot be attributed to any of the POCs.

In each experiment, we define a set of POCs and vary a probability (p) that their transactions will steal the corresponding card. Based on this new *fraud-cards* list, we then obtain a new list of possible *Points-of-Compromise*. Table 5.4 shows the number of *fraud-cards* and *possible Points-of-Compromise* as the probability of the card being stolen increases, when no noise is included.

An increase in the probability of a card being compromised corresponds to an obvious increase in the total number *fraud-cards* and, as a consequence, of the total number of *possible Points-of-Compromise* that must be considered. However, we will see that the increased probability more than offsets the larger search space and higher accuracy can be achieved. Figure 5.7a and Figure 5.7b show the receiver operating characteristic (ROC) and "precision vs recall" curves for the different compromise probabilities. Note that we are able to simultaneously achieve high precision and recall for relatively small



**(a) Receiver Operating Curve: notice the very low false positive rate.**

**(b) High precision and recall, even with low stealing probability ($p$).**

**(c) Receiver Operating Curve: low false positive rate, with 100% noise.**

**(d) Even with 100% added noise, high precision and recall.**

**Figure 5.7: Accuracy with varying probability of a card being a fraud victim. (Top row: without noise. Bottom row: 100% additional fraud-cards as noise.)**

compromise probabilities; we achieve over 90% precision and recall even when only 10% of the cards have been victims of fraud.

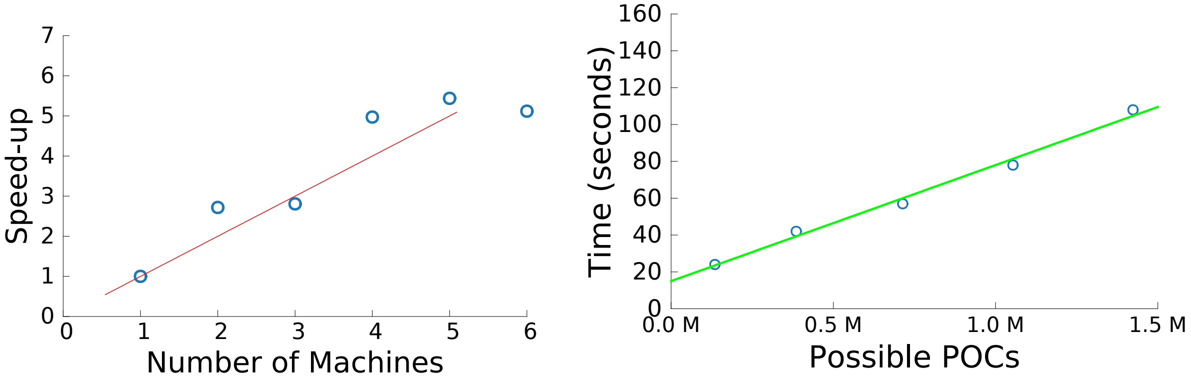
We also analyze the impact of noise in the effectiveness of our method: we double the number of *fraud-cards* by randomly selecting additional cards. These are cards that do not have a corresponding POC in the data, even though they were marked as victims. As before, Figure 5.7c and Figure 5.7d show the ROC and “precision vs recall” curves for the different compromise probabilities. Using the same scenario of a 10% probability of cards being compromised as example, note that we are still able to achieve about 75% recall maintaining 50% precision, even though we are simultaneously considering very aggressive levels of cards mislabeled as *fraud-cards* and early detection.

**Table 5.4: Impact of the infection probability on the number of *fraud-cards* and possible Points-of-Compromise.**

| Probability ($p$) | #fraud-cards | #possible POCs |
|---|---|---|
| 5% | 29 326 | 104 185 |
| 10% | 57 593 | 263 115 |
| 15% | 84 833 | 450 609 |
| 20% | 110 955 | 662 967 |
| 25% | 136 896 | 892 119 |
| 50% | 257 609 | 2 168 733 |

### 5.4.4 Scalability

Scalability experiments are performed using the data described in Table 5.4. We show BREACHRADAR linear scalability on the number of possible POCs (Figure 5.8b) and analyze the runtime of its Spark implementation when changing the number of machines available in a small cluster of 6 quad-core machines (Figure 5.8a). The number of cores in each machine does not provide any advantage, as disk input-output is the bottleneck of our system, not processing power.



**(a) Good distributed speed-up.**

**(b) Linear speed-up with number of possible Points-of-Compromise.**

**Figure 5.8: BREACHRADAR's speed-up and scalability.**

### 5.4.5 Comparison

We compare the precision and recall of our method to (1) FABP - Fast Belief Propagation [KKK+11]; (2) a greedy approximation of Vertex Cover described in Section 5.5; (3)

the ratio as proxy for POC probability, as commonly used by previous methods and (4) the ratio metric combined with the best prior found for BreachRadar, in order to reduce its false positive rate.

FaBP [KKK+11] is a fast approximation to Belief Propagation with low sensitivity to input parameters. We assigned a high prior belief (0.5) to *fraud-card* nodes, a low prior belief (-0.1) to non-*fraud-card* nodes and a neutral belief (0.0) to possible POC nodes. We then decreasingly sort the possible POCs by their final belief, creating the corresponding precision vs recall curve. The curve for the other methods was created similarly, based on the ordering of the possible POCs that they explicitly define.

We compare all methods on the non-noise dataset when considering a stealing probability of 10%.

As can be seen in Figure 5.1b, our method significantly improves over all alternatives. Reasons have been detailed in previous sections, but can be summarized as a combination of appropriate priors and the focused blame of the *fraud-cards*. Vertex Cover's result shows that focused blames are not sufficient, while the ratio with prior's result shows that removing false positives with a small amount of *fraud-cards* is not enough either.

## 5.5 Related Work

While apparently a simple problem, several reasons compound to make the automatic detection of POCs a challenge to naive approaches: a) the variety of *Points-of-Compromise*, e.g., database breaches, card skimming devices, etc; b) the variety of time granularities, e.g. database breaches compromise months of transactions, while an employee skimming cards might do it for a single day; c) the lack of ground-truth labels, as fraudulent charge reports do not identify the origin of the compromise and d) the scale of the problem, as datasets with billions of transactions with millions of possible *Points-of-Compromise* are common.

### 5.5.1 Summary

Table 5.5 characterizes the most relevant methods described in this section. We analyze a method's ability to *find Points-of-Compromise* and to *scale* at least quasilinearly with the number of transactions that need to be processed. We consider that a method has proper *risk assessment* if it doesn't believe that more transactions to safe merchants reduce the probability that the card might have been stolen at a single compromised location. We consider methods to be *Blame-aware* if they acknowledge that cards are likely stolen only once, so they should not significantly contribute to an increased POC likelihood

**Table 5.5: Comparison of BreachRadar with other methods. Properties are described in Section 5.5.**

| | BreachRadar | Ratio | Ratio + Prior | Belief Propagation | Vertex Cover | Real-time Detection |
|---|---|---|---|---|---|---|
| **Finds POCs** | ✔ | ✓ | ✓ | ✓ | ✓ | |
| **Scalability** | ✔ | ✓ | ✓ | ✓ | | ✓ |
| **Risk assessment** | ✔ | ✓ | ✓ | | ✓ | |
| **Blame-aware** | ✔ | | | | ✓ | |
| **Confidence-aware** | ✔ | | ✓ | | | |
| **Early Detection** | ✔ | | | | | |

of multiple locations. We consider a method to be *Confidence-aware* if it incorporates the idea that more evidence improves the confidence of a POC label[1]. Finally, a method is capable of *Early Detection* if it shows high recall even when only a small percentage (e.g., 10%) of the cards at a location are *fraud-cards* (a card with at least one fraudulent transaction).

### 5.5.2 Real-time Fraud Detection

While not able to identify *Points-of-Compromise*, state-of-the-art fraud detection solutions merge statistical, machine learning and data mining tools in order to create models that estimate the fraud probability of individual transactions in real-time. For further information on this orthogonal problem, we refer the reader to specific literature [BH02b, DPCLB+14].

### 5.5.3 Points-of-Compromise

Simple metrics are unable to provide an appropriate measure for the likelihood of a point being the origin of a compromise. Ranking locations by the number of *fraud-cards* with which they interacted does not work, as many merchants process many transactions and thus interact with a high number of *fraud-cards*. The ratio of *fraud-cards* shouldn't be used

[1] As an example, we are more confident that a location has been compromised if 200 out of 600 cards who transacted there were victims, than if 3 out of 6 were.

either, as the majority of the locations have small numbers of transactions and high ratios (by chance) do not imply a compromised location.

Current systems for *Point-of-Compromise* detection are typically hindered by these issues. Absolute number of *fraud-cards* and *fraud-card* ratios are commonly used [Kle09, SPG$^+$ne], perhaps coupled with time-windows [Yan13] to restrict the set of transactions considered. Arbitrary thresholds that indicate whether a merchant was compromised need to be defined, but suggestions have been made that supervised classification algorithms could also be trained, after information about which merchants were in fact compromised is obtained [For05]. A different approach suggests comparing recent fraud-rates at each merchant with their historical fraud-rate and flagging outlier deviations [ZWSW10].

### 5.5.4 Guilt-by-association

The aforementioned problem definition can be framed as a Bayesian Network [Pea85] - a graphical model that represents random variables and their conditional dependencies through directed acyclic graphs. Belief Propagation [Pea82] is one of the most common approaches for performing inference on graphical models. By passing local messages between the connected nodes in the Bayesian Network, it manages to approximate the marginal distribution for each unobserved node, conditional on any observed nodes.

Semi-supervised learning techniques, in particular graph-based methods such as Label Propagation, could be used to label nodes as *compromised* or not-*compromised* in the network; labeled nodes *influence* neighbors according to an Homophily Matrix which establishes whether nearby nodes tend to have similar or opposite labels. Some variations extend Label Propagation to incorporate label confidence and prior information, which could be used when only positive labels (fraud) are observed [YFK15]. Koutra et al. [KKK$^+$11] showed that Belief Propagation and Semi-Supervised Learning are very closely related (but not identical) and could be understood under the same framework when viewed as linear systems.

However, both methods have the underlying assumption that more connections to innocent nodes imply a smaller likelihood that the node has been compromised; this idea is at the core of guilt-by-association algorithms: similarly to how connections to fraudulent nodes increase the probability of a node being fraudulent, then connections to safe locations decrease this probability. For this problem, we know that the opposite is true: connections to innocent merchants do not compensate for the fact that a connection to a compromised location exists. Our results showing Belief Propagation's low performance corroborate this intuition.

### 5.5.5 Vertex Cover

This problem can also be formulated as a vertex cover problem in bipartite graphs: given a set of cards who were victim of fraud, we would like to identify the smallest subset $\mathcal{S}$ of adjacent nodes (i.e., merchants) so that every card who has been a victim has at least one adjacent location in $\mathcal{S}$. Unfortunately, this formulation is NP-hard[2]. Nevertheless, in Section 5.4.5 we evaluate a greedy approximation: on each iteration, we consider as compromised the location with the highest number of adjacent *fraud-cards*, remove them from the bipartite graph and repeat.

## 5.6 Summary

We present, as far as the authors know, the first distributed procedure for the automatic detection of *Points-of-Compromise* in transaction networks. We achieve highly accurate results through the implementation of an in-memory algorithm that updates POC probabilities and blame scores alternatingly, and we have demonstrated surprising empirical evidence in a real dataset. Our main contributions are the following:

1. **Point-of-Compromise Problem.** We formulate a novel and important POC detection problem.
2. **Effectiveness.** BreachRadar accurately identifies *Points-of-Compromise*, achieving over 90% precision and recall when only 10% of the stolen cards have been used in fraud (Figure 5.1b).
3. **Distributed POC-Detection algorithm.** We provide a scalable distributed algorithm for POC detection in big datasets (Figure 5.8b).

[2] This can be easily shown through reduction to Minimum Set Cover, one of Karp's 21 NP-complete problems [Kar72].

# Part II

# Labeled Networks and Tensors

# Overview and Related Work

Part I explored communities and anomalies in unlabeled networks. We analyzed the structure of real communities in multiple settings, devised algorithms in which this structure emerged, and studied the particular scenario of anomalous *Point-of-Compromise* nodes.

However, in reality, interactions are not all the same and we are able to characterize them along different vectors: when they took place, the medium of communication used, the duration of the conversation or the content of the interaction are just a few examples. Incorporating this additional information in community and anomaly detection methods would improve both our understanding of the existing patterns and community quality. In Part II, we start by finding communities in labeled networks, with a particular focus on time-evolving graphs. We present novel algorithms for this problem, one of which is easily distributed. Then, we dig deeper on the time-evolving graphs theme and present new methods for the forecasting of novel relations in datasets where context is available.

## I Related Work: Communities in Edge-labeled Networks

### I.1 Categorical Edge-labels

The detection of communities using categorical edge-labels has not been studied extensively in the literature, but the general consensus is that methods should try to simultaneously co-cluster nodes and labels. Most solutions are motivated by heterogeneous or multi-layered data, where nodes are connected through different kinds of relations.

MutuRank and GMM-NK [WYC+13] start by determining weights of various relation types and objects that are then use to create a single-level network by combining the different probability distributions. They evaluate their approach on a DBLP dataset where each layer corresponds to a research field.

PMM [TWL09] is a spectral method that starts by calculating the eigen-decomposition of the individual adjacency matrices (i.e. considering labels independently), and then clusters the feature vectors of the different nodes together using *k-means*. This way, they find nodes that have a similar "profile" along different edge-labels, but the method is severely penalized as the number of labels increases. Boden [BGHS12, BGHS13] extends the quasi-clique definitions that have been introduced to detect communities where nodes show similarity in subsets of the edge labels.

Other approaches rely on sparsifying the dense and real-valued PARAFAC decomposition in order to identify communities. Possibilities include thresholding the values in the component vectors after the initial decomposition, or modifying the decomposition itself by imposing sparsity using $\ell_1$ penalty terms. Details have been provided in Chapter 2. As an example, GRAPHFUSE [PAI13] starts by calculating a sparse PARAFAC decomposition of the tensor and then assigning each node to the cluster in which it has the highest weight in the decomposition, effectively partitioning the nodes. Because it creates a hard clustering (with no overlapping), GRAPHFUSE is more closely related to graph partitioning than to community detection.

## I.2 Time-evolving Networks

Graph evolution has been a topic of interest for some time, particularly in the context of web data [KNRT03, LKF07]. [AS14] published a very recent survey on evolutionary network analysis, in which they classify evolutionary clustering methods in eight categories (spectral, probabilistic, density-based, matrix factorization, modularity, information theoretic, pattern mining and others). We refer the reader to this survey for a more detailed analysis.

[LYK+08] studies the problem of detecting changing communities, but require selection of a small number of parameters. Furthermore, broadly related work uses tensor-based methods for analysis and prediction of time-evolving "multi-aspect" structures, e.g. [STF06, DKA11].

More related to the work discussed in this dissertation, MDL-based approaches for detecting non-overlapping communities in time-evolving graphs have been previously proposed, such as GRAPHSCOPE [SFPY07]. However, this work focuses on incremental, streaming community discovery, imposing segmentation constraints over time, rather than on discovering communities that might appear intermittently. Another alternative is TimeCrunch [SKZ+15], a community detection algorithms for edge-labeled graphs. They find patterns that minimize the description length (MDL) of the tensor, i.e. a cost for encoding both patterns and errors, with a focus on explainable time-evolving patterns. It first slices the time-evolving graph into static graphs with different time stamps and then uses VoG [KKVF14] 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 Approaches.** Many models [TY11, YCZ$^+$11, XH13, GZC$^+$16, Pei15] detect hidden structures (e.g. community assignments [TY11], group evolving paths [YCZ$^+$11], or hidden class interactions in different time periods [XH13]) 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.

Chapter based on work that appeared at PAKDD 2014 [APG+14] and KAIS 2015 [AGP+15].

# Chapter 6

# Communities in Labeled Networks

In Part I, we saw how communities are part of the natural structure of many different networks and analyzed their shape and properties. For instance, we saw that communities appear as the interaction between users and products and between airports. However, in reality, interactions are not all the same and we are able to characterize them along different vectors: when they take place, the means of communication used, the duration of the communication or the content of the interaction are just a few examples. Incorporating this additional information in community detection methods would allow us to characterize communities on these dimensions and we would be able to detect communities whose nodes have similar interactions, improving community quality. In a social network, for example, we observe nodes corresponding to users and edges correspond to phone calls, emails or text messages. By classifying these interactions in simple categories such as “work”, “school”, “leisure” and “family”, and then grouping people with similar interactions, we are able to significantly increase community quality. Standard techniques would ignore this extra information and mix, e.g., work and family relations.

Here we focus on exactly this problem: how to find communities in an edge-labeled network, in a scalable way without user-defined parameters. We analyze a large, million-node graph, from an anonymous (and anonymized) dataset of mobile customers of a large population and a bipartite computer network with hundreds of thousands of connections, available to the public, to detect time-varying communities. We also analyze flights data (where nodes correspond to airports and edges are labeled with the company operating the flight) to find which companies are the biggest competitors in different regions. Figure 6.1 illustrates a sample community, in which 3 big airlines (Lufthansa, Delta and United Airlines) heavily compete in 16 worldwide airports; we illustrate other specific regional competitors in Section 6.3. We shall refer to the communities we discover as *comet communities*, because they are only active over some labels; when labels represent time, then they (may) come and go, like comets.

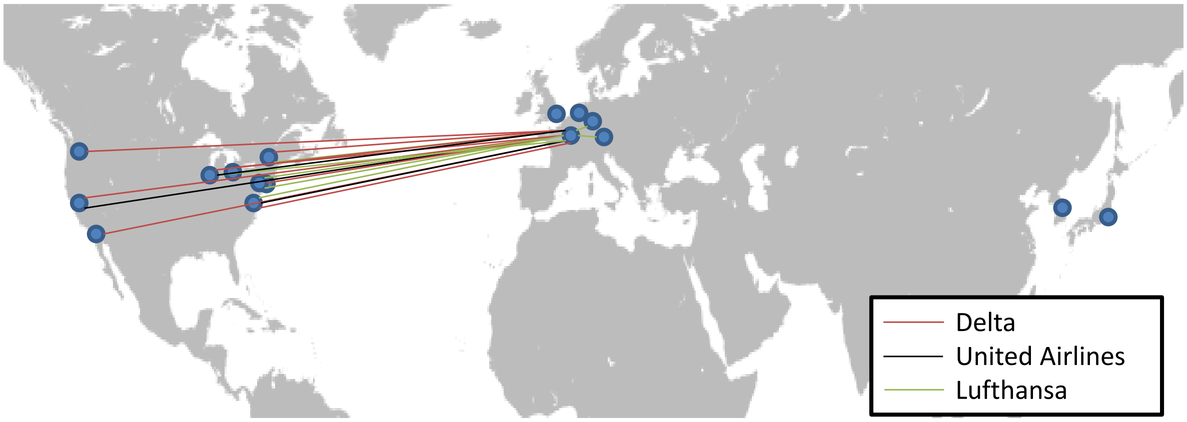


**Figure 6.1: Worldwide flights community: COM$^2$ is able to detect situations of competition in flight records without user-defined parameters. Lufthansa, Delta and United Airlines compete in the 16 biggest world airports flying $37\%$ of the valid routes with significant overlap. In this figure, we only show the flights in this community connected to Charles de Gaule airport, France.**

We seek to answer the following informal problem:

PROBLEM DEFINITION 4. **Finding Labeled Communities**

- **Given** a labeled graph represented as a tensor $\boldsymbol{\mathcal{X}}$.
- **Find** a set of possibly overlapping communities that best represent the labeled network.

We propose COM$^2$, an effective approach to summarize large labeled networks. Similarly to the work described in Chapter 3, we formalize this problem as the minimization of the Minimum Description Length (MDL) required to describe the graph. Our main contributions are the following:

- **Scalability**: COM$^2$ is linear on the input size, thanks to a careful, incremental tensor-analysis method, based on fast, iterated rank-1 decompositions.
- **No user-defined parameters**: COM$^2$ uses a novel Minimum Description Length (MDL) based formulation of the problem, to automatically guide the community discovery process.
- **Effectiveness**: We applied COM$^2$ on real and synthetic data, discovering edge-labeled communities that agree with intuition.
- **Generality**: COM$^2$ can be easily extended to handle higher-mode tensors.

In the rest of this chapter, we first formalize the objective our method and then present the algorithmic solution in Section 6.2. We evaluate our solution in Section 6.3.

## 6.1 Problem Definition

In this section, we formalize our problem through the description of a MDL-based optimization problem which will guide the community discovery process. While our method generalizes to tensors with an arbitrary number of modes, we illustrate our method using 3-mode tensors to simplify its understanding.

We are given a (possibly directed) network consisting of sources $\mathcal{S}$, destinations $\mathcal{D}$ and edge-labels $\mathcal{L}$. We represent this network via a 3-mode tensor $\boldsymbol{\mathcal{X}} \in \{0,1\}^{|\mathcal{S}|\times|\mathcal{D}|\times|\mathcal{L}|}$ where $\mathcal{X}_{ijl} = 1$ if source $i$ is connected to destination $j$ via an edge with label $l$. As abbreviations we use $N = |\mathcal{S}|$, $M = |\mathcal{D}|$, and $K = |\mathcal{L}|$. In many practical scenarios, the set of sources $\mathcal{S}$ equals the set of destinations $\mathcal{D}$.

The goal is to automatically detect the set of communities $\mathcal{C} = \{C_1, ..., C_k\}$ that best describes tensor $\boldsymbol{\mathcal{X}}$, where $k$ is part of the optimization and not known a priori.

**Definition 4.** *Community*
A community is a triplet $C = (S, D, L)$ with $S \subseteq \mathcal{S}$, $D \subseteq \mathcal{D}$, and $L \subseteq \mathcal{L}$ such that elements in $S$ are well connected to elements in $D$ using edges with labels in $L$. An edge is part of the community if both nodes and the corresponding label are part of the community, i.e. $E(C)_{(i,j,l)} = 1 \Leftrightarrow i \in S, j \in D, l \in L$.

We propose to measure the 'importance' of a community via the principle of compression, i.e. by the community's ability to help us compress the 3-mode tensor: if most of the sources are connected to most of the destinations using most of the indicated labels, then we can compress this 'comet-community' easily. By finding the set of communities leading to the best compression of the tensor, we get the overall most important communities.

More specifically, we use the MDL (Minimum Description Length) principle [Grü07]. That is, we aim to minimize the number of bits required to encode the detected patterns (i.e. the model) and to describe the data given these patterns (corresponding to the effects of the data which are not captured by the model). Thus, the overall description cost automatically trades off the model's complexity and its goodness of fit. In the following, we provide more details about the description cost:

*Description cost.* The first part of the description cost accounts for encoding the detected patterns $\mathcal{C} = \{C_1, \ldots, C_k\}$. Each pattern $C_i = (S_i, D_i, L_i)$ can be completely described by the cardinalities of the three included sets and by the information of which nodes and labels belong to these sets. Thus, the coding cost for a pattern $C_i$ is

$$L_1(C_i) = L_{\mathbb{N}}(|S_i|) + L_{\mathbb{N}}(|D_i|) + L_{\mathbb{N}}(|L_i|) + |S_i| \cdot \log N + |D_i| \cdot \log M + |L_i| \cdot \log K \tag{6.1}$$

The first three terms encode the cardinalities of the sets using the MDL optimal universal codelength $L_{\mathbb{N}}$ for integers [Ris83]. The last three terms encode the actual membership information of the sets using block-encoding: e.g., since the original graph contains $N$ sources, each source included in the pattern can be encoded by $\log N$ bits, which overall leads to $|S_i| \cdot \log N$ bits to encode all sources included in the pattern.

Correspondingly, a set of patterns $\mathcal{C} = \{C_1, \ldots, C_k\}$ can be encoded by the following number of bits:

$$L_2(\mathcal{C}) = L_{\mathbb{N}}(|\mathcal{C}|) + \sum_{C \in \mathcal{C}} L_1(C) \tag{6.2}$$

That is, we encode the number of patterns and sum up the bits required to encode each individual pattern.

Since in real world data we expect to find overlapping communities, our model should not be restricted to disjoint patterns. But how to reconstruct the data based on overlapping patterns? As an approach, we refer to the principle of Boolean algebra: multiple patterns are combined by a logical disjunction. That is, if an edge occurs in at least one of the patterns, it is also present in the reconstructed data. This idea is related to the paradigm of Boolean Tensor Factorization [Mie11]. More formally, the reconstructed tensor is given by:

**Definition 5.** *Tensor reconstruction*
Given a community $C$, we define the indicator tensor $\boldsymbol{\mathcal{I}}^C \in \{0,1\}^{N \times M \times K}$ to be the 3-mode tensor with $\mathcal{I}^C_{ijl} = 1 \Leftrightarrow (i,j,l) \in E(C)$.
Given a set of patterns $\mathcal{C}$, the reconstructed tensor $\boldsymbol{\mathcal{X}}^{\mathcal{C}}$ is defined as $\boldsymbol{\mathcal{X}}^{\mathcal{C}} = \bigvee_{C \in \mathcal{C}} \boldsymbol{\mathcal{I}}^C$ where $\vee$ denotes element-wise disjunction.

The second part of the description cost encodes the data given the model. Given that the MDL principle requires a lossless reconstruction of the data and since the reconstructed tensor, $\boldsymbol{\mathcal{X}}^{\mathcal{C}}$, unlikely reconstructs the data perfectly, we also have to encode the 'errors' made by the model. Here, an error might either be an edge appearing in $\boldsymbol{\mathcal{X}}$ but not in $\boldsymbol{\mathcal{X}}^{\mathcal{C}}$, or vice versa. Since we consider a binary tensor, the number of errors can be computed based on the squared Frobenius norm of the residual tensor, i.e. $\left\| \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}^{\mathcal{C}} \right\|_F^2$.

Finally, as 'errors' correspond to edges in the graph, the description cost of the data can now be computed as

$$L_3(\boldsymbol{\mathcal{X}}|\mathcal{C}) = L_{\mathbb{N}}\left(\left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}^{\mathcal{C}}\right\|_F^2\right) + \left\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}^{\mathcal{C}}\right\|_F^2 \cdot (\log N + \log M + \log K) \tag{6.3}$$

Technically, we also have to encode the cardinalities of the set $\mathcal{S}$, $\mathcal{D}$, and $\mathcal{L}$ (i.e. the size of the original tensor). Given a specific dataset, however, these values are constant and thus do not influence the detection of the optimal solution.

### Overall model

Given the functions $L_2$ and $L_3$, we are now able to define the communities that minimize the overall number of bits required to describe the model and the data:

**Definition 6.** *Community model*
Given a tensor $\boldsymbol{\mathcal{X}} \in \{0,1\}^{|\mathcal{S}|\times|\mathcal{D}|\times|\mathcal{L}|}$, the set of communities is defined as the set of patterns $\mathcal{C}^* \subseteq (\mathcal{P}(\mathcal{S}) \times \mathcal{P}(\mathcal{D}) \times \mathcal{P}(\mathcal{L}))$ fulfilling

$$\mathcal{C}^* = \arg\min_{\mathcal{C}}[L_2(\mathcal{C}) + L_3(\boldsymbol{\mathcal{X}}|\mathcal{C})] \tag{6.4}$$

Again, it is worth mentioning that the patterns detected based on this definition are not necessarily disjoint, thus better representing the properties of real data.

## 6.2 Algorithmic solution

We now describe a novel, fast, and efficient search strategy, based on iterated rank-1 tensor decompositions which can discover communities in edge-labeled networks in a fast and effective manner.

Computing the optimal solution of Equation 6.4 is infeasible as it is NP-hard [1]. Therefore, in the following, we introduce a scalable and efficient algorithm that approximates the optimal solution via an iterative method of sequentially detecting important communities. The general idea is to find in each step a single community $C_i$ that contributes the most to the MDL-compression based on local evaluation. That is, given the already detected communities $\mathcal{C}_{i-1} = \{C_1, \ldots, C_{i-1}\}$, we are interested in finding a novel community $C_i$ which minimizes $L_2(\{C_i\} \cup \mathcal{C}_{i-1}) + L_3(\boldsymbol{\mathcal{X}}|\{C_i\} \cup \mathcal{C}_{i-1})$. Since $\mathcal{C}_{i-1}$ is given, this is

[1] It is known that the column reordering problem in two dimensions is NP-hard as well [JKC+04]

equivalent to minimizing

$$L_1(C_i) + L_3(\boldsymbol{\mathcal{X}}|\{C_i\} \cup \mathcal{C}_{i-1}). \tag{6.5}$$

Obviously, enumerating all possible communities is infeasible. Therefore, to detect a single community $C_i$, the following steps are performed:

- **Step 1: Community candidates**: We spot candidate nodes and labels by performing a rank-1 approximation of the tensor $\boldsymbol{\mathcal{X}}$. This step provides a normalized vector for each dimension with the score of each element.
- **Step 2: Community construction**: The scores from the previous step are used in a hill climbing search as a bias for connectivity, while minimizing the MDL costs is used as the objective function for determining the correct community size.
- **Step 3: Tensor deflation**: Based on the current community detected, we deflate the tensor so that the rank-1 approximation is steered to find novel communities in later iterations.

In the following, we discuss each step of the method. Note the differences to the work described in Section 3.3: most operations are now higher-dimensional and cost functions were modified according to our new setting.

### 6.2.1 Community candidates

As mentioned, exhaustively enumerating all possible communities is infeasible. Therefore we propose to iteratively let the communities grow. The challenge, however, is how to spot nodes and/or labels that should be added to a community. For this purpose, we refer to the idea of tensor decomposition. Given the tensor $\boldsymbol{\mathcal{X}}$ (or as we will explain in step 3, the deflated tensor $\boldsymbol{\mathcal{X}}^{(i)}$), we compute vectors $\mathbf{a} \in \mathbb{R}^N$, $\mathbf{b} \in \mathbb{R}^M$, and $\mathbf{c} \in \mathbb{R}^K$ providing a low rank approximation of the community. Intuitively, sources connected to highly-connected destinations at highly active labels get a higher score in vector $\mathbf{a}$ and similarly for the other two vectors.

Specifically, to find these vectors, a scalable extension of the matrix-power-method only needs to iterate over these equations, which are defined in Chapter 2 and repeated here for convenience:

$$\begin{aligned} a_i &\leftarrow \sum_{j=1,k=1}^{M,K} \mathcal{X}_{ijk} b_j c_k \\ b_j &\leftarrow \sum_{i=1,k=1}^{N,K} \mathcal{X}_{ijk} a_i c_k \\ c_k &\leftarrow \sum_{i=1,j=1}^{N,M} \mathcal{X}_{ijk} a_i b_j \end{aligned} \tag{6.6}$$

where $a_i$, $b_j$ and $c_k$ are the scores of source $i$, destination $j$ and label $k$. These vectors are then normalized and the process is repeated until convergence. Initial values are assigned randomly from the range 0 to 1.

Notice that the complexity is linear in the size of the input tensor: Let $E$ be the number of non zeros in the tensor, we can easily show that each iteration has complexity $O(E)$ as we only need to consider the non zero $X_{ijk}$ values. In practice, we select an $\epsilon$ and compare two consecutive iterations in order to stop the method when convergence is achieved. In our experimental analysis in Section 6.3 (using networks with millions of nodes) we saw that a relatively small number of iterations (about 10) is sufficient to provide reasonable convergence.

### 6.2.2 Community construction

Since the tensor decomposition provides numerical values for each node/label, its result cannot be directly used to specify communities. Additionally, there might be no clear threshold to distinguish those nodes/labels belonging to the community and the rest. Algorithm 5 illustrates the construction process in pseudo-code. We exploit vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ as bias in a hill climbing search, with the goal of minimizing the MDL cost. We start by selecting a highly connected entry $(a_0, b_0, c_0)$ in the tensor as the initial seed $S_a = \{a_0\}$, $S_b = \{b_0\}$, $S_c = \{c_0\}$[2]. We then let the community grow incrementally: we randomly select nodes $v_a$, $v_b$ and label $v_c$ that are not currently part of the community but connected to it, using the score vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ as sampling bias. That is, given the current nodes $S_a$, $S_b$ and labels $S_c$, we sample according to

$$P(v_a = i) \propto \begin{cases} a_i & i \notin S_a \wedge \exists y \in S_b, z \in S_c : X_{i,y,z} = 1 \\ 0 & else \end{cases}$$
$$P(v_b = j) \propto \begin{cases} b_j & j \notin S_b \wedge \exists x \in S_a, z \in S_c : X_{x,j,z} = 1 \\ 0 & else \end{cases} \tag{6.7}$$
$$P(v_c = k) \propto \begin{cases} c_k & k \notin S_c \wedge \exists x \in S_a, y \in S_b : X_{x,y,k} = 1 \\ 0 & else \end{cases}$$

For each of these elements, we calculate the description length considering that we would add the element to the community. That is, we calculate MDL$_a$, MDL$_b$ and MDL$_c$

[2] We tested different methods with no significant differences found in the results since the subsequent steps of growing and shrinking lead to the selection of the most relevant edges and the removal of irrelevant ones. Selecting the edge $(i, j, k)$ with highest $min(a_i, b_j, c_k)$ provides a good initial seed.

```
input  : $\boldsymbol{a}$ - scores of the first mode
input  : $\boldsymbol{b}$ - scores of the second mode
input  : $\boldsymbol{c}$ - scores of the labels
output : $\boldsymbol{S_a}$ - set of first-mode nodes
output : $\boldsymbol{S_b}$ - set of first-mode nodes
output : $\boldsymbol{S_c}$ - set of labels
[$\boldsymbol{S_a}$,$\boldsymbol{S_b}$,$\boldsymbol{S_c}$] $\leftarrow initialSeed$($\boldsymbol{a}$,$\boldsymbol{b}$,$\boldsymbol{c}$)
repeat
    $t \leftarrow 0$
    while $t < \Delta$ do                                   // try to grow the community
        $v_a \leftarrow newBiasedNode$([$\boldsymbol{S_a}$,$\boldsymbol{S_b}$,$\boldsymbol{S_c}$], $\boldsymbol{a}$)
        $v_b \leftarrow newBiasedNode$([$\boldsymbol{S_a}$,$\boldsymbol{S_b}$,$\boldsymbol{S_c}$], $\boldsymbol{b}$)
        $v_c \leftarrow newBiasedNode$([$\boldsymbol{S_a}$,$\boldsymbol{S_b}$,$\boldsymbol{S_c}$], $\boldsymbol{c}$)
        MDL$_a \leftarrow L_3(\boldsymbol{S_a} \cup \{v_a\}, \boldsymbol{S_b}, \boldsymbol{S_c})$
        MDL$_b \leftarrow L_3(\boldsymbol{S_a}, \boldsymbol{S_b} \cup \{v_b\}, \boldsymbol{S_c})$
        MDL$_c \leftarrow L_3(\boldsymbol{S_a}, \boldsymbol{S_b}, \boldsymbol{S_c} \cup \{v_c\})$
        [value, index] = $\min($MDL$_a,$ MDL$_b,$ MDL$_c)$
        if $value < L_3(\boldsymbol{S_a}, \boldsymbol{S_b}, \boldsymbol{S_c})$ then
            $\boldsymbol{S}_{index} \leftarrow \boldsymbol{S}_{index} \cup \{v_{index}\}$
            $t \leftarrow 0$
        else
            $t \leftarrow t + 1$
        end
        // try to shrink the community
        foreach element $n$ in $\boldsymbol{S_a}$ do
            if $L_3(S_a \backslash \{n\}, S_b, S_c) < L_3(S_a, S_b, S_c)$ then  $S_a \leftarrow S_a \backslash \{n\}$
        end
        foreach element $n$ in $\boldsymbol{S_b}$ do
            if $L_3(S_a, S_b \backslash \{n\}, S_c) < L_3(S_a, S_b, S_c)$ then  $S_b \leftarrow S_b \backslash \{n\}$
        end
        foreach element $n$ in $\boldsymbol{S_c}$ do
            if $L_3(S_a, S_b, S_c \backslash \{n\}) < L_3(S_a, S_b, S_c)$ then  $S_c \leftarrow S_c \backslash \{n\}$
        end
    end
until [$S_a$,$S_b$,$S_c$] has converged
return [$S_a$,$S_b$,$S_c$]
```

**Algorithm 5:** COM$^2$: Community Construction

based on the sets $S_a \cup \{v_a\}$, $S_b \cup \{v_b\}$ and $S_c \cup \{v_c\}$, respectively. If the smallest of these MDL scores is smaller than the score of the community detected so far, the corresponding element is accepted and the next round of sampling is performed. This process is repeated until $\Delta$ consecutive rejections have been observed. We can show that a small number of rejections $\Delta$ is sufficient:

**Lemma 6** *Let $i$ be an element that was not included in the community when it should have been included. Let $\mathbf{u}$ be the vector corresponding to $i$'s mode (i.e. $\mathbf{u}$ is one of the vectors $\mathbf{a}$, $\mathbf{b}$, or $\mathbf{c}$). Then the probability that $i$ does not belong to this community decreases exponentially with $\Delta$.*

$$P(\text{“}i\text{ not selected”}|\text{“}i\text{ should have been selected”}) \leq (1-u_i)^{\Delta}. \tag{6.8}$$

**Proof 6** *Given that vector $\mathbf{u}$ is normalized (see step 1), at each iteration, the probability that the element $i$ is not chosen is given by $(1-u_i)$. After $\Delta$ iterations, the probability that the element has not been chosen is upper-bounded by $(1-u_i)^{\Delta}$. The exact probability is actually lower as the sampling is done without replacement, ignoring the elements currently in the community.* ■

In our experimental analysis, a value of $\Delta = 50$ has proven to be sufficient; we consider this parameter to be general and it does not need to be defined by the user of the algorithm.

After growing the community (i.e. after $\Delta$ rejections), we try to improve its description cost by removing elements. Intuitively, it is possible that one of the nodes initially selected to be part of the community (when it was small) is not that well connected to the nodes that have since been added. Instead of penalizing the current MDL score and “blocking” the addition of new nodes, we check whether the removal of any node/label currently in the community improves the description cost. This growing/shrinking alternating process is repeated until the community stabilizes, and it is guaranteed to converge as the description cost is strictly decreasing.

### 6.2.3 Tensor deflation

While the output of the previous two steps is a *single* community, the goal of this step is to transform the tensor so that novel communities can be found in future iterations. The challenge of such an iterative processing is to avoid generating the same community repeatedly: we have to explore different regions of the search space.

Note that [PSB13] indicates that extracting one rank (i.e. community) at a time approximates the full-rank decomposition with very high accuracy when the factors are sparse. Therefore, we propose the principle of tensor deflation. Starting with the original tensor $\boldsymbol{\mathcal{X}}^{(1)} := \boldsymbol{\mathcal{X}}$, after each iteration, we remove the community $C_i$ whose edges were already described. We obtain the recursion

$$\boldsymbol{\mathcal{X}}^{(i+1)} := \boldsymbol{\mathcal{X}}^{(i)} - \boldsymbol{\mathcal{I}}^{C_i} * \boldsymbol{\mathcal{X}}^{(i)} \qquad [= \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}^{\mathcal{C}_i} * \boldsymbol{\mathcal{X}}\ ] \tag{6.9}$$

where $*$ denotes the Hadamard product (see Chapter 2).

The method might terminate when the tensor is fully deflated (if possible), or when a pre-defined number of communities has been found, or when some other measure of community quality (e.g. community size) was not achieved in the most recent communities.

### 6.2.4 Complexity Analysis

**Lemma 7** *Our algorithm has a runtime complexity of*

$$O(C \cdot (E + |P| \cdot \log N \cdot \log |P|)),$$

*where $C$ is the number of communities we obtain, $E$ is the number of non-zeros of the tensor, $N$ is the length of the biggest mode, and $|P|$ is the size of the biggest community. Thus, our method scales linearly w.r.t. the input $E$.*

**Proof 7** *Steps 1 to 3 are repeated C times, the number of communities to be obtained. Step 1, the rank-1 approximation, requires $O(E)$ time. Step 2, the core of the algorithm, can be executed using $O(|P|)$ addition and removals, each with the complexity required to calculate the new Minimum Description Length of the community: $O(\log N \cdot \log |P|)$. Finally, step 3, the matrix deflation, can be done in $O(E)$ with a single pass over the edges of the community.*
■

### 6.2.5 Algorithm parameters

Despite the existence of two parameters in the algorithm, their variation has no significant impact when analyzing specific networks.

The first parameter, $\epsilon$, impacts the number of iterations in the rank-1 approximation in step 1. In practice, a fixed value of 10 iterations provides very good results regardless of the network under consideration, an effect that can be explained due to two reasons: firstly, vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are only used as approximations for community candidates and don't require high precision. Secondly, since real graphs are scale-free having small diameter, changes in these vectors propagate very quickly through the network.

| Name | #Nodes | #Non-zeros | #Labels | Description |
|---:|---:|---:|---:|:---|
| OLB | 10-20 | 1 000 - 2 000 | 100 | Overlapping blocks. |
| DJB | 1 000 | 50 000 | 500 | Disjoint blocks. |
| LBNL | 1 647 + 13 782 | 113 030 | 30 | Internet traces from LBNL. |
| PHONE | 3 952 632 | 51 119 177 | 14 | Phone call network. |
| FLIGHTS | 7 733 | 67 663 | 5 995 | Flights network. |

**Table 6.1: Networks used: two small synthetic networks and three large real networks.**

The impact of the second parameter, $\Delta$, has been analyzed in Lemma Lemma 6. The exponential decrease in a node's probability to be wrongly left out of the community implies that a relatively small and fixed value for $\Delta$ can be used.

Therefore, we conclude that these parameters do not need to be defined by the user.

## 6.3 Experiments

COM$^2$ was tested on a variety of real and synthetic tensors in order to assess it's effectiveness, robustness and scalability. Table 6.1 summarizes the networks used, a more detailed description of each dataset is provided later in this section.

In the three fairly different real-world datasets, COM$^2$ was run using the default parameters (cf. Section 6.2.5), showing that it can be applied without any user intervention.

Q1. **Community structure**: How relevant are the communities found? How are they reported when they overlap? How does their density impacts COM$^2$?

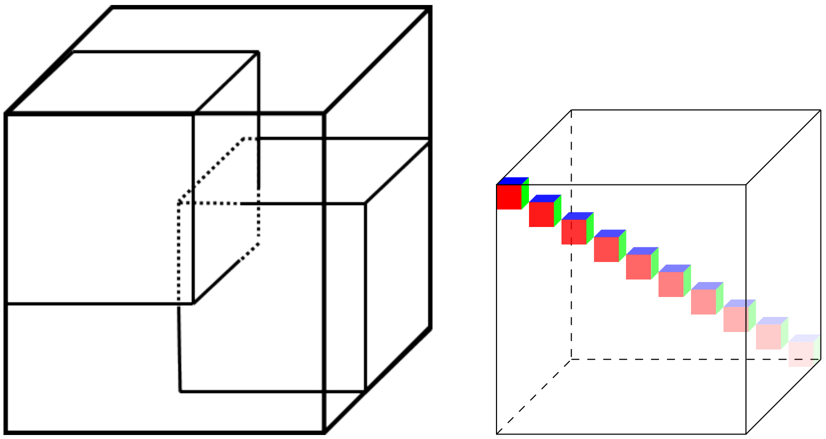
Q2. **Scalability**: How fast is COM$^2$?

Q3. **Discoveries**: How capable is COM$^2$ of detecting meaningful labeled communities?

### 6.3.1 Q1 - Community Structure

Characterizing the quality and robustness of the communities identified by the method is important. In particular, we want to answer the following questions: How are "overlapping blocks" identified? How much overlapping can occur so that different rank-1 decompositions can identify them separately? How "dense" are the communities found?

We rely on synthetic datasets with ground-truth information to answer these questions.



(a) Tensor with overlapping blocks: Illustration of the tensors used in the first experiment, the number of overlapping edges of the two blocks was variable.

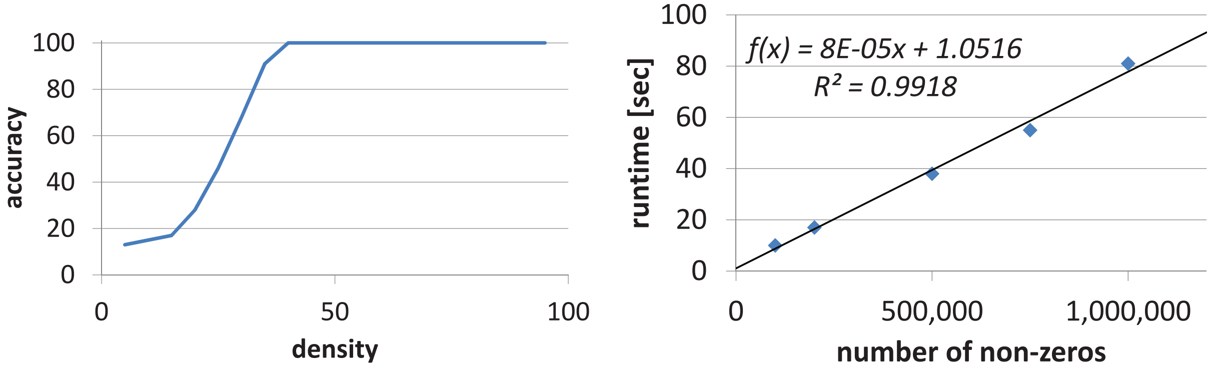
(b) Communities with different densities: Illustration of the tensors used in the second experiment. Opacity indicates the non-zeros density in the blocks.

Figure 6.2: Synthetic datasets.

### Overlapping communities

Analyzing the impact of overlap helps us predict when two distinct communities will be reported as a single entity and, equivalently, how connected internally a community needs to be so that it will not be split in two separate communities by the algorithm.

A tensor with two disjoint and cubic communities was constructed and, iteratively, elements from each of the modes of one of the communities were replaced with elements of the other (see Figure 6.2a). Our tests show that the communities are reported as independent until there is an overlap of about 70% of the elements in each mode, in which case they start being reported as a single community. This corresponds to an overlap of slightly over 20% of the non-zero values of the two communities and the global community formed has 63% of non-zeros. This clearly demonstrates that Com$^2$ has high discriminative power: it can detect the existence of communities that share some of their members and it is able to report them independently, regardless of their size. Note that, due to the 3-dimensional nature of our data, a relatively high overlap of the modes does not immediately correspond to an high overlap of the non-zeros.



**(a) Tensor with disjoint blocks - Com$^2$ identifies communities even at low densities.**

**(b) Com$^2$ scales linearly with input size: Running time versus number of non-zeros for random tensors.**

**Figure 6.3: Experiments on synthetic data.**

### Impact of block density

We also performed experiments to determine how density impacts the number of communities found (see Figure 6.2b). Fifty disjoint communities were created in a tensor with random noise and non-zeros were sampled without repetition from each community with different probabilities. We then analyzed the first fifty communities reported by Com$^2$ in order to calculate its accuracy. As we show in Figure 6.3a, the discriminative power remains high, even with respect to varying density.

## 6.3.2 Q2 - Scalability

As previously detailed, Com$^2$'s running time is linear on the number of communities and in the number of non-zero values in the tensor. We constructed a tensor of size $10000 \times 10000 \times 10000$ and randomly created connections between sources and destinations using random labels. Figure 6.3b shows the runtime versus the number of non-zeros in the tensor when calculating the first 200 communities of the tensor. In addition to its almost linear runtime, Com$^2$ is also easily parallelizable. By selecting different random seeds in the tensor decomposition step, different communities can be found in parallel.

## 6.3.3 Q3 - Discoveries on edge-labeled graphs

Com$^2$ was applied to a dataset of flight routes from 2012 available at `http://openflights.org/data.html` (cf. Table 6.1, Flights). In this setting, nodes correspond to airports and edges are labeled with the airline company performing the route (i.e. there might be more than one edge between each pair of nodes). Our goal is to find a set of companies flying several routes between a set of airports, a strong indicator of local competition. Even though the underlying graph is directed, we chose to work with a single set of airports instead of separating origin and destination sets. For this purpose, we adapted the previously described algorithm so that the sampled vertex is added to both modes: the origin and destination set.

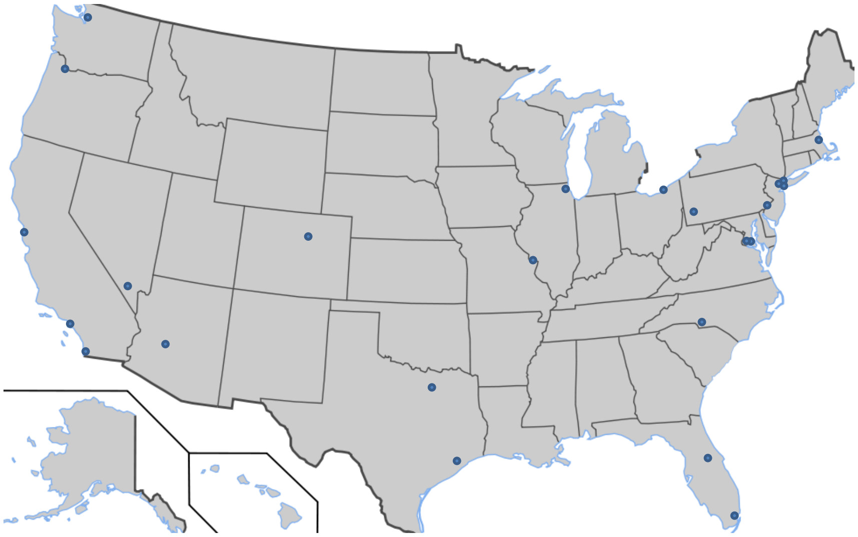
Figure 6.1, depicted in the introduction, illustrates the most international of these communities, with 16 worldwide airports and 3 companies well known for intercontinental travel: Lufthansa, Delta and United Airlines. In order to show COM$^2$'s effectiveness, we showcase three regional communities of competing companies:

- Figure 6.4a and Figure 6.4b represent the major competing companies in the United States of America and China, along with their respective airports. The community pictured in Figure 6.4a corresponds to 26 American airports; US Airways, United and American Airlines operate 915 different routes between these 26 airports. Figure 6.4b shows 25 Chinese airports; Hanan Airlines, Air China, China Southern Airlines and China Eastern Airlines operate 1,150 routes between these airports. These two examples show COM$^2$'s effectiveness in identifying dense subgraphs sharing similar edge-labels.
- Figure 6.5 shows that COM$^2$ is also able to find single-label communities. Ryanair alone operates 988 different routes between 47 European airports. This community can be seen as a dense subsection of the tensor, which is the equivalent to a big star in the unlabeled case (i.e. a dense row/column in a matrix).

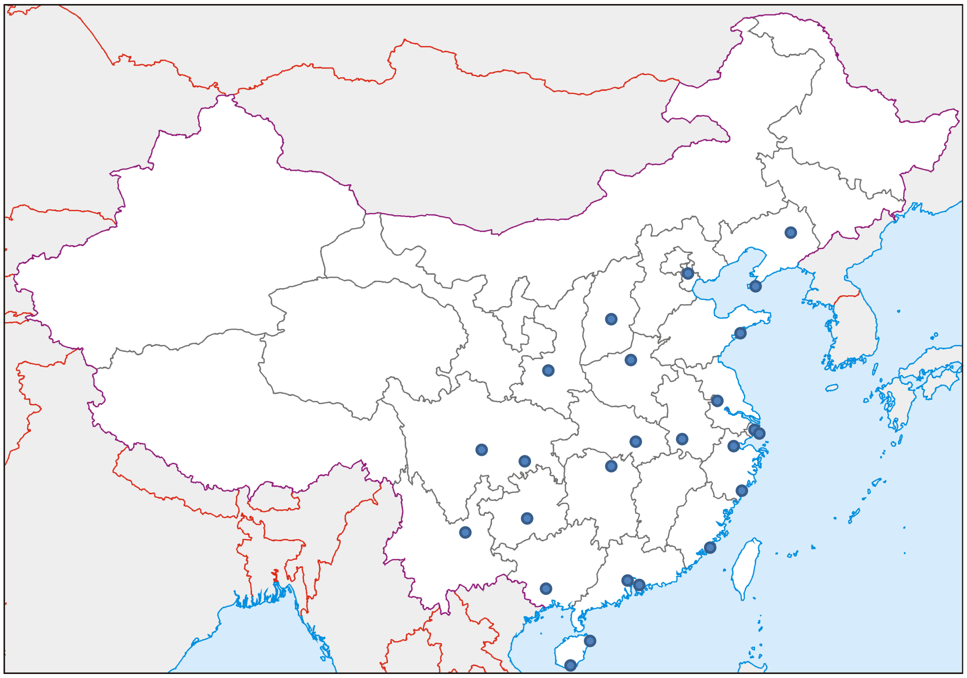
Please note that neither standard community detection algorithms operating on the unlabeled graph, nor multiple runs considering each company independently, could possibly find the competing companies scenario as it requires interaction between several different edge-labels.

### 6.3.4 Q3 - Discoveries on time-labeled graphs

To characterize communities found in real phone-call data, we applied COM$^2$ to a dataset from an anonymous European mobile carrier. We considered the network formed by calls between clients of this company over a period of 14 days. During this period, 3,952,632 unique clients made 210,237,095 phone calls, 51,119,177 of which formed unique (caller, callee, day) triplets (cf. Table 6.1, PHONE). Here, each label corresponds to a specific day. The tensor is very sparse, with density in the order of $10^{-7}$. We extracted 900 communities using COM$^2$. These communities contain a total of 229,287 unique non-zeros. 293 unique



(a) Community in the United States: US Airways, United and American Airlines operate 915 different routes (47%) between these 26 airports.



(b) Community in China: Hanan Airlines, Air China, China Southern Airlines and China Eastern Airlines operate 1,150 routes (48%) between these 25 airports.

Figure 6.4: Regional communities of competing companies found using flight routes.

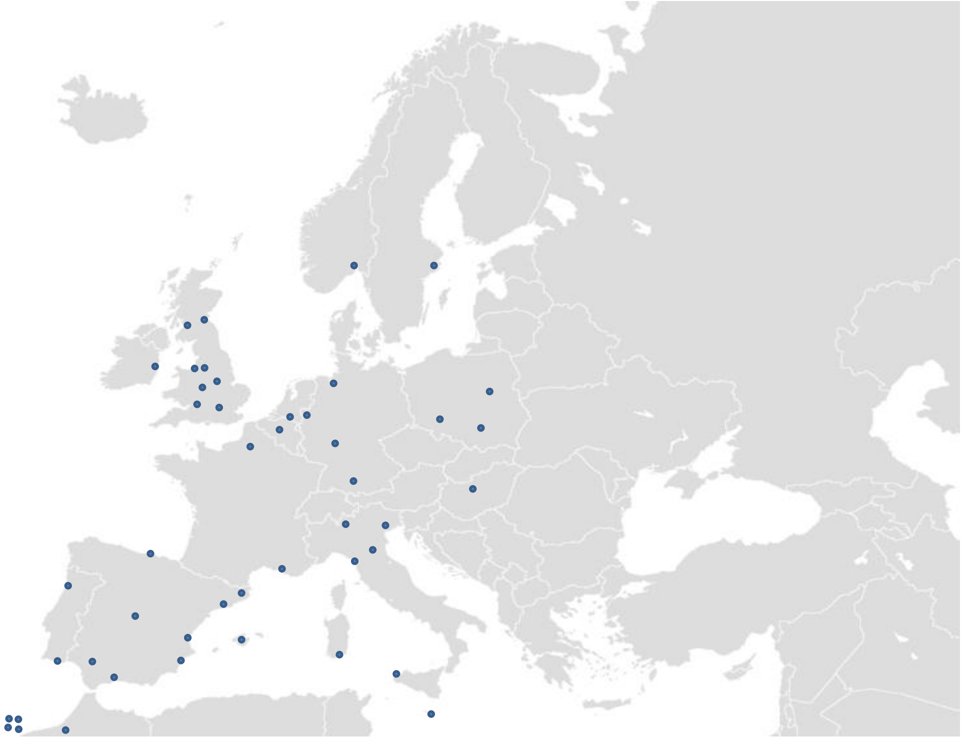


**Figure 6.5: Community in Europe: Ryanair creates near-cliques on its own. It operates 988 unique routes (46% of total possible) between these 47 airports.**

callers and 97,677 unique callees are represented, so the first observation is that the temporal communities are usually heavy on one side with large outgoing stars.

We also applied COM$^2$ to a public computer network dataset captured in 1993, made available by the Lawrence Berkeley National Laboratory [PF95]. 30 days (i.e. edge labels) of TCP connections between 1,647 IP addresses inside the laboratory and 13,782 external IP addresses were recorded (cf. Table 6.1, LBNL). This tensor was completely deflated and a total of 19,046 communities were found (1,930 of them having more than 9 non-zeros).

**Observation 1.** *Community Activity*
The biggest communities are more active during weekdays.

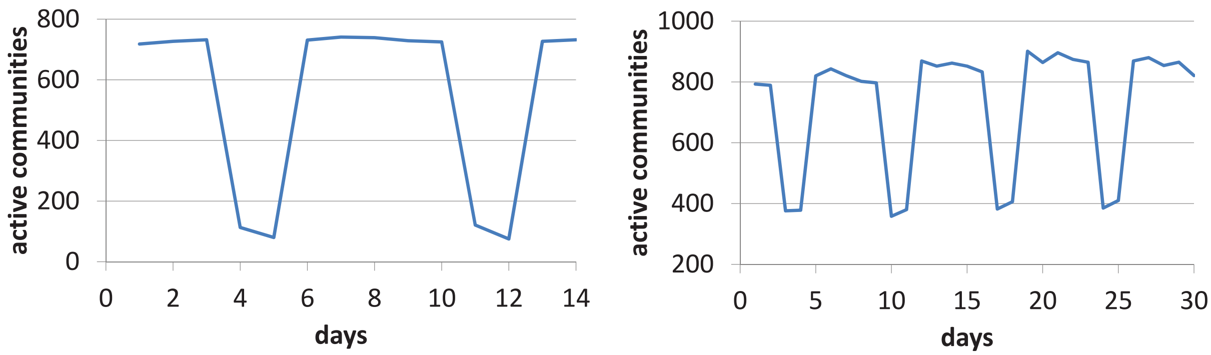
Figure 6.6 shows the number of active communities per day of the week on both datasets and we can see that most communities are significantly more active during weekdays. In the phone call data, we are led to believe that these are mostly companies with reduced activity during weekends, while the reduced activity during the weekends in the research laboratory is to be expected.

**Observation 2.** *Flickering stars*
A typical pattern is the *"Flickering stars"*, subgraphs whose temporal activity is not constant over time.

When analyzing a phone call network, a pattern to be expected is the marketeer pattern in which a single number calls many others a very small number of times (1 or 2). Surprisingly, the stars reported by COM$^2$ were not of this type. Two callers stand out in an analysis of the communities reported: one participated in 78,279 (source, destination, time) triplets as a caller but only in 10 triplets as a receiver, while the other participated in 8,909 triplets as a caller and in none as a receiver. These two nodes are centers of two distinct outgoing stars and were detected by the algorithm. However, the time component of these stars was not a single day but rather spanned almost all the weekdays. This behavior does not seem typical of a marketeer, so we hypothesize that it is a big company communicating with employees. Many of the reported communities are stars of this type: a caller calling a few hundred people in a subset of the weekdays - we call them flickering because, even though there is some activity during the rest of the weekdays, it is significantly reduced and those days are not reported as part of the community.

In the LBNL dataset, one star was particularly surprising. It received connections from over 750 different IP addresses inside the laboratory but only on a single day. One of the other big stars corresponded to 40 connections on a single day to an IP address attributed to the Stanford Research Institute, which is not surprising given the geographical proximity.

We define *Flickering stars* as a common temporal-community that has a varying number of receivers. These communities are active on different days, not necessarily consecutive.



**(a) Weekly periodicity phone call data.** **(b) Weekend activity computer network data.**

**Figure 6.6: Weekly periodicity: number of active communities vs time. Notice the weekend dives on a) days 4, 5 and 11, 12 and b) days 3, 4, 10, 11, 17, 18, 24, 25.**

Stars active on many days (e.g. every weekday) are more common than single day stars.

**Observation 3.** *Temporal Bipartite Cores*
Another typical pattern is the *"Temporal Bipartite Core"*: dense subgraphs whose temporal activity is not constant over time.

Several near-bipartite cores were detected as communities in the phone call dataset. These are communities with about 5 callers and receivers that are active on nearly each day under analysis, and each represents between 75 and 150 of the non-zeros of the original tensor, with a block density of around 40%.

An example of such communities can also be shown for the LBNL data (Figure 6.7). 7 machines of the laboratory communicated with 6 external IP addresses on every weekday of the month. After analyzing their IP addresses, the outside machines were found to be part of the Stanford National Accelerator Laboratory, the University of California in San Francisco, the UC Davis, the John Hopkins University, and the U.S. Dept. of Energy. COM$^2$ was able to detect this research group (possibly in particle physics) using communications data alone.

## 6.4 Summary

Table 6.2 compares some of the most common static and label-aware community detection methods.

COM$^2$ carefully combines a fast and efficient iterated rank-1 tensor decomposition to guide the search for nodes and labels that participate in communities, and a principled

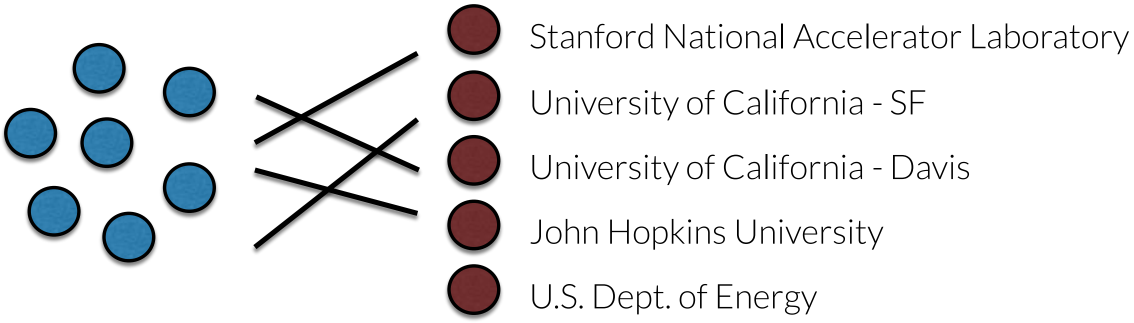

Stanford National Accelerator Laboratory
University of California - SF
University of California - Davis
John Hopkins University
U.S. Dept. of Energy

**Figure 6.7: LBNL community: Com$^2$ detects research group collaborations using computer communications data.**

| | Com$^2$ | Eigenspokes [PSS$^+$10] | METIS [KK95] | Graphscope [SFPY07] | PARAFAC [Har70] | SDP + Rounding [TBW11] | GMM-NK [WYC$^+$13] | PMM [TWL09] | GraphFuse [PAI13] |
|---|---|---|---|---|---|---|---|---|---|
| **Scalability** | ✔ | ✓ | ✓ | ✓ | | | ✓ | | ✓ |
| **Label-awareness** | ✔ | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Label subsets**$^*$ | ✔ | N/A | N/A | | ✓ | ✓ | ✓ | ✓ | |
| **No parameters** | ✔ | ✓ | | ✓ | | | | ✓ | ✓ |
| **Interpretability**$^‡$ | ✔ | ✓ | | ✓ | | ✓ | | ✓ | |
| **Overlapping** | ✔ | ✓ | | | ✓ | | ✓ | | |

$^*$ Communities found in subsets of labels; temporal communities do not need to be contiguous.
$^‡$ Results are easy to interpret; elements of the community can be identified easily.

**Table 6.2: Comparison of community detection methods.**

MDL-based model selection criterion that guides the expansion of communities and provides a stopping mechanism. We have focused on binary tensors, which reveal structural (connectivity) community patterns over edge-labeled graphs, and have demonstrated interesting findings in a variety of real-world datasets. The main contributions are the following:

- **Scalability**: Our method, COM$^2$, is linear on the input size; instead of relying on a complete tensor factorization, we carefully leverage rank-1 decompositions to incrementally guide the search process for community detection.
- **No user-defined parameters**: In addition to the above, efficient, incremental search process, we also proposed a novel MDL-based stopping criterion, which finds communities in a parameter-free fashion.
- **Effectiveness**: We applied COM$^2$ on real and synthetic data, where it discovered communities that agree with intuition.
- **Generality**: COM$^2$ can be easily extended to handle higher-mode tensors.

COM$^2$ is available at http://www.cs.cmu.edu/~maraujo/comdet/com2.html.

**Discussion and Future Work.** Our current methods requires categorical edge labels. Extending MDL to handle real numbers, as opposed to integer values, is a challenging problem. Furthermore, real-valued (possibly continuous, but non-categorical in general) edge labels render tensor representations impossible (i.e. we can't represent non-categorical indices). However, tensor decompositions can be applied to weighted tensors (e.g. representing the strength of connections), potentially enabling interesting findings.

Future work can also focus on expanding our principle to coupled tensor-matrix data, in order to exploit node-related side information such as demographic data. This research direction would provide unified tools to find communities in networks with both edge labels and node attributes. We explore a scenario in which coupled information is used for forecasting in Chapter 8.

Chapter based on work that appeared at MLG 2016 [HAF16].

# Chapter 7

# Distributed Community Detection

In the previous Chapter, we saw how MDL could be used to select nodes and labels to be part of labeled communities. In this Chapter, we follow a similar approach but focus on creating a distributed algorithm instead, exploring simper heuristics. Note that processing *billion-sized* graphs introduces significant memory and time burdens for conventional algorithms; we consider it relevant to evaluate how methods and heuristics that can be trivially distributed impacts their performance.

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 7.1 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 non-distributed 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 7.1. For example, our system is able to distinguish low-cost airlines and standard airlines in Europe.

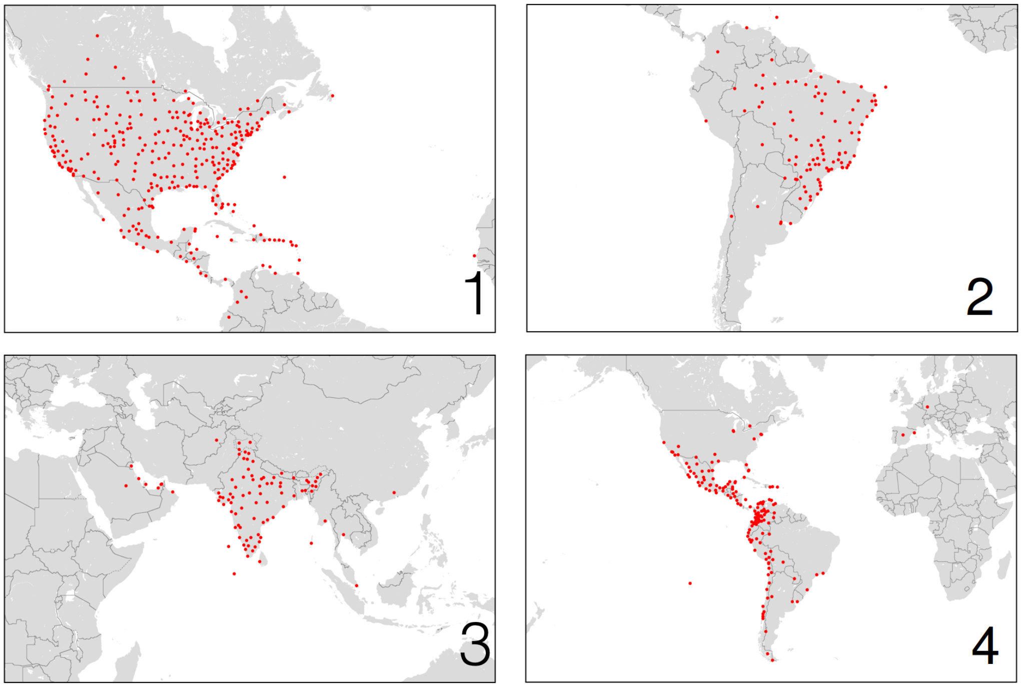


**Figure 7.1: Communities detected in flights dataset: (1) America, (2) Brazil, (3) India, (4) Latin America. Each red dot represents an airport.**

## 7.1 Proposed System

Similarly to the work previously presented, we start from a rank-1 standard PARAFAC [Har70] model. 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 7.2, each round of our method is composed of three steps:

- **Step 1: Factorization:** We use a rank-1 decomposition to extract one set of factor vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$.
- **Step 2: Thresholding:** We establish which elements belong to the current community by thresholding the previously obtained score vectors.
- **Step 3: Tensor deflation:** We save the marked entries as a community and remove the non-zero elements corresponding to this community from the tensor.

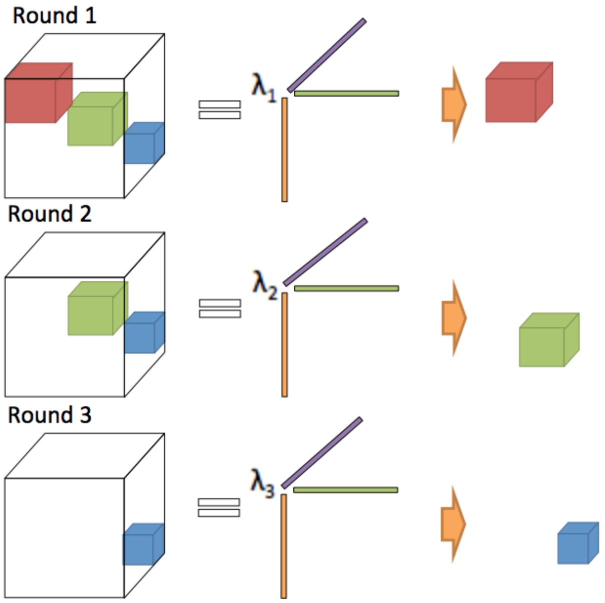


**Figure 7.2: Community detection with our system**

```
input  : $\boldsymbol{E}$ - edge list
input  : $R$ - rank of the decomposition
output : $\boldsymbol{S}$ - list of communities

for $r = 1$ to $R$ do
    $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \leftarrow$ rank1decomposition($\boldsymbol{E}$)
    foreach $e$ in $\boldsymbol{E}$ do
        if $a_{e.i} \geq 1/|\boldsymbol{a}|$ and $b_{e.j} \geq 1/|\boldsymbol{b}|$ and $c_{e.k} \geq 1/|\boldsymbol{c}|$ then
            Add $e$ to $s_r$
            Remove $e$ from $\boldsymbol{E}$
        end
    end
end
return $\boldsymbol{S}$
```

**Algorithm 6:** TeraCom

### 7.1.1 Factorization

Further information about rank-1 factorizations and a detailed proof can be found in Chapter 2, so we won't go into much detail. Succinctly, given a three mode tensor $\boldsymbol{\mathcal{X}}$, we want to find vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ which approximate tensor $\boldsymbol{\mathcal{X}}$ with minimal error

$$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} = \underset{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}}{\operatorname{argmin}} \sum_i \sum_j \sum_k (\boldsymbol{\mathcal{X}}_{i,j,k} - \lambda \cdot \mathbf{a}_i \cdot \mathbf{b}_j \cdot \mathbf{c}_k)^2 \tag{7.1}$$

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

$$\mathbf{a}_i = \frac{1}{Z_a} \sum_j \sum_k \mathbf{b}_j \cdot \mathbf{c}_k \cdot x_{i,j,k} \tag{7.2}$$

$$\mathbf{b}_j = \frac{1}{Z_b} \sum_j \sum_k \mathbf{a}_i \cdot \mathbf{c}_k \cdot x_{i,j,k} \tag{7.3}$$

$$\mathbf{c}_k = \frac{1}{Z_c} \sum_j \sum_k \mathbf{a}_i \cdot \mathbf{b}_j \cdot x_{i,j,k} \tag{7.4}$$

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

### 7.1.2 Thresholding

When building a community, one needs to decide how it is being defined. Is a community a set of nodes and labels or is it a set of labeled edges? This will impact algorithm design as node and label selection is different than edge selection. We opted for edge selection and the most simple and easy to distribute mechanism is thresholding. The goal is to select an appropriate threshold to be applied to the score vectors above which edges are deemed to belong to the community.

We opted for considering the average score as the threshold parameter. We marks an entry $e_{ijk}$ 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.

### 7.1.3 Tensor Deflation

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.

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.

### 7.1.4 Implementation Design

We implement the algorithm based on Spark framework. Spark is a distributed framework can utilize multiple machines to make algorithm running in parallel and help us handle complex situations in distributed computation environment (like fault-tolerance). Also, it stores intermediate computation result in memory, which reduces slow Disk I/O and performs efficiently. 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.
- We **encode shared state using broadcast variables**. For each inner-iteration, we broadcast the latest version of vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ (which are small) to all machines. For each dimension, non-zeros are then mapped to the respective product of Equation 7.2, aggregated and normalized.

Listing 7.1 provides lower-level details to the interested reader.

**Listing 7.1** The Snippet of Rank-1 Decomposition Code

```
def r1_decomposition(
  sc:SparkContext,
  E:RDD[Array[Int]]) :
    (Map[Int, Float], Map[Int, Float], Map[Int, Float]) = {

    var A = genNormalizedSparseVector(E, 0)
    ...

    val A_E = E.groupBy(x => x(0))
   A_E.cache()
    ...

    for (dr <- 0 until kDecomposedRound) {
      val bA = sc.broadcast(A)
      ...

      val newA = A_E
        .mapValues(entries =>
          entries.map(e => bB.value(e(1)) * bC.value(e(2))).sum)
      ...

      val sumA:Float = newA.map(x => x._2).sum().toFloat
      A = newA.mapValues(v => v / sumA).collectAsMap()
      ...
    }

    (A, B, C)
}
```

## 7.2 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?

**Table 7.1: Dataset description: 2 synthetic and 3 real-world datasets.**

| **Name** | $\|\mathcal{V}\|$ | $\|\mathcal{E}\|$ | $\|\mathcal{L}\|$ | **#communities** |
|---|---|---|---|---|
| **10-cubes** | 130 | 33 786 | 150 | 10 |
| **5-disjointed-subgraph** | 1K – 1M | 100K – 100M | 1K – 1M | 5 |
| **DBLP-top100** | 2 244 | 5 054 | 45 | 100 |
| **Airline** | 3 162 | 58 443 | 532 | N/A |
| **NELL** | 74M | 144M | 26M | N/A |

### 7.2.1 Q1 - 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 [Har70] and non-negative [WW01] PARAFAC.

**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 applying 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 [BKo15] as the default implementation of PARAFAC and Non-Negative PARAFAC.

**Data Description.** We use three datasets with ground-truth communities to evaluate accuracy: two synthetic datasets with artificial communities and the DBLP dataset [Ley02], where venue 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.

**Experimental Results.** We measured Normalized Mutual Information (NMI[1]) [YL12b] 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 7.2 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.

**Table 7.2: Precision Experiment**

| 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 |

### 7.2.2 Scalability

**Q2 - 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 [BKo15] - a tensor decomposition tool which is the state of the art tensor computation package designed for a single machine [JPKF15].

The number 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

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

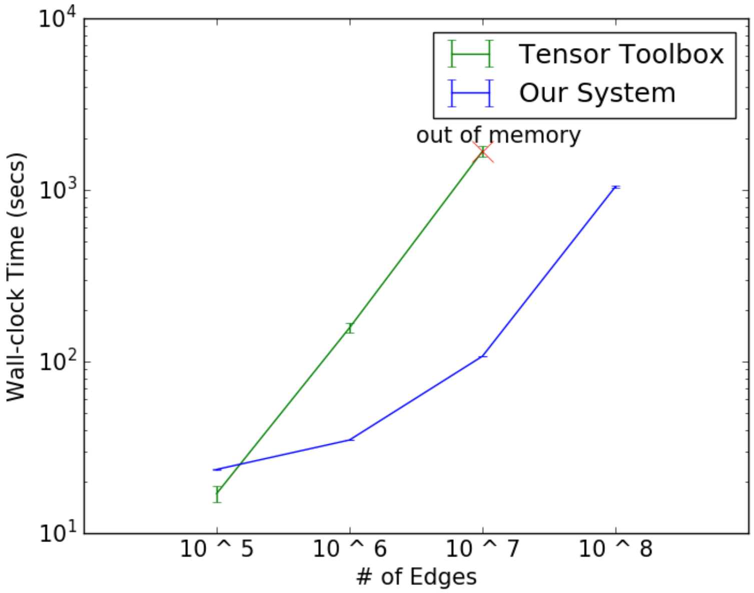


**Figure 7.3: Data Scalability Experiment: Tensor Toolbox (1 machine) v.s. Our System (3 machines)**

system to achieve a lower error, and finally compare their running time at this precision level.

We run the Tensor Toolbox on a single machine with 15GB of memory and 12 Intel® Xeon® 3.20GHz CPUs. We built 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 7.3 shows the experimental results: our system can be applied to 10X larger graphs, while still running faster 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 order to evaluate the third graph which has $10^7$ edges, the Tensor Toolbox requires 1679.89 seconds on average while 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 dispatches 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 7.4 shows the wall-clock time of each outer-iteration as our system processes

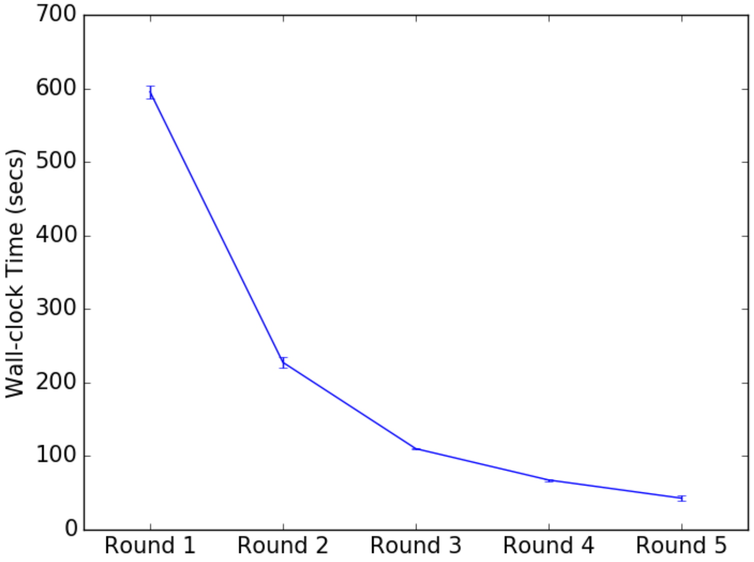


**Figure 7.4: Execution time decreases in each outer-iteration.**

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.

**Q3 - 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 the 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 7.5.

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;

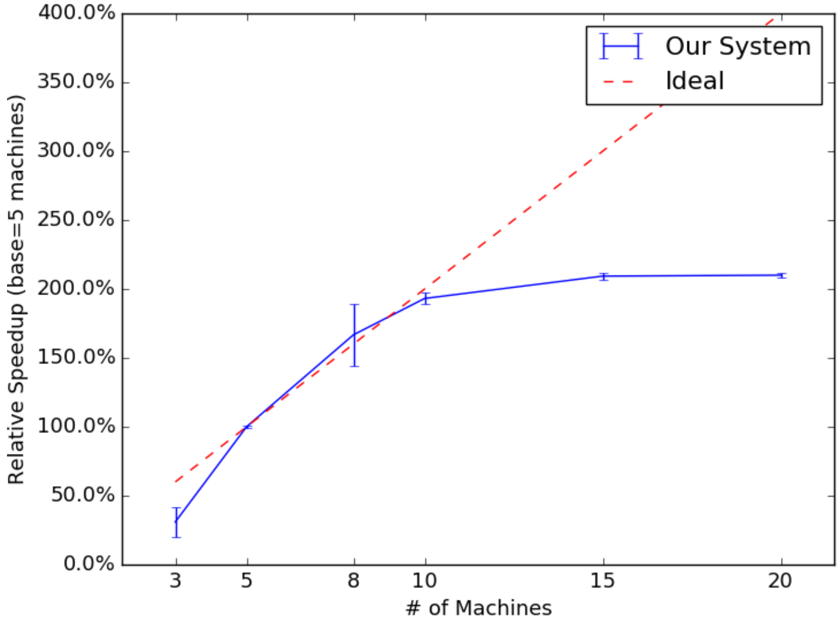


**Figure 7.5: Machine Scalability (Base: 5 machines)**

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.

Three main reasons explain 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 to 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.

### 7.2.3 Q4 - Discoveries

We apply our system to an Airlines dataset previously described in Chapter 4. Each node corresponds to an airport and a link represents an air route. Edge labels 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 7.1.

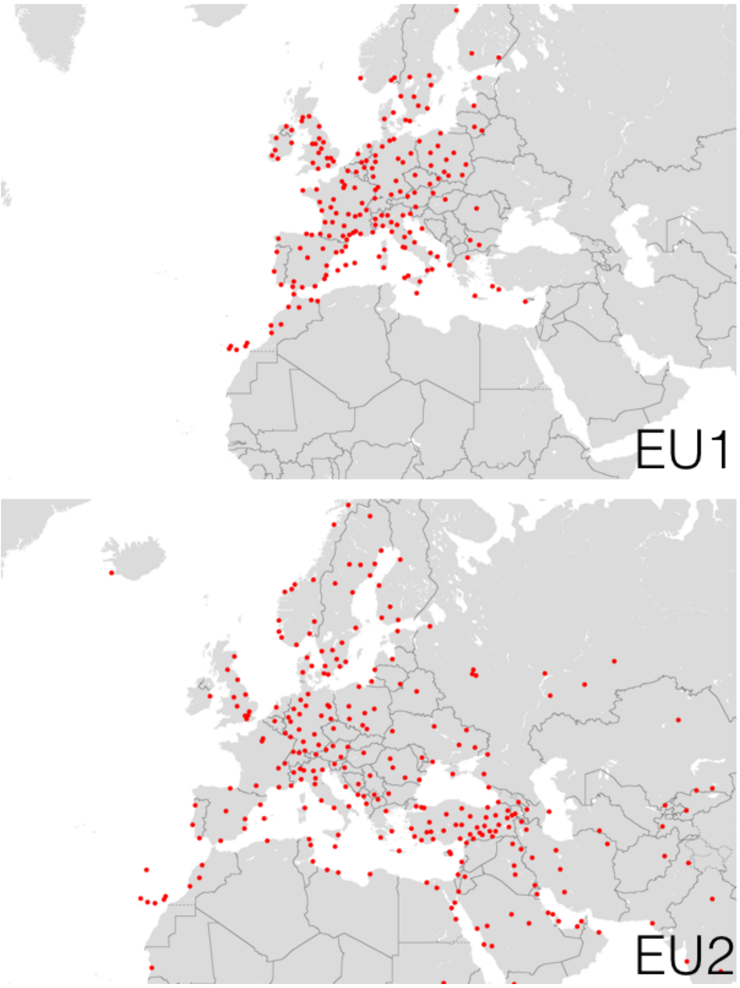


**Figure 7.6: Two clusters of different European airlines. The air routes in EU1 are mainly hosted by low-cost airlines, while routes in EU2 are hosted by regular airlines.**

More interestingly, our system is able to use the edge labels to distinguish two airline clusters in Europe, EU1 and EU2 shown in Figure 7.6. 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 [lis]. In contrast, only 16.74% of the routes in EU2 are hosted by companies on this list. This result is similar to what was achieved in Chapter 4, but TeraCom incorporates airline information in edge labels.

## 7.3 Related Work

We provide an overview of the relevant related work in distributed tensor decompositions.

**Tensor Analysis and Tools** Multiple tensor analysis toolboxes implementing PARAFAC have been developed. While these systems implement the standard PARAFAC [Bro97] 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 [ZCF$^+$10] which allows efficient memory-based operations, not requiring data to be spilled to disk.

Tensor Toolbox [BKo15] is a well-known MATLAB implementation of many tensor decomposition algorithms, such as non-negative PARAFAC with alternative Poisson regression [HPK15] and PARAFAC via optimization [ADK11]. ParCube [PFS12] 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 [KPHF12] developed a large-scale PARAFAC [KB09] 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 [JPKF15] 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 [ZCF$^+$10] 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.

## 7.4 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 7.1. For example, our system is able to distinguish low-cost airlines and standard airlines in Europe.

# Chapter 8

# Forecasting Communities

If a group has been discussing the *#elections* on Twitter, with interest steadily increasing as election day comes, can we predict who is going to join the discussion next week? Intuitively, our forecast should take into account other hashtags (#) that have been used, but also user-user interactions such as followers and retweets.

Similarly, can we predict who is going to publish on a given conference next year? We should be able to make use of, not only the data about where each author previously published, but also co-authorship data and keywords that might indicate a shift in interests and research focus.

Today's data sources are often heterogeneous, characterized by different types of entities and relations that we should leverage in order to enrich our datasets. In order to predict the evolution of some of these interactions, we propose to model these heterogeneous graphs as Coupled Tensors that, when forecasted jointly, generate better predictions than when considered independently.

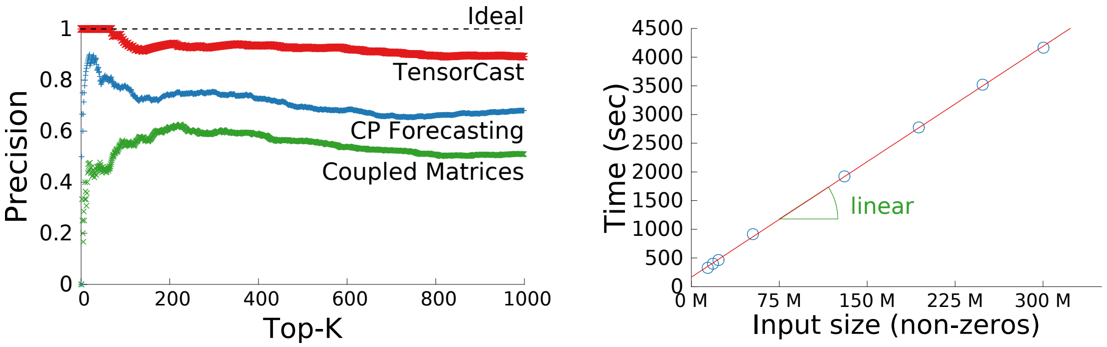
In particular, we will show how the evolution of user to user connections can be used to forecast user to entity relations, e.g. information about who retweets whom improves the prediction of who is going to use a given hashtag, and co-authorship information improves the prediction of who is going to publish at a given venue.

**INFORMAL PROBLEM 1. Forecasting Interactions**

- **Given** historical interaction records between different users and between users and entities.
- **Find** the most likely interactions in the future.

Using a *naive* approach, one would have to individually forecast every pair of users and entities - a prohibitively big number that quadratically explodes. How can one avoid quadratic explosion during forecasting? How can we obtain the top-K most likely interactions without iterating through them all?

As a summary of our results, Figure 8.1a shows that our proposed TensorCast method is able to achieve 20% more precision than competing methods on the task of predicting who is going to publish on which venue in 2015 using DBLP data. Figure 8.1b shows TensorCast scaling to hundreds of millions of non-zeros on Twitter data.



**(a) Higher precision when forecasting $(author, venue)$ relations in the DBLP tensor.**

**(b) TensorCast scales linearly with the number of non-zeros.**

**Figure 8.1: TensorCast is effective and scalable.**

We underline our main contributions:

1. **Effectiveness**: TensorCast achieves over 20% higher precision in top-1000 queries and double the precision when finding new relations than comparable alternatives.
2. **Scalability** : TensorCast scales well ($E + N \log N$) with the input size and is tested in datasets with over *300M* interactions.
3. **Context-awareness**: we show how different data sources can be included in a principled way.
4. **Tensor Top-K**: we show how to quickly find the K biggest elements of sums of three-way vector outer products under realistic assumptions.

## 8.1 Proposed Method: TensorCast

We assume a coupled-tensors setting where multiple tensors, possibly with different dimensions, are related by common modes. We will assume that at least one of these tensors is our tensor of interest: it is a 3-dimensional binary tensor and one of the modes corresponds to a time component which we would like to forecast.

There are many scenarios that can be instantiated under this setting: imagine the existence of membership records of the form *(user, topic, time)*, with $N$ unique users and $M$ unique topics (or communities) over $T$ unique time intervals encoded in a $3rd$-order tensor $\boldsymbol{\mathcal{X}} \in \{0,1\}^{N \times M \times T}$. Maybe we also have available an additional collection of user interaction records of the form *(user, user, time)*, similarly encoded in a $3rd$-order tensor $\boldsymbol{\mathcal{Y}} \in \{0,1\}^{N \times N \times T}$. One possible forecasting problem could be framed as predicting which users will interact with which topics in the future, taking advantage of the information from both sources[1].

We are interested in the following general problem:

**PROBLEM DEFINITION 5. Forecasting Tensor Evolution**

- **Given** two coupled tensors ($\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$), a number of K relations and S time-steps.
- **Forecast**, for the next S time-steps, the ranked list of the $K$ most likely non-zero elements of $\boldsymbol{\mathcal{X}}$, **maximizing** precision.

While Problem 5 is interesting by itself, accurate top-K predictions can often be made by identifying which non-zeros constantly appear in the tensor of interest. In the previous example, these would correspond to users that have constantly discussed the same topics over time. Therefore, we define the following related problem:

**SUBPROBLEM 1. Forecasting Novel Relations**

- **Given** two coupled tensors ($\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$), a number of K relations and S time-steps.
- **Forecast**, for the next S time-steps, the ranked list of the $K$ most likely **new** relations of $\boldsymbol{\mathcal{X}}$.

We define a **new** or **novel** relation as a non-zero that does not exist in the tensor of interest when the time component is collapsed. This is very related to a very well-known and important problem [NGK+06]:

**Observation 4.** *Predicting Churn*
Identifying **novel** relations is akin to identifying churn, i.e., finding which customers will first lose interest on a given service.

This highlights the importance of forecasting novel relations. We argue that Subproblem 1 is more useful in many realistic scenarios where predicting who is joining or leaving a community is more relevant than predicting who is staying. For instance, in the elections example, members who recently joined the discussion are probably easier to influence, while forecasting clients likely to stop doing business with a company is one of the key problems in customer relations.

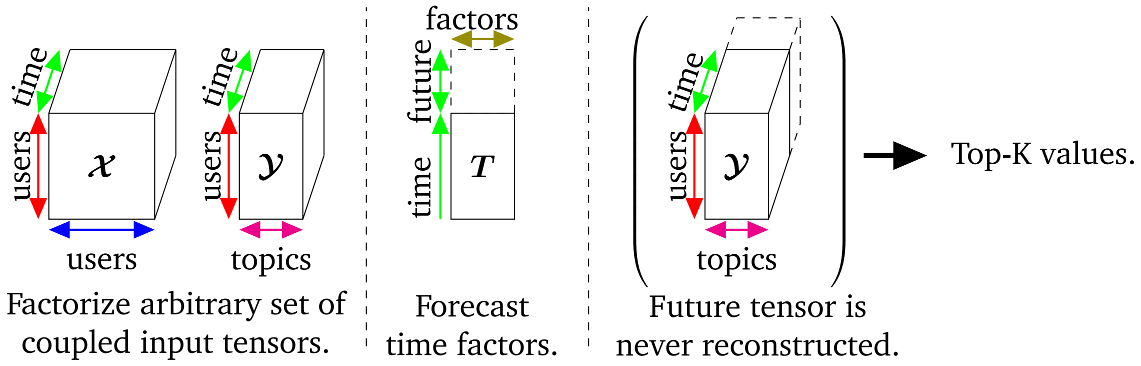
### 8.1.1 Overview.

TENSORCAST is comprised of three successive steps, described in more detail in the following subsections:

[1] One of the experiments in Section 8.2 deals with this scenario.

1. **Non-negative Coupled Factorization:** the factorization will tie together the various input tensors and identify their rank-1 components.
2. **Forecasting:** given the low dimensional space identified, we use standard techniques to forecast the time component.
3. **Top-K elements:** we exploit the factorization structure and identify the top elements without having to reconstruct the prohibitively big future tensor.

Figure 8.2 illustrates the intuition of our method.



Factorize arbitrary set of coupled input tensors.

Forecast time factors.

Future tensor is never reconstructed.

**Figure 8.2: Overview of TensorCast.**

### 8.1.2 Non-negative Coupled Factorization

Consider that the tensor of interest, $\boldsymbol{\mathcal{X}}$, is a 3-dimensional $N \times M \times T$ dataset and that the time component corresponds to the last index of the tensor. Then, naively, the number of elements to be forecasted ($S \times N \times M$) is a prohibitive number when we consider $\boldsymbol{\mathcal{X}}$ to be big and sparse.

Therefore, factorizing the input data achieves a two-fold objective: not only does it reduces the number of elements to be forecasted, but perhaps more importantly, it co-clusters similar elements together enabling generalization. A careful factorization will allow the forecast of previously unseen relations. We opted for a non-negative coupled factorization in order to improve the interpretability of the model; the importance of this feature will be clear when analyzing empirical evidence in Section 8.2.

We explore how user interactions can be leveraged to improve forecasts of future user-entity relations. Under this assumption, the problem is better modeled as two coupled tensors where tensor $\boldsymbol{\mathcal{Y}}$ is a $N \times N \times T$ symmetric tensor. In order to guarantee convergence, we modify the update of the symmetric factor matrix to

$$\boldsymbol{A} \leftarrow \boldsymbol{A} * \sqrt[3]{\frac{\boldsymbol{\mathcal{X}}_{(1)}(\boldsymbol{B} \odot \boldsymbol{T}) + \alpha \boldsymbol{\mathcal{Y}}_{(1)}(\boldsymbol{A} \odot \boldsymbol{T})}{\boldsymbol{A}(\boldsymbol{B} \odot \boldsymbol{T})'(\boldsymbol{B} \odot \boldsymbol{T}) + \alpha \boldsymbol{A}(\boldsymbol{A} \odot \boldsymbol{T})'(\boldsymbol{A} \odot \boldsymbol{T})}} \tag{8.1}$$

See Chapter 2 for further details.

### 8.1.3 Forecasting

Let $\boldsymbol{T}$ be the $T \times F$ factor matrix obtained from the previous step that corresponds to the time component. It consists of a small set of $F$ dense factor vectors, hence easy to forecast, that will provide an approximation $\hat{\boldsymbol{\mathcal{X}}}$ of the next time-step.

The most appropriate forecasting mechanism is data-dependent. We forecast using basic exponential smoothing (Holt's method), but other methods can be applied, e.g. Holt-Winters double exponential smoothing when seasonality is present.

### 8.1.4 Tensor Top-K elements

The forecast of the next time-step is a $N \times M \times S$ tensor represented as $\sum_f \boldsymbol{a_f} \circ \boldsymbol{b_f} \circ \boldsymbol{s_f}$ where $\boldsymbol{A}$ is $N \times F$, $\boldsymbol{B}$ is $M \times F$ and $\boldsymbol{S}$ is $S \times F$.

We extend the literature on the retrieval of maximum entries in a matrix product to the tensor case, leveraging the fact that the factorization was not performed on random data but on a graph that follows typical properties. The goal is to identify the K $(i, j, k)$ positions with highest value

$$\sum_f \boldsymbol{A}_{if} \boldsymbol{B}_{jf} \boldsymbol{S}_{kf}$$

We'll start by showing how this could be achieved if the $\hat{\boldsymbol{\mathcal{X}}}$ tensor was rank-1 and how multiple factors can be combined while preserving performance guarantees. We assume that the number of forecasted time-steps is significantly smaller than the number of users or topics (i.e., $S \ll N, M$) and that the number of topics is of the same order of magnitude but smaller than the number of users (i.e., $M < N$).

**Single factor Top-K.** We start by creating a data structure that lets us obtain the next biggest element in $O(\log(SM))$ time, with only $O(S \log S + M \log M + N \log N + SM)$ preprocessing.

Firstly, we sort the three vectors ($\boldsymbol{s}$, $\boldsymbol{a}$ and $\boldsymbol{b}$) in decreasing order. Note that, now, not only do we know that the biggest element is given by $\boldsymbol{a}_1\boldsymbol{b}_1\boldsymbol{s}_1$, but also that an element $\boldsymbol{a}_i\boldsymbol{b}_j\boldsymbol{s}_k$ only needs to be considered after $\boldsymbol{a}_{i-1}\boldsymbol{b}_j\boldsymbol{s}_k$, $\boldsymbol{a}_i\boldsymbol{b}_{j-1}\boldsymbol{s}_k$ and $\boldsymbol{a}_i\boldsymbol{b}_j\boldsymbol{s}_{k-1}$ have all been identified as one of the biggest K[2]. Hence, we can create a priority queue which only holds, at most, $O(SM)$ elements at a time.

**Combining multiple factors.** The major hurdle is handling the interaction between the multiple factors. We propose a greedy Top-K selection algorithm that, under realistic scenarios, efficiently achieves this goal. Algorithm 7 illustrates the pseudo code of this procedure.

We keep a list ($\boldsymbol{R}$) of the $K$ biggest positions evaluated so far and $\boldsymbol{F}_i.next$ represents the next element not yet considered in factor's $i$ priority queue, as described in the previous section. In each iteration, we consider the element with the highest score in one of the factors and add it to the list after evaluating it across all the factors. We terminate when the sum of the next best scores on each factor becomes smaller than the $K^{th}$ biggest element in $\boldsymbol{R}$.

```
input  : $\boldsymbol{F}$ - priority queues of factors
input  : $K$ - number of elements
output : $\boldsymbol{R}$ - set of biggest elements
while $\sum_i \boldsymbol{F}_i.next.factorScore \leq \boldsymbol{R}.last.Score$ do
    f $\leftarrow \arg\max_i(\boldsymbol{F}_i.next.factorScore)$
    element $\leftarrow \boldsymbol{F}_f.next$
    $\boldsymbol{F}_f.pop$
    $\boldsymbol{R} \leftarrow \boldsymbol{R} \cup element.fullScore$
    if $\boldsymbol{R}.size > K$ then
        $\boldsymbol{R} \leftarrow \boldsymbol{R} - \arg\min(\boldsymbol{R})$
    end
end
return $\boldsymbol{R}$
```

**Algorithm 7:** TensorCast: Top-K Elements

In the following, we prove the correctness and upper bounds on the overall number of elements that need to be evaluated.

**Theorem 1** *Algorithm 7 always returns the correct set of Top-k elements.*

***Proof 8*** *Consider an element $x$ that should be included in $\boldsymbol{R}$ but was never considered. As the algorithm has terminated, it follows that $x$'s score is lower than the sum of all the individual factor scores of elements at the top of each priority queue. However, we know that the smallest element in $\boldsymbol{R}$ is bigger than this, so this is a contradiction and $x$ cannot exist.* ■

[2] For instance, we know that the second biggest element is one of $\boldsymbol{a}_2\boldsymbol{b}_1\boldsymbol{s}_1$, $\boldsymbol{a}_1\boldsymbol{b}_2\boldsymbol{s}_1$ or $\boldsymbol{a}_1\boldsymbol{b}_1\boldsymbol{s}_2$.

Theorem 1 proves that Algorithm 7 always finds the correct set of elements. We now show that the set of elements that need to be considered is small when factors follow common power-law distributions. We assume $\boldsymbol{a}$ and $\boldsymbol{b}$ follow power-laws of the form $(\alpha - 1)x^{-\alpha}$ for $x \geq 1$ and $\alpha > 1$.

**Lemma 8** *If factor vectors* $\mathbf{a}$ *and* $\mathbf{b}$ *follow power-laws with exponents* $\alpha_a$ *and* $\alpha_b$*, then a randomly drawn element from any rank-1 frontal slice created as* $\boldsymbol{C_{kf}} = \mathbf{a} \circ \mathbf{b}$ *asymptotically follows a power-law*

$$p_C(z) = (\alpha - 1)z^{-\alpha}$$

*where* $\alpha = \min(\alpha_a, \alpha_b)$.

***Proof 9*** *Let* $X$ *and* $Y$ *follow power-law distributions of the form*

$$p_X(x) = (\alpha_a - 1)x^{-\alpha_a}$$
$$p_Y(y) = (\alpha_b - 1)y^{-\alpha_b}$$

*Then* $Z = XY$ *has probability distribution [GS01, p. 109]:*

$$p_Z(z) = \int_1^z p_X(w)p_Y\left(\frac{z}{w}\right)\frac{1}{w}dw =$$
$$= \frac{(\alpha_a - 1)(\alpha_b - 1)}{\alpha_a - \alpha_b}(z^{-\alpha_a} - z^{-\alpha_b})$$

*which tends to a power-law with exponent* $-\min(\alpha_a, \alpha_b)$. ■

**Theorem 2** *Algorithm 7 needs to check at most* $KSF^{1+\frac{1}{\alpha}}$ *elements if every frontal slice* $\mathbf{a_f} \circ \mathbf{b_f}$ *follows a power-law.*

***Proof 10*** *We'll consider the frontal slices one at a time and show that one only needs to check* $KF^{1+\frac{1}{\alpha}}$ *elements to find the* $K$ *biggest values of each slice. Let* $\alpha_{1..F}$ *be the exponents of the power-law distribution of each of the* $F$ *factor matrices* $\mathbf{a_f} \circ \mathbf{b_f}$ *of a given frontal slice and let* $\alpha_m = \min \boldsymbol{\alpha}$. *The K-th biggest element of* $\sum_f \mathbf{a_f} \circ \mathbf{b_f}$ *is at least* $K^{-\alpha_m}$*, as that is the* $K^{th}$ *biggest value of the slowest decreasing power-law*[3]*.*

*Given the iterative nature of Algorithm 7, we will prove an upper-bound for the maximum position (i.e., how deep in one of the factors) an element can be, while still having a reconstruction value greater than* $K^{-\alpha_m}$.

[3] Remember that $\boldsymbol{A}$ and $\boldsymbol{B}$ are non-negative matrices. In the worst-case, the score of the $K^{th}$ biggest element is taken from a single power-law and the contribution of the rest of the factors is 0, hence $K^{-\alpha_m}$ is a lower-bound for the $K^{th}$ biggest value.

*Let $x$ be the position of such element*[4]*, then*

$$K^{-\alpha_m} \leq \sum_f x^{-\alpha_f} \leq F x^{-\alpha_m} \implies x \leq K F^{\frac{1}{\alpha_m}}$$

*This means that any top-k element needs to be in a position smaller than $KF^{\frac{1}{\alpha_m}}$ in at least one of the factors, which implies that, in the worst case, Algorithm 7 only needs to check $KF^{\frac{1}{\alpha_m}}F = KF^{1+\frac{1}{\alpha_m}}$ elements to find the $K$ biggest elements on each frontal slice. Therefore, we can upper-bound the total number of elements checked by $KSF^{1+\frac{1}{\alpha}}$.* ■

Note that TensorCast is linear on the number of elements we want to obtain times the number of time-steps forecasted. Furthermore, note that this result agrees with intuition: sharper (i.e., quickly decreasing, higher exponent) power-laws require less elements to be checked, while near-clique factors imply lower exponents and more elements to be analyzed.

Figure 8.3 provides further empirical evidence of the linear growth on the number of values we need to check. We plot the number of positions evaluated as $K$ is increased, on a synthetic network, when forecasting one time-step ($S = 1$), using 8 factors and varying the power-law exponents from $1.5$ to $2.2$.

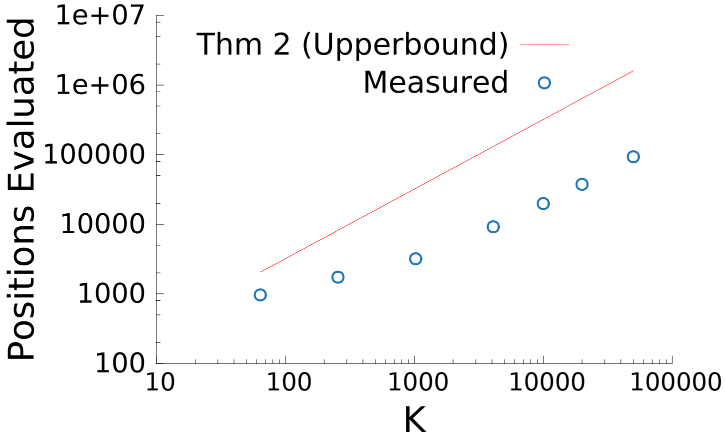


**Figure 8.3: TensorCast only checks a linear number of elements of the tensor.**

### 8.1.5 Complexity Analysis.

**Observation 5.**
TensorCast requires time linearithmic on the number of non-zeros of its input tensors.

[4] In the worst case scenario, this element is at position $x$ in every of the factors.

**Table 8.1: Summary of real-world networks used.**

| Users | Groups | Timesteps | Memberships | Interactions | Description |
|---:|---:|---:|---:|---:|---|
| 1 734 902 | 5 476 | 79 | 8 049 559 | 21 423 244 | DBLP - venues and co-authorships. |
| 12 426 133 | 2 326 843 | 31 | 30 281 817 | 282 280 158 | TWITTER - hashtags and retweets. |

*Rationale.* TENSORCAST's time complexity is a sum of its three stages:

1. The coupled-factorization requires linearithmic time on the number of non-zeros.
2. Forecasting is typically linear on the number of timesteps, although it depends on the algorithm selected.
3. As shown in the previous section, identifying the top-$K$ elements is linear on $K$ and sub-quadratic on the number of factors.

## 8.2 Experiments

We report experiments to answer the following questions:

Q1. **Scalability**: How fast is TENSORCAST?

Q2. **Effectiveness:** and **Context-awareness**: How does TENSORCAST's precision compare with its alternatives? How much improvement does contextual data bring?

Q3. **Trend Following**: How capable is TENSORCAST of detecting and following trends?

Q4. **Precision over Time**: How does TENSORCAST's precision decrease as we forecast farther to the future?

TENSORCAST is tested on two big datasets detailed in Table 8.1. In the DBLP dataset, the tensor to be forecasted consists of authors and venues in which they published from 1970 to 2014, while the co-authorship tensor is used as contextual information. Evaluation is performed on the 2015 $author \times venue$ data. In the TWITTER dataset, the tensor of interest relates users and hashtags (#) they used from June to December 2009, while the auxiliary tensor represents user interactions through re-tweets. Tweets are grouped by week and evaluation is performed on week 51.

Unless otherwise specified, every factorization approach uses 10 factors. On the TWITTER dataset, we weighted the reconstruction of the tensor of interest as 20 times more relevant that the context tensor. On DBLP, we weighted non-zeros of the tensor of interest 2.66 times higher than in the tensor of interest (so that both tensors have the same reconstruction error when considering empty factors).

### 8.2.1 Q1 - Scalability

We start by evaluating our method's scalability when changing the number of non-zeros in the TWITTER dataset[5]. By changing the number of weeks under consideration, we create a sequence of pairs of tensors that increase in size. For each pair, we measure wall-clock time when performing a rank-4 coupled tensor factorization, forecasting and identification of the top-1000 forecasted non-zeros. Figure 8.1b shows TENSORCAST's linear scalability.

### 8.2.2 Q2 - Effectiveness and Context-awareness

Figure 8.1a and Figure 8.4 showcase TENSORCAST's accuracy on the task of predicting relations on future time steps. While Figure 8.1a shows TENSORCAST's superior precision as we increase $K$ on the DBLP dataset, Figure 8.4 focus particularly on forecasting **novel** relations on TWITTER. We would like to highlight the difficulty of this task, as we are predicting whether a given user is going to start using a new hashtag on the next week. Nevertheless, TENSORCAST achieves double the precision of competing methods[6].

Furthermore, note the importance of TENSORCAST's ability of being simultaneously contextual and time-aware, as the precision of the current state-of-the-art is limited due to ignoring either one of these aspects.

The competing *CP Forecasting* [DKA11] method was run using Holt forecasting, given the lack of seasonality of the data. The results of the other competitor, *Coupled Matrices*, were obtained by finding non-negative factors that minimize the reconstruction error of the collapsed tensors, weighted for the same importance. For fairness, all appropriate methods use 10 as the number of factors.

### 8.2.3 Q3 - Trend Following

We evaluate TENSORCAST's ability of predicting an increase or decrease in the activity around a given topic or between a group of users over time. We created a synthetic dataset with 5 "hyperbolic" communities (i.e., with power-law internal degree distribution) of 100 users over 11 days (10 days are used for the factorization and 1 for evaluation). The

[5] We consider the sum of the non-zeros of both tensors.

[6] Note that the quality of absolute precision numbers is affected by 1) how imbalanced the two classes are and 2) the cost of false positives. An improvement from 2% to 5% precision might imply that 1 out of 20 phone-calls we make target a potential customer versus every 1 in 50.

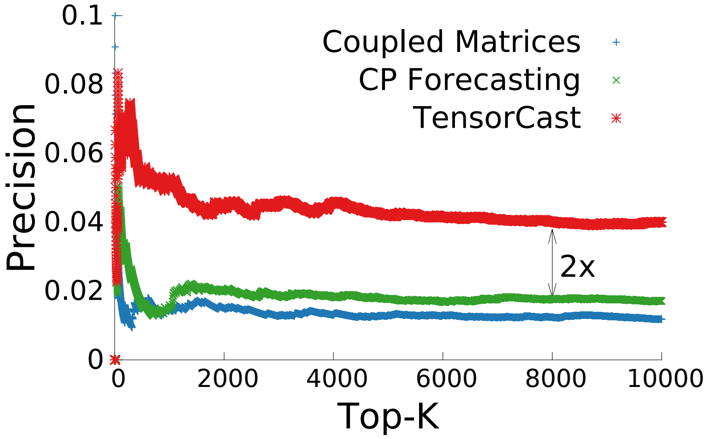


**Figure 8.4: Double precision when forecasting novel $(user, hashtag)$ relations in the TWITTER tensor.**

average density over the first 10 days equals 15% for all communities, but their density changes differently over time: two communities have their densities increasing at 1% and 2% per day, one has constant density and the other have their density decreasing by 1% and 2% per day.

Figure 8.5 shows the scores of the 5 columns of the $\boldsymbol{T}$ matrix after factorization, one per line. We can see that linear changes in density correspond to linear changes of the scores and that TENSORCAST correctly forecasts a similar change in the future.

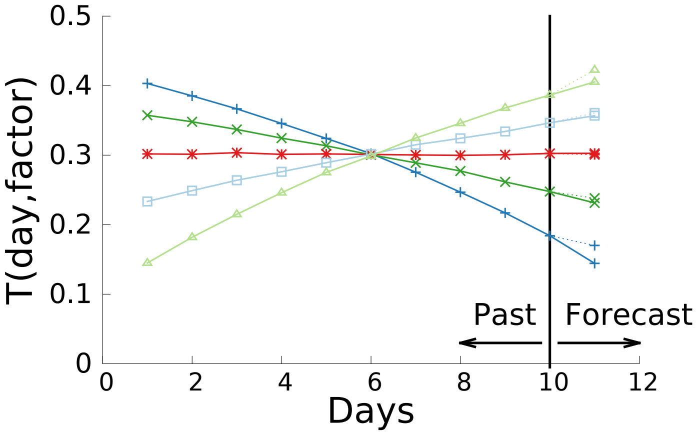


**Figure 8.5: TENSORCAST correctly forecasts growth and decay of groups in synthetic data. [dashed - forecast; solid - real]**

### 8.2.4 Q4 - Precision over Time

We evaluate TensorCast's precision as the forecasting horizon is increased. We use the DBLP dataset, doing five runs with each method when considering different "training" periods (i.e., the first run considered every publication before 2010, while the last run considered every publication before 2015). For each run, we obtained the 1000 most likely non-zeros for each of the next 5 years and calculated the precision of the method, when applicable. Figure 8.6 shows, for each method, the average precision for each forecasting horizon.

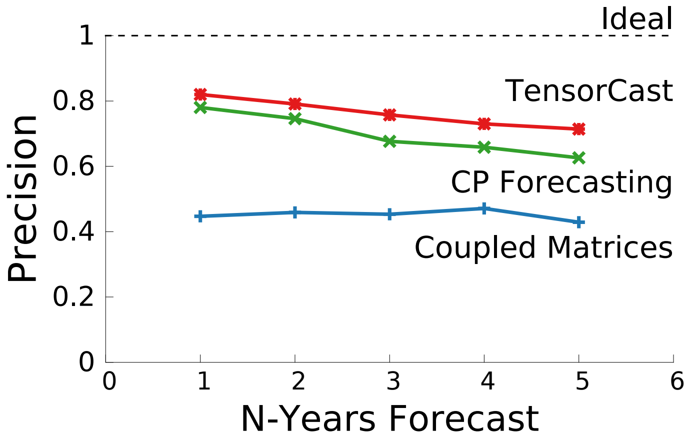
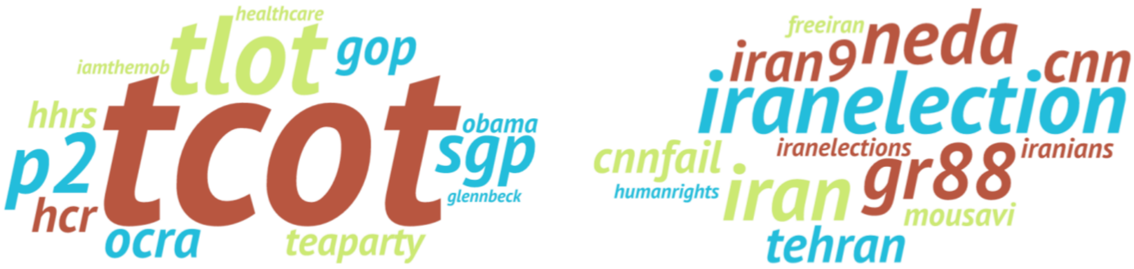


**Figure 8.6: TensorCast achieves higher precision at every forecasting horizon.**

### 8.2.5 Discoveries - TensorCast at work

In addition to the forecast of new relations, groups found by TensorCast are interpretable due to the non-negativeness of the factors. We highlight two groups we identified on the Twitter dataset with their most used hashtags (#) on Figure 8.7 (word size corresponds to importance on factor). The first group corresponds to a group of users who typically use hashtags that show a conservative political orientation: references to the tea party and critics of the healthcare reform. Users in the second group use hashtags related to the Iranian election and human-rights protests, such as *#iranelection* or *#neda*, the name of a student who was killed during the protests.

Figure 8.8 shows TensorCast's ability to predict user interactions based on current interest on a topic. Note that, on the Iranian group, the factorization highlights the week of the elections and the protests (in June), but interest clearly fades in the second-half of the year. On the other hand, we can see that political tags are still used by the same group of users for several months.



**(a) Political group. Hashtags related to "top conservatives on Twitter" (*#tcot*) and, respectively, "liberals" (*#tlot*), *#obama*, *#healthcare* (*#hcr*), "smart girl politics" (*#sgp*), etc..**

**(b) Iranian elections group. Hashtags related to the Iranian elections and human rights protests.**

**Figure 8.7: TensorCast finds and forecasts groups with similar interests on Twitter.**

**Implementation details and Reproducibility.** Similarly to its logical steps, TensorCast was implemented as three different modules run in succession:

1. The non-negative coupled factorization was implemented on Matlab using its Tensor Toolbox [BKo15].
2. Forecasts were done using Gnu's Regression, Econometrics and Time-series Library (GNU gretl) [BD03].
3. Tensor top-K elements' algorithm was implemented in Scala as a stand-alone tool.

TensorCast can be obtained at `http://www.cs.cmu.edu/~maraujo/tensorcast`.

## 8.3 Related Work

### 8.3.1 Top-K elements in Matrix Products

Given the widespread applications of matrix factorizations, finding the top-K elements of a matrix product is an important problem with several use cases, from personalized user recommendations to document retrieval.

The problem can be stated as, given matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of sizes $N \times F$ and $M \times F$, respectively, find the top K $(i, j)$ pairs of the $\boldsymbol{AB^T}$ matrix product. Note that the *naive* solution requires $O(NMF)$ operations, iterating over the (originally) implicitly defined reconstruction matrix. Some attention has been given to this problem, since Ram and Gray [RG12] proposed the use of Cone Trees to speed-up this search. Other approaches map this problem into smaller sets of cosine-similarity searches [TGM15], a related

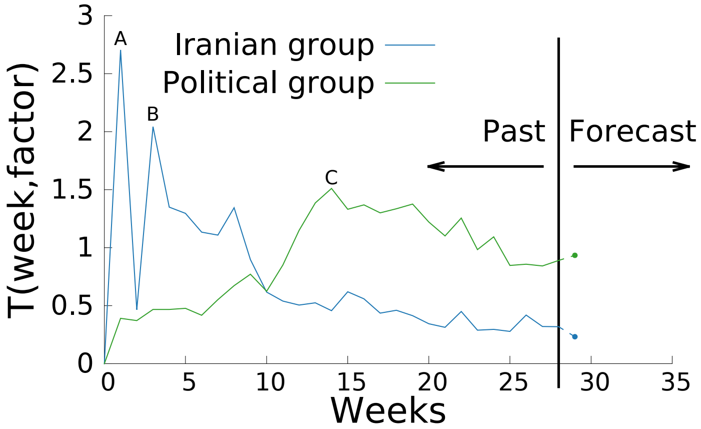


**Figure 8.8: Increased precision is achieved by grouping interactions. [A-Start of 2009 election protests; B-President Obama references Neda Soltan, killed in the protests; C-Taxpayer March in Washington]**

but easier problem given the unit-length of the vectors. Approximate methods have also been tried, such as transforming the problem in a near-neighbor search and using locality sensitive hashing (LSH) [SL14, NS14]. However, this is a non-convex optimization problem in general.

### 8.3.2 Power-laws as building blocks

When factorizing real-life graph data, the scores of the non-negative factors are not uniformly distributed but decrease sharply. We've shown in Chapter 3 that the internal degree distribution of big communities can be well approximated by a power-law across several domains. It has also been shown that eigenvectors of Kronecker graphs exhibit a multinomial distribution [LCKF05, theorem 3]. TensorCast leverages this property in order to speed-up its computation of Top-K elements without reconstructing the forecasted *tensor*.

### 8.3.3 Link Prediction

A large body of literature on link prediction has been created since its introduction [LNK07]. In *structural* link prediction, the original problem, the goal is to predict which links are more likely to appear in the future given a current snapshot of the network under analysis. This setting, where it is typical to assume that links are never or seldom removed, has found multiple applications in predicting interactions in protein-protein networks, social networks (e.g., friendship relations) and recommendation problems. The Netflix challenge sprung the creation of several latent factor models with differing structure and/or regularization terms for this task [Kor08, ME11], but there were also several approaches which showed that using the age of the link could lead to improved predictions [Kor10].

On the other hand, given the increased availability of dynamic or time-evolving graphs (frequently used to model evolving relationships between entities over time), *temporal* link prediction methods have been developed to predict future snapshots. In this setting where links are not guaranteed to persist over time, we distinguish methods that rely on collapsing (matricizing) the input data (e.g., exponential decay of edge weights [SN08, GDG11]) from methods that deal directly with the increased dimensionality, such as tensor-based methods. CP Forecasting [DKA11] finds a low-rank PARAFAC factorization and forecasts the time-component in order to incorporate seasonality. TriMine [MSF+12] similarly factorizes the input tensor, but then applies probabilistic inference in order to identify hidden topics that connect users and entities, which it then draws from in order to generate realistic sequences of future events. These methods are not able to integrate contextual information on their predictions. Other approaches integrate structure and content in the same prediction task, e.g. Gao et al [GDG11] suggest a coupled matrix factorizations and graph regularization technique to obtain the latent factors after an exponential decay of the temporal network.

However, none of these methods fulfills all the requirements for forecasting when contextual information is considered. Table 8.2 contrasts TensorCast against the state of the art competitors on key specs:

a) linear **scalability** with sparse data;

b) **interpretability** of the underlying model;

c) **time-awareness** for forecasting periodic, growing and/or decaying relations;

d) ability to deal with additional **contextual information**;

e) the ability to **forecast** the disappearance of existing relations;

f) the ability of providing an **ordered** ranking of future events by likelihood of occurrence.

## 8.4 Summary

We presented TensorCast, a method which addresses the forecasting problem on big time-evolving datasets when contextual information is available. We leverage typical graph properties in order to create a linearithmic algorithm that can find novel relations

|  | **TensorCast** | *Truncated SVD* | *Truncated Katz* | *Coupled Matrices (e.g., [GDG11])* | *VAR[Ze62], ARIMA[BP70], etc.* | *CP Forecasting [DKA11]* | *TriMine [MSF+12]* |
|---|---|---|---|---|---|---|---|
| **Scalability** | ✔ | ✓ | ✓ | ✓ |  | ✓ | ✓ |
| **Interpretability** | ✔ | ✓ | ✓ | ✓ |  | ✓ | ✓ |
| **Time-awareness** | ✔ |  |  |  | ✓ | ✓ | ✓ |
| **Context-awareness** | ✔ |  |  | ✓ |  |  |  |
| **Forecasting** | ✔ |  |  |  | ✓ | ✓ | ✓ |
| **Ordered Forecasting** | ✔ |  |  |  | ✓ | ✓ |  |

**Table 8.2: TensorCast integrates context and time-awareness.**

in very big datasets efficiently.

The main advantages of our method are:

1. **Generality**: by using generalized tensors, TensorCast can be applied to a multitude of data sources with different structure, including coupled tensors and coupled matrix-tensor scenarios.
2. **Scalability** : TensorCast scales linearithmically with the input size and is tested in datasets with over *three hundred million* non-zeros.
3. **Effectiveness**: TensorCast achieves 20% higher precision in top-1000 queries than comparable alternatives.
4. **Context-awareness**: we show how different data sources can be included in a principled way.

# Part III

# Conclusions and Future Directions

# Chapter 9

# Conclusions

In this dissertation, we study common patterns and groups in multiple networks: phone calls, flights, co-authorships, “purchased together” items or movie ratings are just a few examples of real-world processes that have achieved or might grow to *billions* of relations. Given their scale and ubiquity, crucial issues arise when trying to interpret or distill insights from these processes.

We focus on algorithms and models that effectively and scalably analyze *simple* or *labeled* networks, in two related areas:

- **Communities and Pattern Detection:** We aim to understand the structure of groups of users in large networks, according to commonly used community definitions, and to understand their evolution across time or other edge labels.
- **Anomalies and Novel Relations Discovery:** The goal is to discover anomalous nodes or connections. To that end, at the node level, we study the detection of Points-of-Compromise in bank transactions and, at the relation level, we forecast group evolution and the creation of never-before-seen connections.

As networks grow bigger, we explore the most appropriate representations for their understanding, the most fitting patterns and building blocks, and the best methods to detect anomalous behavior or structure. In order to create efficient near linear time algorithms, we explore and leverage known properties of real-world networks and subgraphs. By exploring edge labels and other contextual information, our methods can be applied to most graph analytics settings.

## 9.1 Networks and Matrices

As hinted in the Introduction, Part I is focused on specific questions regarding the community structure and specific anomalies of unlabeled networks. Solutions to these questions leverage linear algebra, graph theory, information theory and machine learning concepts.

- **"What is the structure of real-world communities?":** Through the analysis of externally provided communities, we developed the *Hyperbolic Community Model*, a model that better represents meaningful subgraphs in big networks: we showed that edges are not uniformly distributed, but rather skewed - patterns that previously were only attributed to the full network. We showed that big communities fit power-law degree distributions fairly well and side-by-side comparison of the adjacency matrix of the theoretical and real communities shows their similarity.
- **"How can we find them?":** We contribute two distinct approaches for finding realistic communities. HYCOM-FIT uses the Minimum Description Length principle to find communities that are modeled as having a power-law degree distribution. We also contribute a fast algorithm, FASTSTEP, which uses a novel binary matrix factorization approach that is able to obtain non-block but binary reconstructions. Applications in a recommendation systems setting and in clustering nodes of a flights' network illustrate FASTSTEP's power.
- **"Can we detect sources of data breaches or infections in billion-sized datasets?"** We introduce BREACHRADAR, a method that is able to detect anomalous nodes whose erroneous behavior is only detected in some of their neighbors. BREACHRADAR's key idea is the alternating optimization of two conflicting aspects: while infected nodes want to increase the blame of their neighbors, they know that they were likely infected by a single source. Therefore, they converge to an agreement after a few rounds of message passing that only requires local information stored in their neighbors. This approach leads to an easy to parallelize probabilistic graphical model that is used in *billion-sized* transaction networks to detect real Points-of-Compromise. Application to real transaction networks shows fraud prevention of over *one million US dollars* per month.

## 9.2 Labeled Networks and Tensors

Part II focused on labeled networks, with significant importance given to temporal networks. We propose algorithmic solutions that rely on tensor algebra, information theory and graph theory, in order to tackle two important questions.

- **"What are typical patterns of temporal communities? How can we find them?":** We contribute COM$^2$, a method that combines tensor algebra and information theory in order to find a succinct description of the labeled network. Similarly to HYCOM-FIT, COM$^2$ also leverages the Minimum Description Length principle in order to find relevant sets of nodes and labels that represent interesting concepts. When labels represent timesteps, we analyzed a phonecalls and a computer communications datasets and found that typical groups are not constant but rather tend to appear over non-consecutive timesteps.
- **"Can we forecast novel relationships when context is available?":** We contribute TENSORCAST, a three stage method that is able to incorporate contextual information in the forecast of future edges in time-evolving networks. The key idea behind TENSORCAST is a three steps process: a coupled tensor factorization is used to reduce the dimensionality of the problem, a time series forecasting method extends the time component, and a novel tensor top-k method is able to efficiently obtain the most likely future connections avoiding the quadratic explosion problem. We report double precision in identifying novel relations when compared to state-of-the-art approaches.

## 9.3 Overall Impact

This work contributes in the cross product of two common themes: on the one hand, we focus on patterns and anomalies, while on the other, we study both static and labeled networks, usually temporal. Hence, it has broad impact on several domains: recommendation systems, fraud detection, contextual forecasting and graph understanding, and has been used in multiple settings:

- **Impact in industry:** Feedzai detects data breaches and other Points-of-Compromise using BREACHRADAR [AAF$^+$17], a work who has a patent submitted.
- **Impact in academia:** At the University of Lisbon, FASTSTEP is the basis of a Masters thesis dissertation.
- **Awards:** COM$^2$ [APG$^+$14] received one of the best paper awards at PAKDD'14.

**Reproducibility.** HyCoM-Fit, FastStep, Com$^2$ and TensorCast are made available for the research community.

# Chapter 10

# Vision and Future Directions

This research was focused on the development of scalable algorithms for finding patterns and anomalies in static and labeled networks. Some of the research directions we envision as extending our work are outlined below.

## 10.1 Systems: "Database Factorizations"

Finding patterns, anomalies, recommendations and forecasting while integrating context are some of the most relevant current problems. In Chapter 8, we explored how coupled tensors could be leveraged to improve forecasts using context cues. However, collecting the data and deciding on the specific decomposition to be applied is cumbersome and requires both domain and technical knowledge. Fortunately, most of these domain relations have already been systematized: they are encoded as primary key relations in most relational database management systems (RDBMS). Can we leverage this information to lift the technical burden from the end-users and to create systems that are able to answer domain-specific questions easily? Can we forecast sales growth by automatically taking into consideration the relations between quarters and orders and between orders and customers and products? Can fraudsters be identified from telecommunication records?

Not only is RDBMS data easy to access and already connected, one could perform most operations directly with SQL, leveraging RDBMS'es key features: on-disk and distributed capabilities, transactions, checkpointing and recovery, integrity constraints, etc. . Other relevant methods that target single SQL tables have already been implemented in SQL, such as PCA [NO09] and K-step neighbors, cores and induced subgraphs [KTS+11]. Other systems such as Vertexica [JRW+14] translate vertex-centric graph operations into SQL. It is only natural to extend these graph analytics capabilities to the complete relational database.

## 10.2 Theory: Adversarial Anomalies

We've been looking at groups (communities) and anomalies (fraudsters, strange groups) in networks. Most examples and success stories involve malicious intent - points-of-compromise, fake reviews, etc. What guarantees do factorization methods provide against these adversarial attacks? Assuming typical network characteristics, can we identify bounds regarding the size and density of the maximal near-clique that won't be reported in a factorization? What is the relevance of camouflage techniques typically used (i.e., fraudsters will follow celebrities in addition to customers)? Answering these questions requires incorporating the known properties of large scale networks and the typical fraud patterns into the theoretical bounds of spectral theory.

# Appendices

# Appendix A

# BreachRadar - Additional details

## A.1 Fraud Label Delays

The availability of fraud labels is an obvious requirement of the procedure described in this work. Without fraud labels, or with a significant delay between the transaction date and the date in which the fraud label was added, our ability to minimize eventual losses is severely restricted. Therefore, we present a preliminary analysis on whether these labels are available in a reasonable time frame.

Figure A.1 shows the distribution that relates the number of fraudulent transactions and the number of days the respective label was assigned. We can see that the number of days that a typical label is delayed is very small; while it might be initially surprising, the reality is that several systems try to detect whether a transaction is fraudulent in real-time. Financial institutions then either abort the transaction or call the card owner in order to validate the purchase.

Furthermore, note that once a card has been cloned and is successfully used in a fraudulent transaction, fraudsters will typically use that opportunity to spend all the available funds as fast as possible, before the rightful owners are able to communicate the unauthorized transactions to their bank. As a consequence, with a nearly empty or even over-drafted account, customers are quick to report these transactions to their banking institution.

## A.2 Multiple Points-of-Compromise

While assuming that *fraud-cards* were stolen at a single *possible Point-of-Compromise* agrees with intuition (a rare event shouldn't affect the same card multiple times), we realized

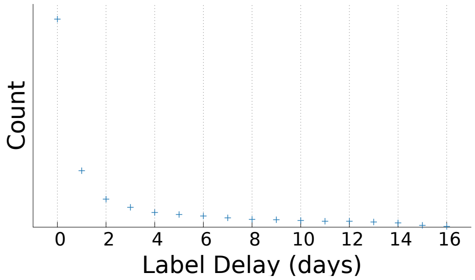


**Figure A.1: Most labels are available shortly after the transaction. (actual counts removed for privacy)**

that this assumption does not hold when these possible POCs have a fine granularity, such as "terminal-week" pairs. Database breaches often imply that multiple transactions (hence possibly multiple buckets) in the same merchant might be Points-of-Compromise, so Equation 5.2 needs to be modified for the algorithm to work in this scenario. In order to do so, we replace the l1 normalization at the POC level with a normalization at the merchant level. The equation is updated as follows:

$$\boldsymbol{B}_{ij} = \begin{cases} \dfrac{\boldsymbol{\theta}_j}{\sum\limits_{k \in \boldsymbol{N}_i \text{ and } m(j) \neq m(k)} \boldsymbol{\theta}_k} & \text{if } f_i = 1 \text{ and } j \in \mathbf{N_i} \\ 0 & \text{otherwise} \end{cases} \tag{A.1}$$

where $m(j)$ is the merchant associated with bucket $j$. In scenarios with such bucket definitions, the `UpdateBlames` function in Algorithm 3 is also changed accordingly.

$$\boldsymbol{B}_{ij} = \frac{\boldsymbol{p}_j}{\boldsymbol{p}_j + \sum\limits_{k \in \mathbf{N_i} \text{ and merch(j)} \neq \text{merch(k)}} \boldsymbol{p}_k} \tag{A.2}$$

# Bibliography

[AABB$^+$07] Evrim Acar, Canan Aykut-Bingol, Haluk Bingol, Rasmus Bro, and Bülent Yener. Multiway analysis of epilepsy tensors. In *ISMB/ECCB Supplement of Bioinformatics*, pages 10–18, 2007. 16

[AAF$^+$17] Miguel Araujo, Miguel Almeida, Jaime Ferreira, Luis Silva, and Pedro Bizarro. Breachradar: Automatic detection of points-of-compromise. In *Proceedings of the 17th SIAM International Conference on Data Mining (SDM)*, Houston, USA, 2017. 4, 7, 67, 147

[ADK11] Evrim Acar, Daniel M Dunlavy, and Tamara G Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics*, 25(2):67–86, Jan 2011. 125

[AGMF14] Miguel Araujo, Stephan Günnemann, Gonzalo Mateos, and Christos Faloutsos. Beyond blocks: Hyperbolic community detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pages 50–65. Springer Berlin Heidelberg, 2014. 3, 33

[AGP$^+$15] Miguel Araujo, Stephan Günnemann, Spiros Papadimitriou, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. Discovery of "comet" communities in temporal and labeled graphs Com$^2$. *Knowledge and Information Systems*, pages 1–21, 2015. 5, 93

[Aka74] Hirotugu Akaike. A new look at the statistical model identification. *Transactions on Automatic Control*, 19(6):716–723, 1974. 29, 62

[APF$^+$06] Balázs Adamcsek, Gergely Palla, Illés J. Farkas, Imre Derényi, and Tamás Vicsek. Cfinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, 2006. 46

[APG$^+$14] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E. Papalexakis, and Danai Koutra. Com2: Fast automatic discovery of temporal ('comet') communities. In *Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 271–283, Tainan, Taiwan, 2014. 5, 7, 93, 147

[ARF16] Miguel Araujo, Pedro Ribeiro, and Christos Faloutsos. Faststep: Scalable boolean matrix decomposition. In *Proceedings of the 20th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Auckland, New Zealand, 2016. 3, 51

[AS14] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Survey*, 47(1):10:1–10:36, May 2014. 90

[ATMF12] Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. Pics: Parameter-free identification of cohesive subgroups in large attributed graphs. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM)*, pages 439–450. SIAM, 2012. 26

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. 11

[Bal13] Brandon Ballenger. Rakuten.com customers reporting credit card fraud. finance.yahoo.com, June 2013. 77

[BBH$^+$14] John Bagnall, David Bounie, Kim P. Huynh, Anneke Kosse, Tobias Schmidt, Scott Schuh, and Helmut Stix. Consumer cash usage: A cross-country comparison with payment diary survey data. ECB Working Paper Series, 2014. 67

[BD03] Giovanni Baiocchi and Walter Distaso. Gretl: Econometric software for the gnu generation. *Journal of Applied Econometrics*, 18(1):105–110, 2003. 139

[BFS15] Kim P. Huynh Ben Fung and Gerald Stuber. The use of cash in canada. Bank of Canada review, 2015. 67

[BGHS12] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. Mining coherent subgraphs in multi-layer graphs with edge labels. In *Proceedings of the 18th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1258–1266. ACM, 2012. 90

[BGHS13] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. RMiCS: a robust approach for mining coherent subgraphs in edge-labeled multi-layer graphs. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 1–23, 2013. 90

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008. 30

[BH02a] James C Bezdek and Richard J Hathaway. Some notes on alternating optimization. In *Advances in Soft Computing—AFSS 2002*, pages 288–300. Springer, 2002. 75

[BH02b] Richard J Bolton and David J Hand. Statistical fraud detection: A review. *Statistical science*, pages 235–249, 2002. 83

[BK07] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explorations Newsletter*, 9(2):75–79, 2007. 51

[BKo15] Brett W Bader, Tamara G Kolda, and others. MATLAB Tensor Toolbox Version 2.6. Feb 2015. 119, 120, 125, 139

[BP70] George EP Box and David A Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association*, 65(332):1509–1526, 1970. 142

[Bro97] R Bro. PARAFAC. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 38(2):149–171, 1997. 125

[BTK$^+$14] Alex Beutel, Partha Pratim Talukdar, Abhimanu Kumar, Christos Faloutsos, Evangelos E Papalexakis, and Eric P Xing. Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In *Proceedings of the 14th SIAM International Conference on Data Mining (SDM)*, pages 109–117. SIAM, 2014. 21

[Cha04] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pages 112–124. Springer, 2004. 30

[Cor15] Fair Isaac Corporation. How europe's card fraud is evolving. Insights White Paper, 2015. 67

[CPMF04] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, and Christos Faloutsos. Fully automatic cross-associations. In *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (KDD)*, Seattle, United States, 2004. ACM. 30, 46

[CSN09] A. Clauset, C. Shalizi, and M. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009. 43

[CWZW11] Bingjing Cai, Haiying Wang, Huiru Zheng, and Hui Wang. An improved random walk based clustering algorithm for community detection in complex networks. In *International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2162–2167. IEEE, 2011. 29

[DA05] J Duch and A Arenas. Community detection in complex networks using Extremal Optimization. *arXiv.org*, (2):027104, Jan 2005. 28

[DBF05] Patrick Doreian, Vladimir Batagelj, and Anuska Ferligoj. *Generalized blockmodeling*, volume 25. Cambridge university press, 2005. 29

[DH73] William E Donath and Alan J Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973. 29

[Dhi01] Inderjit S Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 269–274. ACM, 2001. 29

[DKA11] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorization. *Transactions on Knowledge Discovery from Data (TKDD)*, 2011. 90, 136, 141, 142

[DLDMV00] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. On the best rank-1 and rank-(r1, r2,..., rn) approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342, 2000. 20

[DMM03] Inderjit S Dhillon, Subramanyam Mallela, and Dharmendra S Modha. Information-theoretic co-clustering. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 89–98. ACM, 2003. 51

[DPCLB$^+$14] Andrea Dal Pozzolo, Olivier Caelen, Yann-Ael Le Borgne, Serge Waterschoot, and Gianluca Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert systems with applications*, 41(10):4915–4928, 2014. 83

[Dun98] Robin Dunbar. *Grooming, Gossip, and the Evolution of Language*. Harvard University Press, 1998. 31

[ECA13] Beyza Ermiş, A Taylan Cemgil, and Evrim Acar. Generalized coupled symmetric tensor factorization for link prediction. In *Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2013. 21

[Eur15] European Central Bank. Fourth report on card fraud. Technical report, European Central Bank, 2015. 78

[EY36] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936. 15

[FC07] Santo Fortunato and Claudio Castellano. Community Structure in Graphs. *arXiv.org*, Dec 2007. 28

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *Proceedings of the Conference of the Special Interest Group on Data Communication (SIGCOMM)*, pages 251–262, 1999. 11, 36

[FH16] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016. 25

[FLG00] G.W. Flake, S. Lawrence, and C.L. Giles. Efficient identification of web communities. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 150–160. ACM, 2000. 1, 25

[For05] George Forman. Determining point-of-compromise. US Patent US 20050055373 A1, March 2005. 84

[For10] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75 – 174, 2010. 25, 28, 51

[GDG11] Sheng Gao, Ludovic Denoyer, and Patrick Gallinari. Temporal link prediction by integrating content and structure information. In *Proceedings of the 20th International Conference on Information and Knowledge Management*, pages 1169–1174. ACM, 2011. 141, 142

[GFBS14] Stephan Günnemann, Ines Färber, Brigitte Boden, and Thomas Seidl. Gamer: a synthesis of subspace clustering and dense subgraph mining. *Knowledge and Information Systems (KAIS)*, 40(2):243–278, 2014. 29, 30

[GFRS13] Stephan Günnemann, Ines Färber, Sebastian Raubach, and Thomas Seidl. Spectral subspace clustering for graphs with feature vectors. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 231–240, Dallas, United States, 2013. IEEE. 29

[GK65] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, 2(2):205–224, 1965. 15

[GMS04] Aristides Gionis, Heikki Mannila, and Jouni K. Seppänen. Geometric and combinatorial tiles in 0-1 data. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 173–184, Pisa, Italy, 2004. 29

[GMZ03] Christos Gkantsidis, Milena Mihail, and Ellen W. Zegura. Spectral analysis of internet topologies. In *Proceedings of the 22nd joint Conference of the Computer and Communications Societies (INFOCOM)*, pages 364–374, San Francisco, United States, 2003. IEEE. 29, 30

[Goo61] Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961. 11

[Gre10] Steve Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10):103018, 2010. 28

[Grü07] Peter D Grünwald. *The minimum description length principle*. The MIT Press, 2007. 38, 62, 95

[GS01] Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford university press, 2001. 133

[GZC$^+$16] Amir Ghasemian, Pan Zhang, Aaron Clauset, Cristopher Moore, and Leto Peel. Detectability thresholds and optimal algorithms for community structure in dynamic networks. *Physical Review X*, 6(3):031005, 2016. 91

[HAF16] San-Chuan Hung, Miguel Araujo, and Christos Faloutsos. Distributed community detection on edge-labeled graphs using spark. In *12th International Workshop on Mining and Learning with Graphs (MLG)*, San Francisco, USA, 2016. 113

[Har70] R.A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970. 18, 111, 114, 119

[HPK15] Samantha Hansen, Todd Plantenga, and Tamara G Kolda. Newton-based optimization for Kullback-Leibler nonnegative tensor factorizations. *Optimization Methods and Software*, 30(5):1002–1029, 2015. 125

[HXZ$^+$11] Zhaoshui He, Shengli Xie, Rafal Zdunek, Guoxu Zhou, and Andrzej Cichocki. Symmetric nonnegative matrix factorization: Algorithms and applications to probabilistic clustering. *Transactions on Neural Networks*, 22(12):2117–2131, 2011. 21

[jet11] Restaurant depot/jetro cash & carry customers' credit cards hacked.

DataBreaches.net, December 2011. 77

[JKC+04] D. S. Johnson, S. Krishnan, J. Chhugani, S. Kumar, and S. Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In *Very Large Data Bases (VLDB)*, 2004. 40, 97

[JPKF15] Inah Jeon, Evangelos E Papalexakis, U Kang, and Christos Faloutsos. HaTen2: Billion-scale tensor decompositions. *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1047–1058, 2015. 120, 125

[JRW+14] Alekh Jindal, Praynaa Rawlani, Eugene Wu, Samuel Madden, Amol Deshpande, and Mike Stonebraker. Vertexica: your relational friend for graph analytics! *Proceedings of the VLDB Endowment*, 7(13):1669–1672, 2014. 149

[Kar72] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972. 85

[KB09] T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3), 2009. 17, 18, 19, 125

[Kit13] Tracy Kitten. Bashas' breach exposes security flaws. BankInfoSecurity.com - The Fraud Blog, February 2013. 77

[KK95] George Karypis and Vipin Kumar. Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, 1995. 28, 111

[KKK+11] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Polo Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pages 245–260. Springer, 2011. 81, 82, 84

[KKVF14] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. VoG: Summarizing and understanding large graphs. In *Proceedings of the 14th SIAM International Conference on Data Mining (SDM)*, 2014. 30, 90

[KL51] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. 15

[Kle09] Victor Franklin Klebanoff. Method and system for assisting in the identification of merchants at which payment accounts have been compromised. US Patent US 8473415 B2, August 2009. 84

[KNRT03] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. On the bursty evolution of blogspace. In *Proceedings of the 12nd International World Wide Web Conference (WWW)*, Budapest, Hungary, 2003. 90

[Kor08] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 426–434. ACM, 2008. 141

[Kor10] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010. 141

[KPHF12] U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of the 18th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 316–324. ACM, 2012. 125

[Kre14] Brian Krebs. The target breach, by the numbers. KrebsOnSecurity.com, May 2014. 67

[KTS$^+$11] U Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. Gbase: a scalable and general graph management system. In *Proceedings of the 17th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1091–1099. ACM, 2011. 149

[LCK$^+$10] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research (JMLR)*, 11:985–1042, 2010. 65

[LCKF05] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pages 133–145. Springer, 2005. 140

[Ley02] Michael Ley. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. *SPIRE*, pages 1–10, 2002. 119

[Li05] Tao Li. A general model for clustering binary data. In *Proceedings of the 11th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 188–197. ACM, 2005. 32

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

cost_airlines. 125

[LKF07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *Transactions on Knowledge Discovery from Data (TKDD)*, 2007. 90

[LLDM08] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International World Wide Web Conference (WWW)*, pages 695–704, Beijing, China, 2008. 30, 31

[LNK07] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007. 140

[LS01] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001. 15, 21

[LYK$^+$08] Zheng Liu, J.X. Yu, Yiping Ke, Xuemin Lin, and Lei Chen. Spotting significant changing subgraphs in evolving graphs. In *Proceedings of the International Conference on Data Mining (ICDM)*, Pisa, Italy, 2008. IEEE. 90

[MAB$^+$10] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 135–146. ACM, 2010. 75

[ME11] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pages 437–452. Springer, 2011. 141

[Mie11] Pauli Miettinen. Boolean tensor factorizations. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 447–456. IEEE, 2011. 96

[Mir98] Boris Mirkin. Mathematical classification and clustering: From how to what and why. In *Classification, data analysis, and data highways*, pages 172–181. Springer, 1998. 16

[MMG$^+$08] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *Transactions on Knowledge and Data Engineering (TDKE)*, 20(10):1348–1362, 2008. 32

[MP02] Milena Mihail and Christos Papadimitriou. On the eigenvalue power law. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 254–262. Springer, 2002. 11

[MSF+12] Yasuko Matsubara, Yasushi Sakurai, Christos Faloutsos, Tomoharu Iwata, and Masatoshi Yoshikawa. Fast mining and forecasting of complex time-stamped events. In *Proceedings of the 18th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 271–279. ACM, 2012. 141, 142

[New04] Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004. 28

[New06] M. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences (PNAS)*, 103(23):8577–8582, 2006. 28

[NG04] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004. 28

[NGK+06] Scott A Neslin, Sunil Gupta, Wagner Kamakura, Junxiang Lu, and Charlotte H Mason. Defection detection: Measuring and understanding the predictive accuracy of customer churn models. *Journal of Marketing Research*, 43(2):204–211, 2006. 129

[NO09] Mario Navas and Carlos Ordonez. Efficient computation of pca with svd in sql. In *Proceedings of the 2nd Workshop on Data Mining using Matrices and Tensors*, page 5. ACM, 2009. 149

[NS14] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. *Proceedings of the 32nd International Conference on Machine Learning*, 2014. 140

[PAI13] Evangelos E. Papalexakis, Leman Akoglu, and Dino Ience. Do more views of a graph help? Community detection and clustering in multi-graphs. In *Proceedings of the 16th International Conference on Information FUSION*, pages 899–905, Istanbul, Turkey, 2013. 90, 111

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. 29

[Pea82] Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *AAAI*, pages 133–136, 1982. 84

[Pea85] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. 1985. 84

[Pei15] Tiago P Peixoto. Inferring the mesoscale structure of layered, edge-valued, and time-varying networks. *Physical Review E*, 92(4):042807, 2015. 91

[Pen56] Roger Penrose. On best approximate solutions of linear matrix equations. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 52, pages 17–19. Cambridge Univ Press, 1956. 18

[PF95] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *Transactions on Networking*, 3:226–244, 1995. 109

[PFS12] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pages 521–536. Springer, 2012. 16, 125

[PM10] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010. 16

[Pot97] Alex Pothen. Graph partitioning algorithms with applications to scientific computing. In *Parallel Numerical Algorithms*, pages 323–368. Springer, 1997. 28

[PPDSBLP12] Arnau Prat-Pérez, David Dominguez-Sal, Josep M Brunat, and Josep-Lluis Larriba-Pey. Shaping communities out of triangles. In *Proceedings of the 21st International Conference on Information and Knowledge Management*, pages 1677–1681. ACM, 2012. 28

[PPDSLP14] Arnau Prat-Pérez, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd International Conference on World Wide Web (WWW)*, pages 225–236. ACM, 2014. 28

[PRES11] Ioannis Psorakis, Stephen Roberts, Mark Ebden, and Ben Sheldon. Overlapping community detection using bayesian non-negative matrix factorization. *Physical Review E*, 83(6):066114, 2011. 29

[PSB13] Evangelos E. Papalexakis, Nicholaos D. Sidiropoulos, and Rasmus Bro. From k -means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *Transactions on Signal Processing*, pages 493–506, 2013. 19, 101

[PSS$^+$10] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2010. 111

[RB07] Martin Rosvall and Carl T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. *Proceedings of the National Academy of Sciences (PNAS)*, 104(18):7327–7331, 2007. 30

[Res15] Financial Technology Partners Research. Transaction security at the nexus of e-commerce, payment market structure complexity and fraud, August 2015. 67

[RG03] Alexander W Rives and Timothy Galitski. Modular organization of cellular networks. *Proceedings of the National Academy of Sciences (PNAS)*, 100(3):1128–1133, 2003. 30

[RG12] Parikshit Ram and Alexander G Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 931–939. ACM, 2012. 139

[Ris83] Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, pages 416–431, 1983. 13, 96

[Roy13] Abhirup Roy. Online retailer nomorerack.com probes likely card breach-report. Reuters, March 2013. 77

[S$^+$78] Gideon Schwarz et al. Estimating the dimension of a model. *The annals of Statistics*, 6(2):461–464, 1978. 29, 62

[ŞECA13] Umut Şimşekli, Beyza Ermiş, A Taylan Cemgil, and Evrim Acar. Optimal weight learning for coupled tensor factorization with mixed divergences. In *Proceedings of the 21st European Signal Processing Conference (EUSIPCO)*, pages 1–5. IEEE, 2013. 21

[SFFF03] Georgos Siganos, Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. Power-laws and the AS-level internet topology. *Transactions on Networking*, 11(4):514–524, 2003. 11

[SFPY07] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 687–696, San Jose, United States, 2007. ACM. 90, 111

[SKJ06] T. Sen, A. Kloczkowski, and R. Jernigan. Functional clustering of yeast proteins from the protein-protein interaction network. *BMC Bioinformatics*, 7:355–367, 2006. 1, 25

[SKZ+15] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. TimeCrunch: Interpretable Dynamic Graph Summarization. *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1055–1064, 2015. 90

[SL14] Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). *arXiv preprint arXiv:1410.5410*, 2014. 140

[SM00] J. Shi and J. Malik. Normalized cuts and image segmentation. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):888–905, 2000. 27, 30

[SN08] Umang Sharan and Jennifer Neville. Temporal-relational classifiers for prediction in evolving domains. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 540–549. IEEE, 2008. 141

[Sol15] LexisNexis Risk Solutions. Merchants contend with increasing fraud losses as remote channels prove especially challenging. LexisNexis True Cost of Fraud Study, September 2015. 67

[SPG+ne] Kevin Paul Siegel, Randi Annette Paynter, Robert L. Grossman, Christopher Brown, Charles Raymond Byce, Thomas Dwyer, and Aoyu Chen. System and method for identifying a point of compromise in a payment transaction processing system. US Patent US 8473415 B2, 2013 June. 84

[SSU03] Andrew I Schein, Lawrence K Saul, and Lyle H Ungar. A generalized linear model for principal component analysis of binary data. In *Proceedings of the 9th international workshop on artificial intelligence and statistics*, pages 14–21, 2003. 32

[STF06] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 374–383, Philadelphia, United States, 2006. ACM. 16, 90

[SXZF07] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs. In *Proceedings of the 7th SIAM International Conference on Data Mining (SDM)*, volume 127, page 366. SIAM, 2007. 65

[TBW11] C. Tantipathananandh and T. Y. Berger-Wolf. Finding communities in dynamic social networks. In *Proceedings of the International Conference on Data Mining (ICDM)*. IEEE, 2011. 111

[TFP08] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Random walk with restart: fast solutions and applications. *Knowledge and Information Systems (KAIS)*, 14(3):327–346, 2008. 29

[TGM15] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. Lemp: Fast retrieval of large entries in a matrix product. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 107–122. ACM, 2015. 139

[TSS05] Amos Tanay, Roded Sharan, and Ron Shamir. Biclustering algorithms: A survey. *Handbook of computational molecular biology*, 9(1-20):122–124, 2005. 51

[TWL09] Lei Tang, Xufei Wang, and Huan Liu. Uncovering groups via heterogeneous interaction analysis. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 503–512, Miami, United States, 2009. IEEE. 89, 111

[TY11] Xuning Tang and Christopher C Yang. Dynamic community detection with temporal dirichlet process. In *3rd International Conference on Privacy, Security, Risk and Trust (PASSAT) and 3rd International Conference on Social Computing (SocialCom)*, pages 603–608. IEEE, 2011. 91

[VFM$^+$14] Michail Vlachos, Francesco Fusco, Charalambos Mavroforakis, Anastasios Kyrillidis, and Vassilios G Vassiliadis. Improving co-cluster quality with application to product recommendations. In *Proceedings of the 23rd Conference on Information and Knowledge Management*, pages 679–688. ACM, 2014. 51

[Vij15] Jaikumar Vijayan. Schnucks supermarket chain struggled to find breach that exposed 2.4m cards. ComputerWorld, April 2015. 77

[WF94a] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994. 25, 26, 29, 30

[WF94b] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge, UK, 1994. 1

[WH04] Dennis M Wilkinson and Bernardo A Huberman. A method for finding communities of related genes. *Proceedings of the National Academy of Sciences (PNAS)*, 101(suppl 1):5241–5248, 2004. 30

[WW01] Max Welling and Markus Weber. Positive tensor factorization. *Pattern Recognition Letters*, 22(12):1255–1261, 2001. 19, 21, 119

[WYC$^+$13] Zhiang Wu, Wenpeng Yin, Jie Cao, Guandong Xu, and Alfredo Cuzzocrea. Community detection in multi-relational social networks. *Web Information Systems Engineering (WISE)*, pages 43–56, 2013. 89, 111

[XGFS13] Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 2:1–2:6, New York, NY, USA, 2013. ACM. 77

[XH13] Kevin S Xu and Alfred 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. 91

[Yan13] Shuo Yan. System and method for detecting account compromises. US Patent US 8600872 B1, December 2013. 84

[YCZ$^+$11] Tianbao Yang, Yun Chi, Shenghuo Zhu, Yihong Gong, and Rong Jin. Detecting communities and their evolutions in dynamic social networks–a bayesian approach. *Machine Learning*, 82(2):157–189, feb 2011. 91

[YFK15] Yuto Yamaguchi, Christos Faloutsos, and Hiroyuki Kitagawa. Socnl: Bayesian label propagation with confidence. In *Advances in Knowledge Discovery and Data Mining*, pages 633–645. Springer, 2015. 84

[Yıl12] Yusuf Kenan Yılmaz. *Generalized tensor factorization*. PhD thesis, Bogaziçi University, 2012. 21

[YL12a] Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 1170–1175. IEEE, 2012. 27, 29, 34, 46

[YL12b] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 745–754. IEEE, 2012. 25, 34, 120

[ZCF$^+$10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. *HotCloud 2010*, 2010. 75, 125

[Zel62] Arnold Zellner. An efficient method of estimating seemingly unrelated regressions and tests for aggregation bias. *Journal of the American Statistical Association*, 57(298):348–368, 1962. 142

[ZH05] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005. 13

[Zho03] Haijun Zhou. Distance, dissimilarity index, and network community structure. *Physical Review E*, 67:061901, Jun 2003. 29

[ZL04] Haijun Zhou and Reinhard Lipowsky. Network brownian motion: A new method to measure vertex-vertex proximity and to identify communities and subcommunities. In *Proceedings of the International Conference on Computational Science*, pages 1062–1069. Springer, 2004. 29

[ZLD$^+$10] Zhong-Yuan Zhang, Tao Li, Chris Ding, Xian-Wen Ren, and Xiang-Sun Zhang. Binary matrix factorization for analyzing gene expression data. *Data Mining and Knowledge Discovery (DMKD)*, 20(1):28–52, 2010. 32, 51

[ZWSW10] Scott M. Zoldi, Liang Wang, Li Sun, and Steven G. Wu. Mass compromise/-point of compromise analytic detection and compromised card portfolio management system. US Patent 7761379 B2, July 2010. 84

[ZWWZ09] Yuzhou Zhang, Jianyong Wang, Yi Wang, and Lizhu Zhou. Parallel community detection on large networks with propinquity dynamics. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 997–1006. ACM, 2009. 28