Boston University

OpenBU **http://open.bu.edu**

Theses & Dissertations Boston University Theses & Dissertations

2016

Matrix completion with structure

https://hdl.handle.net/2144/19743

Boston University

BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

MATRIX COMPLETION WITH STRUCTURE

by

NATALI RUCHANSKY

B.S., Boston University, 2011

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
2016

© 2016 by
NATALI RUCHANSKY
All rights reserved

Approved by

First Reader

Evimaria Terzi, PhD
Associate Professor of Computer Science

Second Reader

Mark Crovella, PhD
Professor of Computer Science

Third Reader

George Kollios, PhD
Professor of Computer Science

Everything starts from a dot.
Wassily Kandinsky

Acknowledgments

Dedicated to my grandfather.

I would like to thank my advisors Evimaria Terzi and Mark Crovella. I was extremely fortunate to have had advisors whose easygoingness, curiosity, and excitability matched my own. I am eternally grateful for the flexibility they gave me in my work, and their patience when I was too intrigued by an irrelevant topic. Among all that I have learned, there is one thing I will always take with me: that ”at least we learned something.”

Thanks to all my friends. Starting from my pre-PhD friends: thank you for still being my friends. To my diverse but equally suffering PhD friends, Davide Proserpio, Dimitri Papadopoulos, Jeffrey Finkelstein, Harry Mavroforakis, and Ben Fuller: thank you for consistently peer-pressuring me to stop working. Thank you to the many other dear friends I have made at BU whom I cannot all list, especially my office-mates Giovanni Comarela and Bashir Rastegarpanah for making sure I wasn't frozen from the office air conditioning, and Larissa Spinelli for feeding me chocolate.

Thank you to the staff for doing the dirty work, and especially to Chris and Nora for providing a level of abstraction, but more importantly for the jokes. I am indebted to the Security group for keeping me well-fed by allowing me to steal their seminar sandwiches; thanks Sharon Goldberg and Leo Reyzin. I am also deeply grateful to both of you for always opening your doors, offering great advice, and giving support throughout my years of undergrad and PhD.

Finally, the largest thank you to my parents and my family. For the sacrifices you have made for my cousins and me, for your love and support, and for never letting me take myself too seriously. Parents, thank you for giving me the luxury to learn whatever my heart desired, and being patient when that something was nothing.

MATRIX COMPLETION WITH STRUCTURE

NATALI RUCHANSKY

Boston University, Graduate School of Arts and Sciences, 2016

Major Professors: Evimaria Terzi, PhD
Associate Professor of Computer Science

Mark Crovella, PhD
Professor of Computer Science

ABSTRACT

Often, data organized in matrix form contains missing entries. Further, such data has been observed to exhibit effective low-rank, and has led to interest in the particular problem of low-rank matrix-completion: Given a partially-observed matrix, estimate the missing entries such that the output completion is low-rank. The goal of this thesis is to improve matrix-completion algorithms by explicitly analyzing two sources of information in the observed entries: their locations and their values.

First, we provide a categorization of a new approach to matrix-completion, which we call structural. Structural methods quantify the possibility of completion using tests applied only to the locations of known entries. By framing each test as the class of partially-observed matrices that pass the test, we provide the first organizing framework for analyzing the relationship among structural completion methods.

Building on the structural approach, we then develop a new algorithm for active matrix-completion that is combinatorial in nature. The algorithm uses just the locations of known entries to suggest a small number of queries to be made on the missing entries that allow it to produce a full and accurate completion. If a budget is placed on the number of queries, the algorithm outputs a partial completion, indicating which entries it can and cannot accurately estimate given the observations at hand.

Finally, we propose a local approach to matrix-completion that analyzes the values of the observed entries to discover a structure that is more fine-grained than the traditional low-rank assumption. Motivated by the Singular Value Decomposition, we develop an algorithm that finds low-rank submatrices using only the first few singular vectors of a matrix. By completing low-rank submatrices separately from the rest of the matrix, the local approach to matrix-completion produces more accurate reconstructions than traditional algorithms.

Contents

1 **Introduction to matrix-completion** **1**

1.1 The history of matrix-completion 1

1.2 Contributions of this thesis 4

2 **Related work** **7**

2.1 Statistical matrix completion 7

2.2 Structural matrix completion 11

2.3 Active matrix completion 12

2.4 Localized matrix completion 13

2.5 Low-rank submatrix discovery 14

3 **Landscape of completability** **17**

3.1 Notation and assumptions 17

3.2 Classes of completability 18

3.3 Locally completable masks 20

3.4 Uniquely completable masks 22

3.4.1 Closure ($\mathbb{CC}$): 23

3.4.2 Stress ($\mathbb{SC}$): 25

3.4.3 Information cascade ($\mathbb{ICMC}$): 26

3.5 Categorization 28

3.5.1 Theoretical exploration 29

3.5.2 Empirical exploration 34

4 Active structural-completion **37**

4.1 Notation and assumptions 39

4.2 The problem of active completion 41

4.3 The `Order&Extend` algorithm 43

4.3.1 Completion as a sequence of linear systems 44

4.3.2 Ordering and fixing incomplete systems 48

4.3.3 Finding and alleviating unstable systems 49

4.3.4 Putting everything together 54

4.4 Experimental results 57

4.4.1 Methodology 59

4.4.2 Evaluating `Order&Extend` 60

4.5 Extension to tensors 64

4.5.1 Completion as a linear system of equations 65

4.5.2 Hypergraph propogation 67

4.5.3 The `Order&ExtendTensor` algorithm 69

4.5.4 Experiments 69

4.6 Active completability for other classes 71

4.6.1 Bringing a mask into $\mathbb{L}$ 71

4.6.2 Bringing a mask into $\mathbb{SC}$ 73

4.6.3 Bringing a mask into $\mathbb{CC}$ 74

5 Localized matrix-completion **76**

5.1 Notation and assumptions 77

5.2 Localized matrix-completion 78

5.3 Low-rank submatrix discovery 79

5.3.1 Exposing low-rank submatrices 80

5.3.2 An algorithm for extracting low-rank submatrices 98

5.3.3 Improvements for matrices with missing entries 100
5.4 Experiments 103
5.4.1 Evaluation of localized matrix-completion 103
5.4.2 Evaluation of `SVP` 108

6 Conclusions 115

A 117

A.1 Background on graph rigidity 117

References 121

Curriculum Vitae 125

List of Tables

5.1 The running time of `SVP`, `BinaryLoops`, and `LRSC` in seconds for an $n \times n$ input matrix $\mathbf{M}$. 113

List of Figures

2·1 Example of the possible error in completion. 10

2·2 A ground truth matrix and two different partially observed matrices. 11

3·1 The mask graph G_Ω of mask $\Omega = \{(1,1), (1,2), (1,3), (2,2), (3,1)\}$. . 18

3·2 The structure of the completion matrix. 21

3·3 An example of a mask that is uniquely completable at $r = 3$ but does not satisfy the `check-CC` condition (for Lemma 7). 29

3·4 Example of re-ordering of rows and columns. 31

3·5 Every entry that is ICMC-recovered is also CC-recovered. 31

3·6 Example of a mask in $\mathbb{CC}$ but not in $\mathbb{ICMC}$ for $r = 3$. 33

3·7 Containment of completability classes. 34

3·8 *Random* masks with density = 0.08, 0.10, 0.12, 0.14. 35

3·9 *PowerLaw* masks with density = 0.035, 0.040, 0.045, 0.050. 35

4·1 The mask graph G_Ω of mask $\Omega = \{(1,1), (1,2), (1,3), (2,2), (3,1)\}$. . 40

4·2 An intermediate step of `Sequential` algorithm. 45

4·3 The direction on the edges imposed by the ordering of nodes shown in Figure 4·2. 48

4·4 RelError of completion achieved by `Order&Extend`, `LMaFit` and `OptSpace` on datasets with *approximate* rank; x-axis: query budget b; y-axis: RelError of the completion. 57

4·5 RelError of completion achieved by `Order&Extend`, `LMaFit` and `OptSpace` on datasets with *exact* rank; x-axis: query budget b; y-axis: RelError of the completion. 60
4·6 Recovery process using `LMaFit`, and `Order&Extend`. Each column is a particular b, increasing from left to right. 63
4·7 Visual of tensor CP-decompositon. 66
4·8 Hypergraph for tensor completion 67
4·9 RelError of completion achieved by `Order&ExtendTensor`, `HardCompletion` and `CP` on datasets with *exact* rank; x-axis: query budget b; y-axis: RelError of the completion. 70

5·1 Relative error in completion on the synthetic data. 105
5·2 Relative error in completion on a matrix with multiple low-rank submatrices. 106
5·3 Relative error in completion for the Traffic dataset. 107
5·4 F-Score achieved by the `SVP`, `BinaryLoops`, and `LRSC` algorithms as: the rank ℓ of $\mathbf{S}$, the size of $\mathbf{S}$, the background rank r, or π vary. . . . 109
5·5 F-Scores of `SVP`, `BinaryLoops`, and `LRSC` in robustness studies.In (a) $\mu = 1$ and σ varies and in (b) $\sigma = 1$ and μ varies. 111
5·6 Comparison of algorithms for finding the singular vectors of an incomplete matrix in terms of the F-Score of `SVP` as a function of the percentage of known entries in the data. 112
5·7 Left: F-Score of `SVP`, `BinaryLoops`, and `LRSC` on data with multiple low-rank submatrices as a function of the rank of the submatrices. . . 113

A·1 A rigid structure. 118

Chapter 1

Introduction to matrix-completion

1.1 The history of matrix-completion

Much of the data we encounter nowadays is obtained by measurement: from recommender systems collecting user-preference data, to scientists measuring levels of protein interaction, to monitors measuring road or Internet traffic. A common way to represent data is using a matrix. For example, city traffic measurements can be organized into a matrix where rows are sources, columns are destinations, and each entry holds the volume of traffic that passed from a particular source to a particular destination. As another example, rows and columns can represent users and movies, and each entry can carry the rating that a particular user gave to a movie. There are a variety of reasons why a city may collect traffic volume data, for example, to identify areas where congestion commonly occurs or to estimate the C02 emission through the city. At the same time, a city most likely does not have the resources to measure the traffic between all possible pairs of destinations. This limitation means that the volumes on some source-destination pairs are not known, leading to a matrix representation with unknown values in some entries. To assess the congestion or pollution throughout, the city must be able to estimate these unknown values. This task of estimating missing values in data arises in a vast array of applications, and can be formalized as the problem of matrix completion: *given a partially-observed matrix, estimate the missing entries.*

The problem of matrix completion traces it roots to the 1980's. At that time mathematicians and engineers were especially interested in the following problem: given a matrix capturing partial Euclidean distance or geometry information, can it be completed in a way that preserves the Euclidean property. As Johnson (1990) and Lee and Seol (2001) explain, this led to the study of finding a matrix consistent with the observed entries so that it has a particular property, with the most studied being positive-definiteness.

More recently, there have been significant new algorithmic advances for the problem of *low-rank matrix completion*. In this version of the problem, the key assumption made about the matrix is that it is either exactly or approximately low-rank. The rank of a matrix is the number of independent rows or columns it contains. When the matrix is composed of measurement data, the underlying fully-known matrix is typically of full rank due to noise or error; however, it has been empirically observed that the *effective rank* is low. Intuitively this means that while all rows and columns of the matrix are independent, only a few rows and columns are needed to find a good approximation of the matrix. For example, the data used in the Netflix Prize competition is a matrix of ratings from 480K users over 18K movies, and the estimated rank of the matrix is only about 20-40. The rank can be interpreted as the fact that there are about 20-40 features of movies that can predict a user's movie preference, for example, genre, or language, or year. Since only 1% of the entries are observed, estimating the unknown user-movie ratings can be framed as an instance of low-rank matrix completion.

Around 2007, there was a surge in progress on low-rank matrix-completion, brought on by connections between low-rank matrix-completion and compressive sensing in the statistical signal processing community. Analytic insights built on this connection yielded algorithms capable of solving low-rank matrix completion even when the vast majority of elements are missing – carrying significant practical impact. These algorithms typically address a problem where the goal is to minimize the rank of the estimate while keeping the observed entries intact. While this precise formulation is NP-hard, it was recognized by Candès and Tao (2010) to be solvable approximately via convex relaxation under certain fairly general conditions. Following this initial result, a flurry of new algorithms for low-rank matrix-completion have emerged, which we refer to as *statistical*, including those developed by Candès and Recht (2012), Chen et al. (2014), Jain et al. (2013), Keshavan et al. (2010a,b), Wen et al. (2012).

Statistical matrix-completion algorithms have three main characteristics. First, statistical approaches assume that the locations of known entries follow a random model; that is, the known observations are fairly evenly dispersed throughout the matrix without any particular pattern. Second, there is the additional assumption that there are enough known entries with respect to the size and rank of the matrix. Third, the problem is posed as an optimization where the objective is to minimize the rank of the estimate while remaining close to the initially observed entries. A key property of statistical methods is that they output a completion of the partially observed matrix without an analysis of whether the information contained in the visible entries satisfies the two main assumptions. Since the error is evaluated over only the observed entries, it is often not representative of the error in the estimate of the unobserved entries.

Recently a new class of *structural* matrix-completion, introduced by Király et al. (2013), Király and Tomioka (2012b), Meka et al. (2009), Singer and Cucuringu (2010), has started to address the limitations of statistical methods. Structural methods combine tools from areas such as graph theory and algebraic geometry, to characterize the possibility of producing an accurate completion; specifically, these methods identify whether the number of completions of a partially observed matrix is infinite, finite, one, or none. A key observation shared by all structural approaches is that this number does not depend on the values of the observed entries, but rather only on their positions. That is, whereas statistical methods seek to show that particular sampling of the observed entries is sufficient for reconstruction with high probability, structural methods explicitly analyze the information content of the visible entries, and are capable of stating definitively that there is sufficient information for reconstruction. Not only that but unlike the statistical methods that give some estimate for any (even the smallest) set of observations, structural methods specify precisely which elements can and cannot be recovered.

To date, no method exists that allows one to unambiguously state whether a set of observed entries is sufficient for reconstruction. Rather, what are known are three methods that are partially effective for making this identification.That is, there are three algorithms proposed in the literature so far that have the property that if they return "yes," it is possible to complete the input partially observed matrix – but if they return "no," no conclusion can be drawn. Further, there has been no study on the relationship between these three methods, nor on the relationship between the partially observed matrices for which each returns "yes".

1.2 Contributions of this thesis

In this thesis, we start by providing the first common framework for analyzing the structural matrix completion methods. For each method we define the associated *class* of matrices as those for which the method answers 'yes'. Specifically we define: the $\mathbb{SC}$ class for matrices that pass the tests described in Singer and Cucuringu (2010), the $\mathbb{CC}$ class for the test in Király et al. (2013), and the $\mathbb{ICMC}$ class for the test implied by Meka et al. (2009). Through the framework, we provide new results on the relationship between the different classes, and new insights regarding their sizes and organization. We establish some basic proven relationships between the classes, and then investigate the remainder through MCMC-style experiments on random partially observed matrices. This understanding is valuable as an organizing guide to the state of the art in structural matrix-completion, and hence comprises the first contribution of this thesis.

Through a theoretical and empirical exploration of the framework, we make the interesting observation that despite being the hardest class to fall into randomly, the $\mathbb{ICMC}$ class is the easiest to move into. In other words, a random partially-observed matrix is not likely to satisfy the $\mathbb{ICMC}$ condition, yet we find that there is a natural way to bring the matrix into the class with the addition of a few entries. From an applications standpoint, adding new entries to a matrix by querying is very natural, for example, Netflix can simply send out a user-preference survey. Based on this observation we develop a combinatorial algorithm for the problem of active matrix-completion. Given a partially observed matrix that cannot be completed, the proposed algorithm analyzes the locations of observed entries to pinpoint portions of the data the can be completed accurately. By modeling the problem as an epidemic propagation on a graph, the algorithm then identifies a small number of additional entries that need to be added so that the matrix is a member of $\mathbb{ICMC}$ and there is enough information for full recovery. Finally, the algorithm tackles numerical error and noise to achieve an almost perfect completion of the missing entries. We also discuss a natural extension to active tensor-completion.

Both statistical and structural matrix-completion algorithms rely on the assumption that the whole matrix is of a given rank r; in fact, the incoherence and genericity assumptions made in much of the work imply that submatrices are also approximately rank r. However, in working with active matrix-completion on matrices corresponding to real data an interesting property was observed that countered traditional assumptions and was occasionally a source of error: the presence of lower-rank submatrices. In fact, in the design of many algorithms for analyzing and utilizing this data, there is the underlying assumption of the existence of such lower-rank submatrices. For example, in user-preference data, there are expected to be subsets of users that behave similarly on subsets of products. Such similarity is utilized in standard collaborative filtering where recommendations are made to a user based on the preferences of other similar users Aggarwal (2016), Herlocker et al. (2004), Sarwar et al. (2001), as well as in the analysis of traffic networks, user-preferences, and datasets in the natural sciences (e.g., gene expression data).

After demonstrating that traditional matrix-completion algorithms do not accurately estimate partially observed matrices that contain low-rank submatrices, we propose a `Localized` approach. The key is to isolate the low-rank submatrices and complete them separately from the rest of the data. To do this, we first study the problem of finding a low-rank submatrix which is an interesting problem even when the matrix is fully known. We develop an algorithm based on the Singular Value Decomposition that uses the first singular vectors of the matrix to find the low-rank submatrix. A central contribution is an analysis of when this algorithm is expected to succeed. With an extension to partially observed matrices, we apply to algorithm towards matrix completion and demonstrate that the `Localized` approach finds a more accurate completion than the traditional `Global` one.

Chapter 2

Related work

In this chapter, we give an overview of the work related to this thesis. We start with a distinction between two types of approaches to matrix completion which we call statistical and structural, and discuss the work related to each. Next, we examine two variants of the problem: active and localized matrix completion. Active matrix completion allows for a small number of additional entries to be revealed, and localized completion targets local structure in matrices.

2.1 Statistical matrix completion

Historically, the first methods for low-rank matrix completion to be developed were statistical in nature, for example: Bhojanapalli and Jain (2014), Candès and Recht (2012), Candès and Tao (2010), Chen et al. (2014), Jain et al. (2013), Keshavan et al. (2010a,b), Negahban and Wainwright (2012), Wen et al. (2012). Statistical matrix-completion algorithms have three key features. First, they assume that the locations of known entries follow a random model; that is, the known observations are fairly evenly dispersed throughout the matrix without any particular pattern. Second, there is the additional assumption that there are enough known entries with respect to the size and rank of the matrix. Third, the problem is posed as an optimization where the objective is to minimize the rank of the estimate while remaining close to the initially observed entries.

Consider an $n \times m$ true matrix $\mathbf{M}$ of rank r with entries as real numbers, and its partially observed version $\mathbf{M}_\Omega$ with $\Omega \subseteq \{(i,j) | 1 \leq i \leq n,\ 1 \leq j \leq m\}$ denoting the set of known entries. The standard optimization formulation is:

$$\min \operatorname{rank}(\hat{\mathbf{M}}) \text{ subject to } \hat{\mathbf{M}}(i,j) = \mathbf{M}(i,j)\ \forall (i,j) \in \Omega$$

That is, minimize the rank of the estimate while keeping the observed values as accurate as possible. Since minimizing the rank of a matrix is NP-hard, a variety of different formulations have been proposed that are often targeted towards specific additional assumptions. In a seminal work, Candès and Tao (2010) showed that the problem can be optimized efficiently when the rank is replaced with the nuclear norm $\|\hat{\mathbf{M}}\|_* = \sum_i \sigma_i$ where σ_i is the i-th largest singular value of $\hat{\mathbf{M}}$. In other words, the NP-hard formulation is replaced with the convex relaxation:

$$\min \|\hat{\mathbf{M}}\|_* \text{ subject to } \hat{\mathbf{M}}(i,j) = \mathbf{M}(i,j)\ \forall (i,j) \in \Omega$$

The above can be solved efficiently to produce an accurate estimate if certain conditions hold and the number of observed entries is above a certain threshold.

Under the assumption of randomly sampled locations of known entries, Candès and Recht (2012) prove that an $n \times m$ matrix of rank r should have at least $|\Omega| > Cn^{\frac{6}{5}} r \log(n)$ for their algorithm to succeed with high probability, where n is the largest of m and n without loss of generality. The high-level idea comes from the insight that even a rank-1 matrix cannot be completed unless every row and column have at least one known entry. Assuming samples are taken uniformly at random, the Coupon Collector problem tells us that given n items, $n \log(n)$ samples are needed to cover each item. Hence, if we want at least one observation in each row and column of an $n \times n$ matrix, an order of $n \log(n)$ random samples are needed. For arbitrary rank r this translates to an $nr \log(n)$ lower bound for accurate completion.

We highlight that in practice, this bound often corresponds to a significantly large number of samples. For example, adopting the rank $r \approx 40$ of top solutions to the Netflix Challenge, the number of known ratings is over 151 million entries below the threshold.

Following this result numerous other approaches have been proposed, each with a threshold that differs slightly but is essentially $O(nr\log(n))$. We highlight two algorithms in particular for their efficiency, generality, and ability to estimate the rank of a matrix: `LMaFit` and `OptSpace`.

`LMaFit` was proposed by proposed by Wen et al. (2012). The starting observation is that the matrix $\mathbf{M}$ can be written as the product of two factors $\mathbf{M} = \mathbf{XY}$ where $\mathbf{X}$ is size $n \times r$ and $\mathbf{Y}$ is size $r \times m$. The problem is then formalized as:

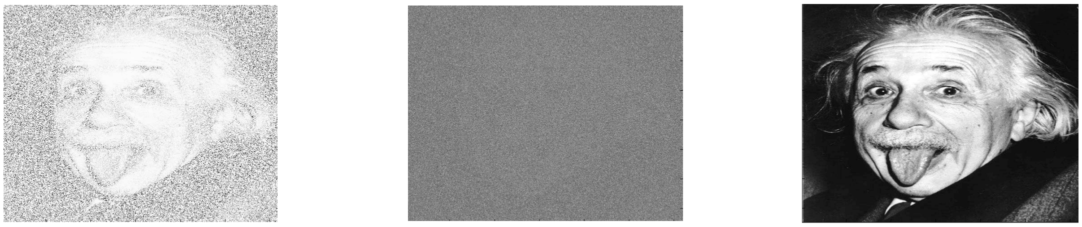
$$\begin{aligned} \underset{\mathbf{M},\mathbf{X},\mathbf{Y}}{\text{minimize}} \quad & \frac{1}{2}\|\hat{\mathbf{M}} - \mathbf{XY}\|_f^2 \\ \text{subject to} \quad & \hat{\mathbf{M}}(i,j) = \mathbf{M}_\Omega(i,j)\ \forall(i,j) \in \Omega \end{aligned}$$

The algorithm is then based on an alternating least-squares method where two of $\hat{\mathbf{M}}$, $\mathbf{X}$, and $\mathbf{Y}$ are fixed and the third is solved for. The approach is shown to converge under mild assumptions.

`OptSpace` was proposed by Keshavan et al. (2010a), and solves a slightly different optimization problem formulated as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\hat{\mathbf{M}}_\Omega - \mathbf{M}_\Omega\|_f \\ \text{subject to} \quad & \text{rank}(\hat{\mathbf{M}}) < r \end{aligned}$$

A major distinction in this version of the problem is that the objective is to minimize the error over (only) the observed entries while keeping the rank small. The algorithm combines spectral methods with a convex-optimization step that aims to minimize the disagreement in the observed entries. First the algorithm estimates the rank of the matrix using an important trimming step of removing overrepresented rows and



(a) Partially observed. (b) Consistent estimate. (c) True underlying object

Figure 2·1: Example of the possible error in completion.

columns. After this is done, the objective is minimized using gradient descent.

Limitations: A key property of statistical methods is that they output a completion of the partially observed matrix on *any* input, whether or not the information contained in the visible entries satisfies the two main assumptions. Since the error is evaluated over only the observed entries, it is often not representative of the error in the estimate of the unobserved entries. For example, consider the partially observed matrix in Figure 2·1a where the missing entries are white. Despite the fact that the matrix is 80% zeros, the human eye can still recognize this as a famous photo of Einstein. Now consider an application of a matrix completion algorithm that produces a full estimate completion depicted in Figure 2·1b. Since the matrix in Figure 2·1b is approximately rank-1 and the initially observed entries are identical to 2·1a, it is deemed an accurate completion despite the fact that it is visibly very far from the true completion depicted in Figure 2·1c.

In the absence of noise, this type of error occurs when the rank and the set of observed entries are not constraining enough, allowing multiple possible completions. In other words, there are many ways to fill in the zeros in Figure 2·1a such that the completion has rank-1. To better understand this idea of multiple consistent completions, consider the rank-1 matrix in Figure 2·2a and suppose that it is only partially observed as in Figure2·2b. On input $\mathbf{M}_\Omega$, statistical matrix completion algorithms try to find a rank-1 matrix with a small error on the two observed entries.

Given this approach, the estimate in Figure 2·2c is a highly accurate estimate because the matrix is rank-1 and has zero error on the observed entries; however, it is clearly very different from the ground truth. In fact, if the second row in the estimate is any multiple of the first row it will match both the observed entries and the rank. In other words, there are infinitely many consistent completions that have zero error in the statistical sense but are not the true $\mathbf{M}$.

$$\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix} \qquad \mathbf{M}_\Omega = \begin{bmatrix} 1 & 2 \\ ? & ? \end{bmatrix} \qquad \hat{\mathbf{M}} = \begin{bmatrix} 1 & 2 \\ 500 & 1000 \end{bmatrix} \qquad \mathbf{M}_\Omega = \begin{bmatrix} 1 & 2 \\ 3 & ? \end{bmatrix}$$

(a) Ground truth (b) Many solutions (c) Possible estimate (d) Single solution

Figure 2·2: A ground truth matrix and two different partially observed matrices.

Alternatively, consider the partially observed matrix in Figure 2·2d. With just the single additional entry fixed, there is now only a single completion that has rank-1 and zero error on the observed entries. That is, since $\mathbf{M}_\Omega(2,1) = 3$ is fixed, the only consistent rank-1 matrix is the one in Figure 2·2a.

This example shows that the statistical notion of error is not always representative of the quality of the completion (with respect to the ground truth). For the partially observed matrix in Figure 2·2d, an error of zero over the observed entries is indicative of having produced a good estimate, but in Figure 2·2b it is not. At the same time both match the observed entries perfectly.

2.2 Structural matrix completion

While statistical approaches to matrix completion often provide good results, there are cases where they fail without explanation or indication. Recently, a new approach has been proposed which we call *structural* and which was catalyzed by a connection between matrix completion and graph rigidity. (An overview of the connection is given in Appendix A.1.) Structural methods analyze the information contained in the visible entries to determine whether an accurate reconstruction is possible. This notion of possibility is quantified by the number of consistent completions of a partially observed matrix: infinite, finite, one, or none.

A critical observation shared by all structural approaches is that the number of possible completions does not depend on the values of the observed entries, but rather only on their positions. This statement proved by Király et al. (2013), Singer and Cucuringu (2010), was used to develop conditions under which a partially observed matrix has a certain number of completions. In fact, Singer and Cucuringu (2010) showed that there is a necessary and sufficient condition that can be used to test if a partially observed matrix has a *finite* number of completion. No such condition has been developed for the stronger case of a *unique* completion. However, there do exist three conditions that are sufficient, meaning if a partially observed matrix satisfies a particular condition then it has a single unique completion, but if it does not meet the condition, then no conclusion can be drawn.

The conditions for *finite* and *unique* completion are the focus of Chapter 3 of this thesis where we present new insights regarding each algorithm and the relationships between them.

2.3 Active matrix completion

Structural and statistical matrix completion are two branches of matrix completion algorithms. While fundamentally different in their approach, both requries a certain number of known entries to work well; unfortunately real world data often has far fewer observations than the methods require. This brings about the active version of matrix completion that asks for the most accurate completion given an allowance of queries to unknown entries.

Although active learning has been studied for some time, work in the active matrix completion area has only appeared recently in Chakraborty et al. (2013), Sutherland et al. (2013). In both these works, the authors are interested in determining which entries need to be revealed in order to reduce error in matrix reconstruction. Their methods choose to reveal entries with the largest predicted uncertainty based on various measures. A characteristic of these approaches is that they construct a querying strategy *independently* of the completion algorithm, and then use off-the-shelf matrix completion algorithms for the reconstruction phase. The fact that queries are not chosen in consideration with consideration of the completion approach limits the success of the algorithms. These methods appear to have other drawbacks. In the experiments, Chakraborty et al. (2013) start with partial matrices where 50-60% of entries are already known – far greater many statistical and structural thresholds. Further, their proposed query strategy does not lead to a significant improvement over pure random querying. While Sutherland et al. (2013) report low reconstruction error, the main experiments are run over 10×10 matrices, providing no evidence that the methods scale.

2.4 Localized matrix completion

The traditional approaches to matrix completion, both statistical and structural, make assumptions on the matrix as a whole. However, in many real-world settings, there is often the implicit assumption that the data contains substructure that does not adhere to the global model.

The benefit of capturing substructure in the data has been observed in the context of several other problems, such as image colorization by Kwok (2015) and matrix factorization by Lee et al. (2013); the common theme is a shift from a single global model to a combination of smaller models built from subparts of the data that are specified as input. The problem of local matrix factorization studied by Lee et al. (2013) is the most related to our work on active completion, but differs mainly in two ways. First, the specific location of substructure in the matrix is assumed to be an input to the problem. The algorithm receives the pairwise distances between rows and columns and uses a kernel function to specify a neighborhood of close points for each row-column pair, in other words, to specify a neighborhood submatrix for each entry.

The goal is to find the factorization of each neighborhood submatrix, and then incorporate this *local* factorization into the *global* model. Since this approach is expensive, the authors propose to sample q anchor points, find factorizations of each of the q neighborhoods, and then model each entry in the matrix as a linear combination of these q factorizations. Hence the second difference is modeling the data as a linear combination of small factorization, whereas we argue that the parts of the data should be modeled with their own factorizations.

2.5 Low-rank submatrix discovery

In the previous section, we described a new localized approach to matrix completion. An essential tool in any localized approach is the ability to find low-rank submatrices in a matrix (that is unless the locations are assumed to come as input). Although there is work on discovering interesting local structures in the data, there is little work on the particular problem of discovering low-rank submatrices embedded in larger matrices. Below we review two main lines of research that are the most related to the problem of finding a low-rank submatrix which we call LRDiscovery.

To the best of our knowledge, Rangan (2012) was the first to explicitly ask the question of finding a low-rank submatrix from a larger matrix. His work focuses on a particular instance where the entries of $\mathbf{M}$ *and* the entries in the submatrix

S adhere to a standard Gaussian distribution with mean zero and variance one. Assuming such structure, Rangan poses the problem of finding the largest submatrix with rank less than five. The proposed algorithm searches for a low-rank submatrix by comparing the ±-sign patterns of the values in rows and columns. Due to the underlying assumptions, Rangan's algorithm is restricted to succeed in cases where the rank of the submatrix is less than five and its size is greater than $\sqrt{n}$, where n is the size of the whole dataset. In contrast, our analysis does not make the assumptions required by Rangan and instead of using the signs of the data it uses the data values themselves. As a result, the algorithm propose in Chapter 5.3 (called `SVP`) succeeds on a larger range of submatrix sizes and ranks. Furthermore, the output of Rangan's algorithm is nested collections of rows and columns, and it is not clear how to decide which collection corresponds to the desired solution. In contrast, we propose algorithm directly outputs a subset of rows and columns indexing the discovered submatrix. Finally, Rangan's algorithm requires setting tuning a parameter while `SVP` is free of such parameters and thus easier to use in practice.

The second line of work that is related to LRDISCOVERY, though it does not address the same problem, is *subspace clustering* (SbC), as studied in Elhamifar and Vidal (2009), Vidal and Favaro (2014). SbC assumes that the rows of **M** are drawn from a union of *independent* subspaces, and the problem is to find a clustering of the rows into separate subspaces. SbC is related to LRDISCOVERY in the special case where the submatrix **S** is a subset only of rows and spans all columns, and where **S** is also an independent subspace. Assuming independence of subspaces, the algorithms for SbC can accurately cluster the rows of **M** based on the representation of each row in terms of the others. In our experiments, we observe that these algorithms – appropriately modified for our problem cannot accurately separate **S** when the values in **S** are not significantly larger than the rest. We also observe that they are sensitive to the presence of large values elsewhere in the data. In contrast, our analysis does not assume that $\mathbf{S}$ is an independent subspace and our algorithm – inspired by this analysis – proves to be much more resilient than the corresponding algorithms for SbC. Regarding running time, `SVP` is significantly more efficient than the SbC algorithms; after all the former only requires computing the first singular vector of $\mathbf{M}$ while the latter rely on the computation of the full SVD of the input matrix.

Our experimental evaluation with data generated using different models demonstrates that `SVP` can succeed in the identification of $\mathbf{S}$ even when $\mathbf{S}$ has relatively large rank, or when there are multiple low-rank submatrices planted in $\mathbf{M}$. None of the other methods available for the same problem today can exhibit the same level of success in such a wide range of inputs. Further, we demonstrate that unlike `BinaryLoops` and `LRSbC`, `SVP` extends naturally to matrices with missing entries, a property which is crucial for the application of matrix-completion

Chapter 3

Landscape of completability

In this chapter, we introduce a novel framework organizing the existing techniques of structural matrix completion which has not been done to date. Structural algorithms quantify the number of possible completions of a partially observed matrix (a finite, an infinite, one, or none) by checking if the matrix satisfies a particular condition. A key insight is that the number of completions depends only on the locations of known entries of a partially observed matrix, i.e. the *mask*. Hence, using only the locations of known entries we study the relationships between the conditions that exist so far, and between the matrices that satisfy them. For the duration of this chapter we use the mask, mask graph, and partially observed matrix interchangeably as one implies the other.

3.1 Notation and assumptions

Throughout this chapter we use $\mathbf{M}$ to refer to an $n \times m$ fully-known matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ of rank r, and $\mathbf{M}(i, j)$ to the entry in the ith row and jth column. The number of degrees of freedom in the matrix is called $\theta = r(n + m - r)$. If $\mathbf{M}$ is not completely known we use $\Omega \subseteq \{(i, j) | 1 \leq i \leq n, 1 \leq j \leq m\}$ to denote the subset of entries that are observed; we call this set of known entries the *mask of* $\mathbf{M}$. When a matrix completion algorithm is applied to the partially observed $\mathbf{M}_\Omega$, we use $\hat{\mathbf{M}}$ to denote the output estimate.

Each mask is also associate with a bipartite *mask graph* $G_\Omega = (V_1, V_2, E)$. Every

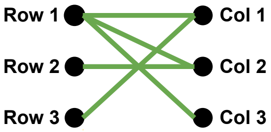


Figure 3·1: The mask graph G_Ω of mask $\Omega = \{(1,1),\ (1,2),\ (1,3),\ (2,2),\ (3,1)\}$.

node $i \in V_1$ represents a row of $\mathbf{M}$ and every $j \in V_2$ represents a column. An edges $(i,j) \in E$ exist in G_Ω if and only if $(i,j) \in \Omega$, i.e., the entry $\mathbf{M}(i,j)$ is known. For example, if $\mathbf{M}$ is a 3×3 matrix and mask $\Omega = \{(1,1),(1,2),(1,3),(2,2),(3,1)\}$, then the mask graph that corresponds to Ω is shown in Figure 4·1. We will use the mask and its corresponding mask graph interchangeably since one specifies the other.

Genericity assumption: Any argument for matrix completion needs to make an assumption about how the underlying matrix $\mathbf{M}$ has been generated. There are several assumptions common in the literature, e.g., incoherence made by Candès and Recht (2012), non-spikyness in Negahban and Wainwright (2012), and *genericity* in Király et al. (2013), Király and Tomioka (2012b). Genericity is the most broad sampling assumption; it is based on algebraic concepts and a formal definition can be found in Király et al. (2013), Király and Tomioka (2012b). We note here that this assumption has important implications: for example, a generic matrix of rank r has no vanishing $(r \times r)$ minor, meaning any $(r \times r)$ submatrix of $\mathbf{M}$ has rank r.

3.2 Classes of completability

Posing the problem of matrix completion in terms of graph rigidity (reviewed in Appendix A.1) catalyzed structural matrix-completion which brought the ability to study the possible number of completions of a matrix. In practice, the cases of interested are those of a single completion or a finite number of completions. Seeking to evaluate the completability of a partially observed matrix, Singer and Cucuringu

(2010) and Király et al. (2013) introduced a necessary and sufficient condition for having a finite number of completions, called being *locally completable.* The set of masks that satisfy this condition can be thought of as the *class* of locally completable matrices, which we call $\mathbb{L}$. Similarly, the set of masks with only a single completion can be thought of as the class of uniquely (globally) completable matrices, which we call $\mathbb{U}$.

Unfortunately, to date there does not exist a condition that is both necessary and sufficient for checking membership in $\mathbb{U}$. Lack of such condition also means that there is no algorithm that takes as input a mask and returns "yes" if and only if the mask is completable. However, the literature by Király et al. (2013), Meka et al. (2009) and Singer and Cucuringu (2010) has presented three conditions that are sufficient for membership in $\mathbb{U}$, with which we define three subclasses: $\mathbb{SC}$, $\mathbb{CC}$, and $\mathbb{ICMC}$.

Each class can be associated with a *check routine* that takes as input a mask and checks the existing conditions to determine membership among the classes. In particular, the existence of a necessary and sufficient condition for local completion leads to a polynomial algorithm for determining whether or not a matrix is in $\mathbb{L}$. For $\mathbb{U}$ and the three subclasses, when a check routine outputs "yes," the input mask is in $\mathbb{U}$ – but if the check routine outputs "no," then no conclusion can be drawn.

In the rest of the chapter, we provide the first framework for studying the relationship between the classes of completability. We start by discussing the locally completable class $\mathbb{L}$ and the specific conditions associated with, and then do the same for $\mathbb{U}$ along with each of the three subclasses. Next, we theoretically and empirically study the relationship between the classes and their relative sizes. Together, these aspects make up the first study of the landscape of structural matrix-completion.

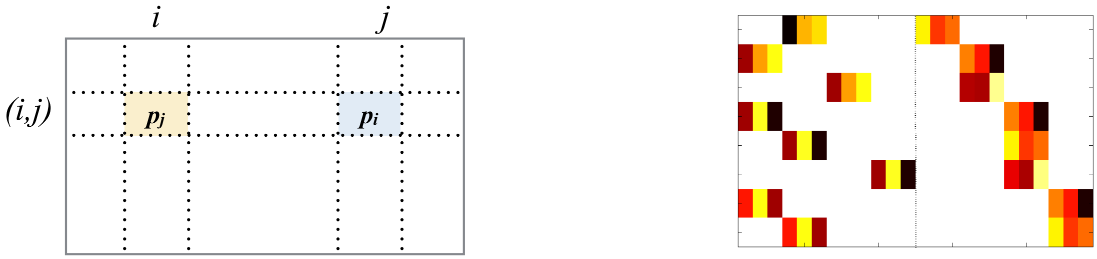
3.3 Locally completable masks

Recall that locally completable masks correspond to partially observed matrices with multiple possible completions, a finite number of them. In other words, there are a finite number of matrices that have the same observed values as $\mathbf{M}_\Omega$, but different values elsewhere.

Definition 1 (Locally-Completable Masks $\mathbb{L}$)**.** *Given a mask Ω, we say that Ω is locally completable, or belongs to class $\mathbb{L}$, if for any generic and partially-observed matrix $\mathbf{M}_\Omega$ there is a finite set of matrices $\mathcal{M}$ such that for each $\hat{\mathbf{M}} \in \mathcal{M}$ it holds that $\hat{\mathbf{M}}(\Omega) = \mathbf{M}(\Omega)$.*

Interestingly,both Király et al. (2013) and Singer and Cucuringu (2010) showed that checking if a mask is in $\mathbb{L}$ can be done in polynomial time via checking the rank of a special matrix called the *completion matrix* $\mathcal{C}_\Omega$.

The *completion matrix*: For a mask graph G_Ω with $n_v = n + m$ nodes and n_e edges, the completion matrix $\mathcal{C}_\Omega$ is an $n_e \times r(n + m)$ matrix where r corresponds to the rank of the matrix $\mathbf{M}$. To construct the completion matrix, first assign to each node i an r-dimensional random vector, and call this p_i. (We note that the set of these r-dimensional vectors $\{p_i | i \in V\}$ represents a random initialization of a configuration discussed in Appendix A.1.) Every node i in G_Ω is also associated with r columns in $\mathcal{C}_\Omega$. Each row of $\mathcal{C}_\Omega$ encodes an edge (i, j) in G_Ω and has only $2r$ nonzero entries: p_i in the r columns corresponding to node j and p_j in the r columns corresponding to node i. This is also visually illustrated in Figure 3·2. The left-hand side shows the construction, and the right-hand side shows a full example with white representing zero.



(a) Completion matrix

(b) Example for $r = 3$

Figure 3·2: The structure of the completion matrix.

The vectors p_i corresponding to rows can be aggregated into an $n \times r$ matrix, and the same can be done with the columns to form an $m \times r$ matrix; these two matrices $\mathbf{X}$ and $\mathbf{Y}$ can be viewed as the *factors* of $\mathbf{M}$. Király et al. (2013) demonstrate that at a high level, the completion matrix can be thought of as the Jacobian showing how the n_e visible entries of $\mathbf{M}_\Omega$ vary with respect to $\mathbf{M}$'s two factors. Note that there is a similarity between the completion matrix of a mask and the rigidity matrix for a graph as described by Roth (1981) and in Appendix A.1.

To see the high-level relationship between the completion matrix and the number of completions, consider a system of three equations in three variables. If the equations are independent, then the system is linearly independent, and there is a unique solution for the value of each variable. Alternatively, if there is dependency among the equations, then there is at least one variable that is unconstrained and can take infinitely many values, i.e. a free variable. Király et al. (2013) showed that the completion matrix encodes precisely this kind of relationship between the vectors (between the nodes). Intuitively, if there is not enough independence in the completion matrix $\mathcal{C}_\Omega$, then there are not enough constraints on the

Proposition 1 (Király et al. (2013), Singer and Cucuringu (2010))**.** *A mask* Ω *is locally completable if and only if the corresponding completion matrix* C_Ω *has rank* $\phi = r(n+m) - r^2$.

We refer to ϕ as the critical rank. Proposition 1 implies that checking whether a mask and the corresponding partially observed matrix $\mathbf{M}_\Omega$ are locally completable can be done simply by constructing and computing the rank of the completion matrix. Observe that if $\mathcal{C}_\Omega$ has rank ϕ it must also have ϕ independent rows; since each row corresponds to an edge, $\mathbf{M}_\Omega$ must have at least ϕ known entries. Hence having at least ϕ known entries is a necessary condition for $\mathbb{L}$ and not coincidentally, an $n \times m$ matrix of rank r has exactly ϕ degrees of freedom.

Using the condition in Proposition 1, we develop an check routine called `check-LC` that takes as input a mask and determines if the mask belongs to $\mathbb{L}$.

Algorithm 1 The `check-LC` algorithm.

```
Input $\Omega$, rank $r$
Construct the completion matrix $\mathcal{C}_\Omega$
if rank($\mathcal{C}_\Omega$)=$r(n+m-r)$ then output 'yes'
else output 'no'
end if
```

Because Proposition 1 gives a condition that is both necessary and sufficient, we can say that if Algorithm 1 outputs 'yes' then $\Omega \in \mathbb{L}$, and if it outputs 'no' then $\Omega \notin \mathbb{L}$. The latter implies that the mask does not have a finite number of completions but rather an infinite number or none.

3.4 Uniquely completable masks

Although there is a necessary and sufficient condition that can be used to check if a mask is locally completable, in practice, a unique completion is more desirable than a finite number of completions. While a finite number of completions can still result in a great deal of error which will lead to inaccuracies in the data analysis, a unique completion guarantees accuracy.

Definition 2 (Uniquely Completable Masks ($\mathbb{U}$))**.** *Given a mask Ω, we say that Ω is* uniquely completable*, or belongs to class $\mathbb{U}$, if for any generic partially-observed matrix $\mathbf{M}_\Omega$, there is a unique matrix $\hat{\mathbf{M}}$ such that $\hat{\mathbf{M}}(\Omega) = \mathbf{M}(\Omega)$.*

As we have already mentioned, there is no algorithm that takes as input a mask and returns "yes" if and only if the mask is uniquely completable. The following condition, among others, is shown to be necessary by Király et al. (2013).

Proposition 2. *To be uniquely completable, it must be that $|\Omega| \geq \phi = r(n+m) - r^2$.*

In addition, a number of sufficient conditions for a mask to be in $\mathbb{U}$ have been recently introduced by Király et al. (2013), Meka et al. (2009) and Singer and Cucuringu (2010). In these cases when the check routine outputs "yes," the input mask is in $\mathbb{U}$, but if the check routine outputs "no" then no conclusion can be drawn. We now discuss in detail each subclass of $\mathbb{U}$ and the corresponding check routines.

3.4.1 Closure ($\mathbb{CC}$):

The $\mathbb{CC}$ class is defined by all the masks that are "accepted" by a check routine described by Király et al. (2013).

Under the rank and genericity assumptions, every $r \times r$ minor in a rank r matrix $\mathbf{M}$ will be non-vanishing, i.e. the minor will have a nonzero determinant. In contrast, every minor of size $(r+i) \times (r+i)$ for $i > 0$ will vanish. Now consider the partially observed $\mathbf{M}_\Omega$ and an $(r+1) \times (r+1)$ minor with only one entry missing. The key observation is that since this minor is vanishing, its determinantal equation is equal to zero and has only one variable, the missing entry; hence, the missing entry can be recovered by solving the equation. Another way to view this is that the minor represents an $(r+1) \times (r+1)$ submatrix of rank-r. Hence, the row with the single missing entry can be written as a linear combination of the other r rows, and the missing entry can be recovered.

In the bipartite mask graph G_Ω, these vanishing minors with one entry missing take the form of $(r+1) \times (r+1)$-cliques that are missing one edge, which we denote with $K^-_{r+1,r+1}$. The proposed routine finds these almost-cliques $K^-_{r+1,r+1}$ in the graph, and adds the missing edge to each one at a time; representing the fact that the missing entry can be recovered. If in the end, all missing edges have been added (all entries have been recovered), then $\mathbf{M}_\Omega$ is uniquely completable at rank r. The details of the associated check routine are shown in Algorithm 2.

Algorithm 2 The `check-CC` algorithm.

```
Input $\Omega$, rank $r$
Construct the mask graph $G_\Omega = (V_1, V_2, E)$
while there is at least one $K^-_{r+1,r+1}$ do
    Find all $K^-_{r+1,r+1}$ in $G_\Omega$ and add each missing edge to $\Omega$
end while
if $G_\Omega$ is the complete graph then output 'yes'
else output 'no'
end if
```

Recall that since $\mathbb{CC}$ presents a condition which is merely sufficient, a ‘no’ answer implies that no conclusion can be drawn regarding membership in $\mathbb{CC}$. In cases when the output of `check-CC` is ‘no’, Király et al. (2013) describe extensions beyond $K^-_{r+1,r+1}$, for example, by finding two cliques $K^{-2}_{r+1,r+1}$ that overlap on the two missing entries. However, the approach becomes more complex and is outside the scope of this thesis.

Note that the running time of `check-CC` is exponential and is thus impractical for large datasets. Király et al. (2013) proposed a probabilistic version of the routine as a speed-up, but from our experiments, neither this improvement nor the one from our heuristics was worth the loss in accuracy.

3.4.2 Stress ($\mathbb{SC}$):

The $\mathbb{SC}$ class is defined by all the masks that are "accepted" by the `check-SC` routine first proposed by Singer and Cucuringu (2010) and later by Király et al. (2013)

The intuition behind this routine is that as a measure of how a structure (such as a bridge or building) reacts when a stress force is applied to it; does the structure stay rigid? Or does it move? In graph rigidity, this is measured using a *stress vector* $\vec{\rho}$ that lives in the left null space of the completion matrix $\mathcal{C}_\Omega$, i.e. each product of the $\vec{\rho}$ with a row of $\mathcal{C}_\Omega$ is zero. The vector $\vec{\rho}$ can be transformed into an $n \times m$ stress matrix $\mathcal{S}_\rho$ in a way such that the product of $\mathcal{S}_\rho$ with each factor $\mathbf{X}, \mathbf{Y}$ is zero (recall that $\mathbf{X}$ and $\mathbf{Y}$ are the collections of r-dimensional points as discussed in Section 3.3). The transformation is as follows: for row z of $\mathcal{C}_\Omega$ that corresponds to edge $(i, j) \in G_\Omega$, set $\mathcal{S}_\rho(i, j) = \rho(z)$. The procedure is described in the `Stressify` routine in Algorithm 4. Singer and Cucuringu (2010) and Király et al. (2013) show that the result of applying a stress on G_Ω can be captured with the rank of the stress matrix $\mathcal{S}_\rho$. If $\text{rank}(\mathcal{S}_\rho) = \min(n, m) - r$ then the stress is said to be resolved and the mask Ω is uniquely completable.

Algorithm 3 The `check-SC` algorithm.

```
Input $\Omega$, rank $r$
Construct $\mathcal{C}_\Omega$
Compute a random vector $\vec{\rho}$ in the left nullspace of $\mathcal{C}_\Omega$
$\mathcal{S}_\rho = \texttt{Stressify}(\mathcal{C}_\Omega, \vec{\rho})$
if $\text{rank}(\mathcal{S}_\rho) = \min(n, m) - r$ then output 'yes'
else output 'no'
end if
```

Observe that the `check-SC` algorithm runs in polynomial time, as it only needs to compute the left nullspace of $\mathcal{C}_\Omega$ and the rank of the stress matrix $\mathcal{S}_\rho$. Therefore, checking whether a mask is in $\mathbb{SC}$ is polynomial (albeit computationally demanding).

Algorithm 4 The `Stressify` algorithm.

```
Input $\mathcal{C}_\Omega, \vec{\rho}$
for row $z$ in $\mathcal{C}_\Omega$ do
    Find the edge $(i, j)$ that row $z$ corresponds to
    $\mathcal{S}_\rho(i, j) = \vec{\rho}(z)$
end for
Output $\mathcal{S}_\rho$
```

Proposition 3. *For a mask* Ω *to be in* $\mathbb{SC}$*, the corresponding mask matrix* $G_\Omega(V_1, V_2, E)$ *must have at least* $r(|V_1| + |V_2|) - r^2 + 1$ *edges.*

Proof. The proof of the proposition relies on the fact that the stress matrix S_ρ cannot be constructed unless the completion matrix $\mathcal{C}_\Omega$ has a left nullspace. Since any mask in $\mathbb{SC}$ must have $r(|V_1| + |V_2|) - r^2$ independent rows (because of its containment in $\mathbb{L}$), the corresponding mask graph must have at least $r(|V_1| + |V_2|) - r^2 + 1$ edges. This can be confirmed by referencing the relationship between the row space, null space, and rank of a matrix. □

Király *et al.* have conjectured the following: "…it seems that masks not in $\mathbb{SC}$ are rare among the uniquely completable masks and correspond to pathological cases". However, we will show in Section 3.5 that our experiments indicate that this is not the case.

3.4.3 Information cascade ($\mathbb{ICMC}$):

Unlike the classes $\mathbb{CC}$ and $\mathbb{SC}$ which are specified as a condition and suggest an algorithm, the class discussed here is specified as an algorithm and suggests a condition.

The $\mathbb{ICMC}$ class and the corresponding `check-ICMC` routine are based on the connection to information propagation introduce by Meka et al. (2009). Similar to the `check-CC` routine, the underlying idea is that the presence of independence can be tied to the completability. However, instead of searching for minors, `check-ICMC` utilizes the factors, $\mathbf{X}$ and $\mathbf{Y}$, of $\mathbf{M}$. Intuitively Meka et al. (2009) show that each row of the factors can be written as the solution x of an $r \times r$ linear system of equations $\mathbf{A}x = b$ where b is composed of the entries of $\mathbf{M}_\Omega$. If the mask is such that for each row of the factors there exists a b composed solely of known entries and no unknowns, then each factor can be uniquely recovered; hence the mask is in $\mathbb{U}$. Meka et al. (2009) provide a more intuitive interpretation of this condition through the lens of an epidemic propagation on the bipartite mask graph G_Ω which we discuss below.

The algorithm starts by considering all possible subsets L_0 of r initial nodes (which, without loss of generality, are subsets of V_1). For any initial subset, the r nodes are labeled as *infected.* Infection subsequently propagates in the iterative fashion from one side of the mask graph to the other. A node becomes infected after at least r of its neighbors are infected. If there exists an L_0 such that all nodes can be eventually infected, then `check-ICMC` returns "yes", and the input mask Ω is in $\mathbb{ICMC}$ and $\mathbb{U}$.

Since `check-ICMC` checks all subsets of V_1 of size r, it has exponential running time. A more efficient variant of `check-ICMC`, only checks a single L_0 composed of the r highest-degree nodes in G_Ω. Meka *et al.* have shown if the mask-graph follows a power-law distribution, then with high probability this variant returns 'yes' whenever the exhaustive one does. The running time of this variation is linear to the cardinality of the mask Ω and thus much more practical. Details are shown in Algorithm 5.

The infection-propagation view of `check-ICMC` can also be adopted for `check-CC`. This view reveals a high-level similarity between the two algorithms: In `check-ICMC` a node gets infected if it is connected to at least r other infected nodes, while in `check-CC` a node gets infected if it is part of a $K^-_{r+1,r+1}$ subgraph.

Algorithm 5 The `check-ICMC` algorithm

```
Input $\Omega$, rank $r$
Construct the mask graph $G_\Omega = (V_1, V_2, E)$
Set $L_0$ to the $r$ nodes in $V_1$ with highest degree
Mark all nodes in $L_1 = L_0$ as infected and set $L_2 = \emptyset$
while $L_1 \neq V_1$ and $L_2 \neq V_2$ do
    Find nodes $\ell_2 \subseteq \{V_2 \setminus L_2\}$ that are connected to $\geq r$ infected nodes in $V_1$.
    Mark nodes in $\ell_2$ as infected and add to $L_2 = L_2 \cup \ell_2$
    Find nodes $\ell_1 \subseteq \{V_1 \setminus L_1\}$ that are connected to $\geq r$ infected nodes in $V_2$.
    Mark nodes in $L_1$ as infected and add to $L_1 = L_1 \cup \ell_1$
    if $\ell_1$ and $\ell_2$ are empty then return 'no'
    end if
end while
Return 'yes'
```

3.5 Categorization

We started by defining $\mathbb{L}$ as the class of matrices with a finite number of completions. While necessary and sufficient condition exists for checking membership in $\mathbb{L}$, the most compelling case in practice is unique completion since it results in zero error in theory. This case is represented by the $\mathbb{U}$ class. Although there do not exist strong conditions like for $\mathbb{L}$, we have demonstrated the three sufficient conditions introduced to date: $\mathbb{SC}$, $\mathbb{CC}$, and $\mathbb{ICMC}$, each of which defines a subclass of $\mathbb{U}$. Given the importance of $\mathbb{U}$, a natural question to ask is whether there is a single subclass that can act as a good proxy for $\mathbb{U}$. In other words is there a condition that is satisfied by almost all masks in $\mathbb{U}$? On the same note, we can wonder about the relationship between the subclasses. Are they distinct? Perhaps they all specify the same set of matrices?

To date, there has been no investigation into these relationships. We start by providing some basic proven relationships between the classes. Then we investigate the remainder through MCMC-style experiments on random masks.

3.5.1 Theoretical exploration

First, we note that by definition and using results from Király et al. (2013), we have the following:

Proposition 4 (Király et al. (2013))**.** *For $r > 1$, the class $\mathbb{U}$ is a strict subset of the class $\mathbb{L}$. That is, $\mathbb{U} \subset \mathbb{L}$.*

The above means that any mask that is uniquely completable is also locally completable. In fact, for the special case where the rank of $\mathbf{M}$ is $r = 1$, the following can be shown (Proposition 2.15 in Király and Tomioka (2012a)):

Corollary 5. *For masks of matrices of rank $r = 1$ we have that $\mathbb{L} = \mathbb{U}$.*

An immediate corollary of Proposition 4 is the the containment of the subclasses.

Corollary 6. *$\mathbb{ICMC} \subseteq \mathbb{L}$, $\mathbb{CC} \subseteq \mathbb{L}$ and $\mathbb{SC} \subseteq \mathbb{L}$.*

For the class $\mathbb{CC}$, we can prove something more strict: $\mathbb{CC}$ is a strict subset of $\mathbb{U}$.

Proposition 7. *The following relationship holds: $\mathbb{CC} \subset \mathbb{U}$.*

Proof of Proposition 7. The proof is constructed by providing an example of a mask that is in $\mathbb{U}$ but not in $\mathbb{CC}$, as in Király et al. (2013). Consider Figure 3·3 provided by Király et al. (2013). The first thing we observed is that with $r = 3$ this mask does not belong to $\mathbb{CC}$ because there is not a single $K^{-}_{4,4}$, i.e. there is not 4×4 submatrix with only one missing entry.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Figure 3·3: An example of a mask that is uniquely completable at $r = 3$ but does not satisfy the `check-CC` condition (for Lemma 7).

On the other hand, the mask is uniquely completable! To see this, consider the two 4×4 submatrices that overlap on the two zeros in the middle; namely the submatrix S_1 formed by rows 1-4 and columns 1-4, and the submatrix S_2 formed by rows three to six and columns three to six. First show that the middle two zeros can be recovered uniquely. Recall that the $\mathbb{CC}$ technique to recover entries uniquely is writing the equation of the determinant with the single variable being the missing entry; however, writing the determinant for each of S_1 gives an equation in two variables. Nevertheless, since the missing entries are in the same column, the two variables will have a linear relationship in the determinantal equation. Since the same applies for S_2, we obtain two equations that are linear in two variables; hence the variables have a unique solution.

Once these two zeros are flipped to ones, it is easy to find a sequence of $K^-_{4,4}$ for each remaining zero. In the end, since all entries have been recovered uniquely, the mask is in $\mathbb{U}$.

□

A similar result can be shown regarding $\mathbb{CC}$ and $\mathbb{ICMC}$.

Proposition 8. *The following relationship holds:* $\mathbb{ICMC} \subset \mathbb{CC}$.

To prove this lemma we first show that an mask in $\mathbb{ICMC}$ is also in $\mathbb{CC}$ and then give an example of a mask that is in $\mathbb{CC}$ but not in $\mathbb{ICMC}$. First we fix some terminology. We say that an element (i, j) of Ω is *ICMC-recovered* if and only if nodes v_i and v_j in G_Ω, corresponding respectively to row i and column j of M_Ω, have been infected by the `check-ICMC` routine. Similarly, an element (i, j) is *CC-recovered* if it is part of a $K^-_{r+1,r+1}$-clique in $\mathbf{M}_\Omega$.

Lemma 9. *Given an* $\mathbb{ICMC}$ *completable mask* Ω*, any ICMC-recovered entry is also a CC-recovered entry.*

Proof of Lemma 9. **Base Step:** Since Ω is ICMC-completable, M_Ω must contain an $r - clique$ (in order to spread the infection G_Ω must contain r nodes that are each connected to all r nodes in L_0). Now we re-order the rows and columns of M_Ω so that this r-clique is in the top-left of M_Ω (i.e. the infected rows and columns appear first), and call them R_0 and C_0. This is illustrated in Figure 3.5.1 for $r = 3$ – note that $\mathbf{M}_\Omega(R_0, C_0) = 1$.

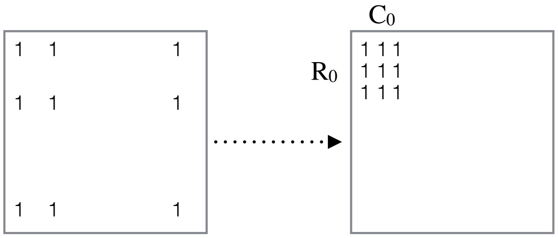


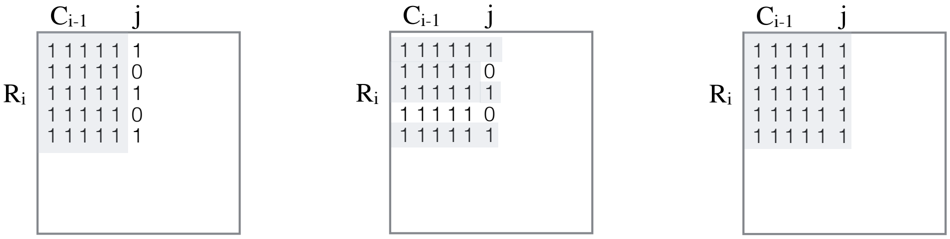
Figure 3·4: Example of re-ordering of rows and columns.

This re-ordering is maintained at every step of the proof, so that the infected rows and columns appear first.

Inductive Step: In each step i, `check-ICMC` will infect either a set of columns or rows. Consider the current state where R_i rows and C_{i-1} columns are infected. By the assumption that `check-ICMC` returns "yes", there must be at least one column to infect, call it j, which means this column must have at least r ones in $\mathbf{M}_\Omega(R_i, j)$.

We establish two facts about every remaining zero (z, w) in $\mathbf{M}_\Omega(R_i, j)$. First, there are at least r ones in $\mathbf{M}_\Omega(z, C_{i-1})$ and second, there are at least r ones in $\mathbf{M}_\Omega(R_i, w)$. Further, the base step ensured that $\mathbf{M}_\Omega(R_i, C_{i-1})$ is a block of ones at least size $r \times r$. This is depicted in 3·5a.

Hence, we can claim that every 0 in column j is part of a $K^-_{r+1,r+1}$-clique which means that every 0 is CC-recovered; shown in Figure 3·5b for the zero at $(2, j)$. Since the column j becomes infected all 0 are flipped to 1s and are ICMC-recovered. The logic above is symmetric for an infected row.



(a) New column j (b) $K^-_{r+1,r+1}$ for 0 at $(2, j)$ (c) All entries recovered

Figure 3·5: Every entry that is ICMC-recovered is also CC-recovered.

Given the starting assumption that Ω passes the `check-ICMC`, all the rows and columns of M_Ω are eventually infected as shown in Figure 3·5c. Since we know that any zero of a newly infected row or column is CC-recovered, we know that any entry that is ICMC-recovered is also CC-recovered. In other words any mask Ω that passes `check-ICMC`, also passes `check-CC` $\square$

We proceed to prove the strict containment $\mathbb{ICMC} \subset \mathbb{CC}$ by providing an example of a matrix in $\mathbb{CC}$ but not in $\mathbb{ICMC}$.

Proof of Proposition 8. By Lemma 9 we have that $\mathbb{ICMC} \subseteq \mathbb{CC}$.

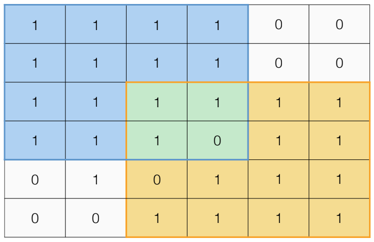
We describe a generative process to construct a mask which will belong to $\mathbb{CC}$ but not $\mathbb{ICMC}$. Any M_Ω that exhibits the following properties will pass the `check-CC` but not `check-ICMC`.

1. M_Ω is of size $(r+3) \times (r+3)$
2. There are two $(r+1)$-cliques (blocks of ones) with an overlap of size $(r-1) \times (r-1)$
3. One entry in the overlap is 0
4. An additional 0 is in one of the cliques but outside of the overlap
5. A single 1 outside of both cliques in the same row or column as the 0 from (4)

For intuition consider the example in Figure 3·6 with $r = 3$. The blue and yellow 4×4 blocks of 1s (cliques) and there is a single 0 in the green overlap. There is an additional 0 in position $(5, 3)$ outside the overlap but in the yellow clique. Finally, outside of both the blue and the yellow cliques there is a single 1 in the same row as the latter 0, i.e. in position $(5, 2)$.

We claim that there is a sequence of $K^{-}_{4,4}$-cliques such that all the 0s switch to 1s, starting from the zero entry in the green overlap in position $(4, 4)$, followed by the zero in postion $(5, 2)$, and so on. The 1 from property (5) is important for the `check-CC` to continue past the first two 0s. Hence, the mask passes the `check-CC`.

However, regardless of the starting point `check-ICMC` does not ouput 'yes'. There are three possible starting points: one 3×3 blocks of 1s in the blue clique, and two in the yellow. The blue clique will infect the rows and columns of the green 0, but nothing past that. Both yellow clique will not manage to infect even a single additional row or columns.



1	1	1	1	0	0
1	1	1	1	0	0
1	1	1	1	1	1
1	1	1	0	1	1
0	1	0	1	1	1
0	0	1	1	1	1

Figure 3·6: Example of a mask in $\mathbb{CC}$ but not in $\mathbb{ICMC}$ for $r = 3$.

Any mask with the structure described above will be in $\mathbb{CC}$ but not in $\mathbb{ICMC}$, hence we $\mathbb{ICMC}$ is a strict subset of $\mathbb{CC}$.

□

Though we have established a containment relationship between $\mathbb{L}$, $\mathbb{CC}$, and $\mathbb{ICMC}$, but the relationship with the $\mathbb{SC}$ class is not so clear.

Proposition 10. *The following relationship holds* $\mathbb{CC} \not\subset \mathbb{SC}$.

Proof of Proposition 10. Consider the mask below at rank $r = 1$.

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

While the mask is in $\mathbb{CC}$, it can be shown that the rank of the stress matrix is $1 \neq \min(m, n) - r = 2$, hence the mask is not in $\mathbb{SC}$.

□

The remaining relationships are unclear, for example, it may be that $\mathbb{SC} \subset \mathbb{CC}$ or that there is no containment relationship between the two. A visual of our results and stipulates is shown in Figure 3·7.

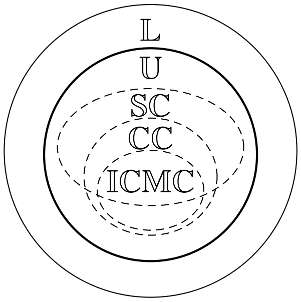


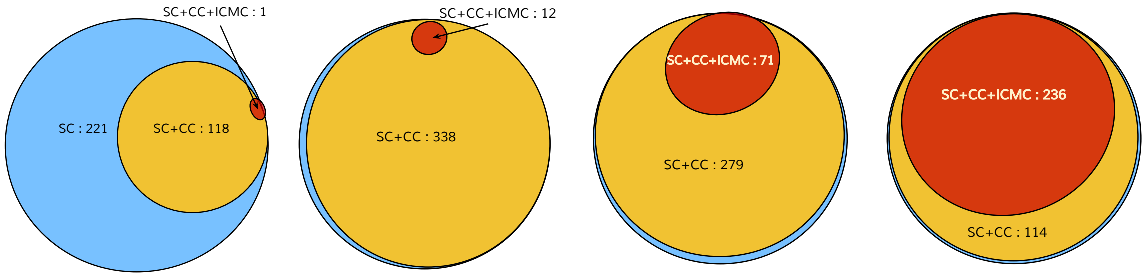
Figure 3·7: Containment of completability classes.

3.5.2 Empirical exploration

The existing literature allowed us to pose several lemmas regarding the relationship between matrix-completion classes. To get an idea of the remaining relationships that we cannot reason about with theory, we conduct an empirical exploration. Our goal is to gain intuition about the likelihood of random masks to be members of one or more of the subclasses. Through this, we get a sense of the relative sizes and relationships between the classes.

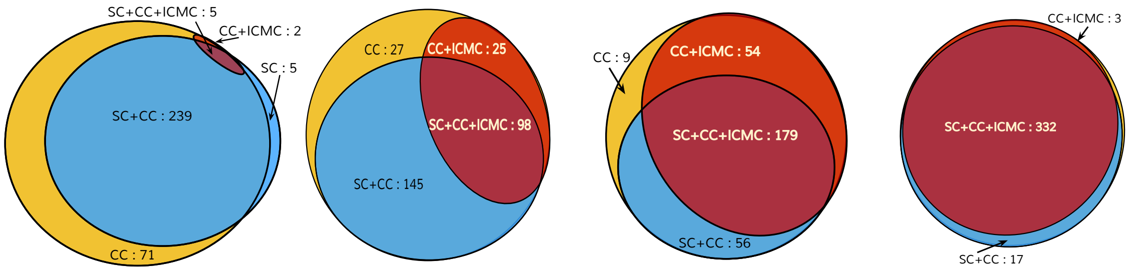
We study two important types of masks: *Random* masks, in which visible entries occur uniformly at random with some given density; and *PowerLaw* masks, in which the distribution of visible entries is not uniform, but rather follows a power-law distribution with respect to the number of visible entries in each row and column. *Random* masks have been commonly studied, especially in the signal processing literature. *PowerLaw* masks have been argued to more accurately model the sampling patterns seen in real data, for example, by Meka et al. (2009). For each type of graph, we study a range of densities of observed values.

Specifically, 350 masks of each type were generated at each density. For a given density (i.e., number of visible entries), positions of visible entries were chosen uniformly at random in the *Random* case, or using Guillaume and Latapy (2006) in the *PowerLaw* case (power-law exponent of 2.5;). All masks were of size 200 × 300. For



(a) 0.08 (b) 0.1 (c) 0.12 (d) 0.14

Figure 3·8: *Random* masks with density = 0.08, 0.10, 0.12, 0.14.



(a) 0.035 (b) 0.04 (c) 0.045 (d) 0.05

Figure 3·9: *PowerLaw* masks with density = 0.035, 0.040, 0.045, 0.050.

both mask types, densities were varied over the critical range where membership in the subclasses changed most significantly.

The results are shown in figures 3·8 and 3·9. Each class is shown as an ellipse: $\mathbb{CC}$ is yellow; $\mathbb{SC}$ is blue; and $\mathbb{ICMC}$ is red. The figures have been drawn so that the areas of the ellipses and their intersections are proportional to the number of masks that fall into each. Thus one can judge how often a mask falls into each subclass from the areas shown in the figure.

The figures reveal a number of interesting features. First, observed that as we conjectured with Figures 3·7, there are masks that belong to $\mathbb{SC}$ and not $\mathbb{CC}$, meaning there is no containment relationship between the two. The most significant and consistent effect concerns the relative sizes of the classes as the density of visible entries varies. Specifically, at low densities, many masks fall into $\mathbb{SC}$, and very few fall into $\mathbb{ICMC}$; the latter shows that empirically speaking, $\mathbb{ICMC}$ is the 'hardest' subclass to fall into randomly.

Apart from the difficulty of falling into $\mathbb{ICMC}$, a number of other properties are visible. First, the properties of $\mathbb{CC}$ vary with the graph type: for *Random* masks at low density, $\mathbb{CC}$ is relatively difficult to fall into, while for *PowerLaw* masks, $\mathbb{CC}$ is the easiest class to fall into. Another property concerns the relationship between $\mathbb{SC}$ and $\mathbb{CC}$: For *PowerLaw* masks, $\mathbb{CC}$ is generally larger than $\mathbb{SC}$, while this never true for random masks. This suggests that neither $\mathbb{SC}$ nor $\mathbb{CC}$ can be close in size to $\mathbb{U}$ in general. In other words, none of the three subclasses can be consistently good approximations to $\mathbb{U}$. This is unfortunate, because as has been discussed, $\mathbb{U}$ is the class of fundamental interest for unique matrix-completion.

Chapter 4

Active structural-completion

In the previous chapters, we have discussed the problem of matrix completion and setup a distinction between structural and statistical methods. While statistical methods optimize for the best-fit solution assuming certain conditions hold, structural methods analyze the information content to quantify the possibility of completion. Both approaches rely on a having at least a certain number of observations.

In practice, datasets do not have enough known entries to meet the lower bound prescribed by statistical methods, which limits the possibility of completion and translates to estimates that may differ significantly from the true matrix. For example, adopting the rank $r \approx 40$ of the top solutions to the Netflix Challenge, the Netflix data is 151 million entries below the lower bound.

In many cases, it is possible to address the insufficiency of $\mathbf{M}_\Omega$ by actively obtaining additional observations. For example, in recommender systems, users may be asked to rate several items; in traffic analysis, new monitoring points may be installed. These additional observations can lead to an augmented Ω' such that $\mathbf{M}_{\Omega'}$ carries more information about $\mathbf{M}$ and may result in more accurate estimates $\hat{\mathbf{M}}$. In this *active* setting, the data analyst can become an *active* participant in data collection by posing *queries* to $\mathbf{M}$. Of course, such active involvement will only be acceptable if the number of queries is small.

In this chapter, we present a method for generating a small number of queries so as to ensure that the combination of observed and queried values provides sufficient information for an accurate completion; i.e., the $\hat{\mathbf{M}}$ estimated using the entries $\mathbf{M}_{\Omega'}$ is significantly better than the one estimated using $\mathbf{M}_{\Omega}$. We call the problem of generating a small number of queries that guarantee small reconstruction error the ActiveCompletion problem.

The difference between the classical matrix-completion problem and our problem is that in the former, the set of observed entries is *fixed* and the algorithm needs to find the best completion given these entries. In ActiveCompletion, we are asked to design both a *completion* and a *querying* strategy in order to minimize the reconstruction error. On the one hand, this task is more complex than standard matrix completion – since we have the additional job of designing a good querying strategy. On the other hand, having the flexibility to ask some additional entries of $\mathbf{M}$ to be revealed should yield lower reconstruction error. At a high level, ActiveCompletion is related to other recently proposed methods for active matrix completion discussed in Section 2. However, existing approaches identify entries to be queried *independently* of the method of completion. In contrast, a strength of our algorithm is that it addresses completion and querying in an integrated fashion.

The main contribution of this chapter is `Order&Extend`, an algorithm that simultaneously minimizes the number of queries to ask and produces an estimate matrix $\hat{\mathbf{M}}$ that is very close to the true matrix $\mathbf{M}$. We build upon the methods of structural matrix-completion to analyze the amount of information in the observed entries, and then guide the selection of queries.

The design of `Order&Extend` is inspired by the view of structural matrix-completion as a linear system of equations. Based on this, we ask which queries need to be added so that the partially observed matrix admits a unique completion. We go one step further and observe that there is a relationship between the ordering in which systems are solved, and the number of additional queries that need to be posed. Therefore, the first step of `Order&Extend` focuses on finding a good ordering of the set of linear systems, relying only on the combinatorial properties of the mask graph. Since there is no necessary and sufficient condition for membership in $\mathbb{U}$; we target the $\mathbb{ICMC}$ subclass but discuss techniques for entering other subclasses.

In the second step, `Order&Extend` considers the linear systems in the chosen order, and asks queries every time it encounters a *problematic* linear system $\mathbf{A}x = b$. A linear system can problematic in two ways: (a) when there are not enough equations for the number of unknowns, so that the system does not have a unique solution; (b) when solving the system $\mathbf{A}x = b$ is numerically unstable *given the specific b involved.* Note that, as we explain in the paper, this is not the same as simply saying that $\mathbf{A}$ is ill-conditioned; part of our contribution is the design of fast methods for detecting and ameliorating such systems.

Our experimental analysis with datasets from a variety of application domains (presented in Ruchansky et al. (2015)) demonstrates that `Order&Extend` requires significantly fewer queries than any other baseline querying strategy, and compares very favorably to approaches based on well-known matrix completion algorithms. In fact, our experiments indicate that `Order&Extend` is "almost optimal" as it gives solutions where the number of entries it queries is very close to the information-theoretic lower bound for completion.

4.1 Notation and assumptions

Throughout this chapter we use $\mathbf{M}$ to refer to an $n \times m$ fully-known matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ of rank r, and we use to denote the i-th row or column as $\mathbf{M}(i,:)$ and $\mathbf{M}(:,i)$ or $\mathbf{M}_i$ when it is clear from the context. We use $\mathbf{M}_{ij}$ the entry (i,j). Any rank-r matrix can be written as the product of two factors which we refer to as $\mathbf{M} = \mathbf{XY}$ where $\mathbf{X}$ is $n \times r$ and $\mathbf{Y}$ is $r \times m$. A row of $\mathbf{X}$ is denoted $\mathbf{X_i}$ and a column of $\mathbf{Y}$ as $\mathbf{Y_i}$ and

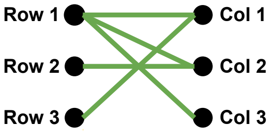


Figure 4·1: The mask graph G_Ω of mask $\Omega = \{(1,1),\ (1,2),\ (1,3),\ (2,2),\ (3,1)\}$.

an entry (i,j) as $x_{i,j}$.

The number of degrees of freedom in the matrix is called $\theta = r(n+m-r)$. If $\mathbf{M}$ is not completely known we use $\Omega \subseteq \{(i,j) | 1 \leq i \leq n, 1 \leq j \leq m\}$ to denote the subset of entries that are observed; we call this set of known entries the *mask of* $\mathbf{M}$. When a matrix completion algorithm is applied to the partially observed $\mathbf{M}_\Omega$, we use $\hat{\mathbf{M}}$ to denote the output estimate.

Each mask is also associate with a bipartite *mask graph* $G_\Omega = (V_1, V_2, E)$. Every node $i \in V_1$ represents a row of $\mathbf{M}$ and every $j \in V_2$ represents a column. An edges $(i,j) \in E$ exist in G_Ω if and only if $(i,j) \in \Omega$, i.e., the entry $\mathbf{M}(i,j)$ is known. For example, if $\mathbf{M}$ is a 3×3 matrix and mask $\Omega = \{(1,1),(1,2),(1,3),(2,2),(3,1)\}$, then the mask graph that corresponds to Ω is shown in Figure 4·1.

To measure the error of a completion we use the relative Frobenius error:

$$\texttt{RelErr}(\hat{\mathbf{M}}) = \frac{\|\mathbf{M} - \hat{\mathbf{M}}\|_f^2}{\|\mathbf{M}\|_f^2} \tag{4.1}$$

Rank assumption: The assumption that the true rank of a matrix is known is fundamental in obtaining many of the theoretical results in this work – as well as most papers in the matrix-completion literature Candès and Recht (2012), Király and Tomioka (2012b), Király et al. (2013), Negahban and Wainwright (2012).

A standard assumption is that the matrix $\mathbf{M}$ has *low effective rank* r ($\ll \min(n,m)$). Were $\mathbf{M}$ exactly rank r, it would have r non-zero singular values; when $\mathbf{M}$ is *effectively* rank r, it has full rank but its top r singular values are significantly larger in magnitude than the rest. In practice, many matrices obtained through empirical measurements are found to have low effective rank.

In practice, most matrices obtained through empirical measurements (even those that are fully observed) are full rank, because of the presence of noise, measurement error, and minor confounding factors. However it is often the case that real datasets have small *low effective rank* r ($\ll \min(n, m)$). Were $\mathbf{M}$ exactly rank r, it would have r non-zero singular values; when $\mathbf{M}$ is *effectively* rank r, it has full rank but its top r singular values are significantly larger in magnitude than the rest.

The assumption of effective low-rank is used in obtaining many theoretical results in the matrix-completion literature Candès and Recht (2012), Király et al. (2013), Király and Tomioka (2012b), Negahban and Wainwright (2012). Yet in practice, the important assumption is that of low effective rank. Even if r is unknown but required as input to an algorithm, one could try several values of r and choose the best performing one. For the rest of the thesis, we consider r to be known and omit reference to it when it is understood from context.

Degeneracy assumption: A version of the genericity assumption (see Section 3.1) that was specified by Meka et al. (2009) is called degeneracy. An $n \times m$ rank r matrix $\mathbf{M}$ is non-degenerate if there exist two matrices $\mathbf{X}\in \mathbb{R}^{n\times r}$ and $\mathbf{Y}\in \mathbb{R}^{k\times m}$ with $\mathbf{M} = \mathbf{XY}$ such that any r rows of $\mathbf{X}$ are linearly independent. and any r columns of $\mathbf{Y}$ are linearly independent.

4.2 The problem of active completion

For simplicity, we discuss our problem and algorithms in the context of unlimited access to all unobserved entries of $\mathbf{M}$. However, our results still apply, and our algorithm can still work in the presence of constraints on which entries of $\mathbf{M}$ may be queried. We define our problem as follows:

Problem 1 (ActiveCompletion)**.** *Given an integer $r > 0$ that corresponds to the effective rank of $\mathbf{M}$ and the values of $\mathbf{M}$ in certain positions Ω, find a set of additional entries Q to query from $\mathbf{M}$ such that for $\Omega' = \Omega \cup Q$,* RelError$(\hat{\mathbf{M}}_{\Omega'})$ *as well as $|Q|$ are minimized.*

Note that the above problem definition has two minimization objectives: (1) the number of queried entries and (2) the reconstruction error. In practice, we can only solve for one and impose a constraint on the other. For example, we can impose a *query budget b* on the number of queries to ask and optimize for the error. Alternatively, one can use *error budget ϵ* to control the error of the output, and then minimize the number of queries to ask. In principle, our algorithm can be adjusted to solve any of the two cases. However, since setting a desired b is more intuitive for our active setting, in our experiments we do this and optimize for the error. We will focus on this version of the problem (with the budget on the queries) for the majority of the discussion.

The exact case: A special case of ActiveCompletion is when $\mathbf{M}$ is exactly rank r, and the maximum allowed error ϵ is zero. In this case, the problem asks for the minimum number of queries required to reconstruct $\hat{\mathbf{M}}$ that is exactly equal to the true matrix $\mathbf{M}$. This problem is an instance of checking whether the mask of $\mathbf{M}_\Omega$ is uniquely completable, which we have established to rely only on Ω and not on the actual values $\mathbf{M}_\Omega$. In this special case, the goal of `Order&Extend` is to entries to bring the mask into $\mathbb{U}$ by entering the subclass $\mathbb{ICMC}$.

Critical mask size: The number of degrees of freedom of an $n \times m$ matrix of rank exactly r is $r(n + m - r)$, which we denote ϕ. Hence, regardless of the nature of Ω, any solution with $\epsilon = 0$ must have $|\Omega'| \geq \phi$. Therefore we call ϕ the *critical mask size* as it can be considered as a (rather strict) *lower bound* on the number of entries that need to be in Ω' to achieve small reconstruction error.

Empty masks: For the special case of exact rank and $\epsilon = 0$, if the input mask Ω

is empty, i.e., $\Omega = \emptyset$, then ActiveCompletion can be solved optimally as follows: simply query the entries of r rows and r columns of $\mathbf{M}$. This optimal solution will require ϕ queries, which will construct a mask Ω' that determines a unique reconstruction of $\mathbf{M}$. Therefore, when the initial mask $\Omega = \emptyset$, the ActiveCompletion problem can be solved in polynomial time.

Non-empty masks: In practice, the input mask Ω is not empty. In fact according to Proposition 2 in the previous chapter, there may be cases where the input mask Ω contains ϕ entries, yet Ω is not uniquely completable. As an example, consider the following partially-observed rank-2 matrix:

$$\mathbf{M}_\Omega = \begin{bmatrix} & 8 & 4 & 7 \\ 21 & & 7 & 11 \\ 29 & 20 & & 15 \\ 42 & 28 & 14 & \end{bmatrix}.$$

In this case the mask Ω has exactly $\phi = r(n+m-r) = 12$ observed entries, yet in fact there are (exactly) *two* possible reconstructions of this matrix at rank 2. Both reconstructions agree on the off-diagonal elements while having different values in their diagonal elements. The two possible settings for diagonal elements that yield rank-2 versions of $\mathbf{M}$ are $diag(\mathbf{M}) = (13, 14, 10, 22)$ and $diag(\mathbf{M}) = (12, 14\frac{10}{23}, 9\frac{2}{3}, 24\frac{1}{2})$. This example shows that not only the number but also the position of the visible entries is important in determining unique completion.

4.3 The `Order&Extend` algorithm

In this section, we present our algorithm, `Order&Extend`, for addressing the ActiveCompletion problem.

The starting point for the design of `Order&Extend` is the low (effective) rank assumption of $\mathbf{M}$ and the approach of structural matrix-completion. As it will become clear, this means that the unobserved entries are related to the observed entries through a set of linear systems. Thus one approach to matrix completion is to solve a sequence of linear systems. Each system in this sequence uses observed entries in $\mathbf{M}$ or entries of $\mathbf{M}$ reconstructed by previously solved linear systems to infer more missing entries.

The reconstruction error of such an algorithm depends on the quality of the solutions to these linear systems. As we will show below, each query of $\mathbf{M}$ can yield a new equation that can be added to a linear system. Hence, if a linear system has fewer equations than unknowns, a query must be made to add a new equation to the system. Likewise, if solving a system is numerically unstable, then a query must be made to add an equation that will stabilize it. Crucially, the need for such queries depends on the nature of the solutions obtained to linear systems *earlier in the order*. Thus the order in which systems are solved, and the nature of these systems are inter-related. A good ordering will minimize the number of "problematic" systems encountered. However, problematic systems can appear even in the best possible order, meaning that good ordering alone is insufficient for accurate reconstruction.

At a high level, `Order&Extend` operates as follows: first, it finds a promising ordering of the linear systems. Then, it proceeds by solving the linear systems in this order. If a linear system that requires additional information is encountered, the algorithm either strategically queries $\mathbf{M}$ or moves the system to the end of the ordering. When all systems have been solved, $\hat{\mathbf{M}}$ is computed and returned. The next subsections describe these steps in detail.

4.3.1 Completion as a sequence of linear systems

In this section, we explain the particular linear systems that the completion algorithm solves, the sequence in which it solves them, and how the ordering in which systems are solved affects the quality of the completion. For the purposes of this discussion, we assume that $\mathbf{M}$ is of rank exactly r. In this case $\mathbf{M}$ can be expressed as the product

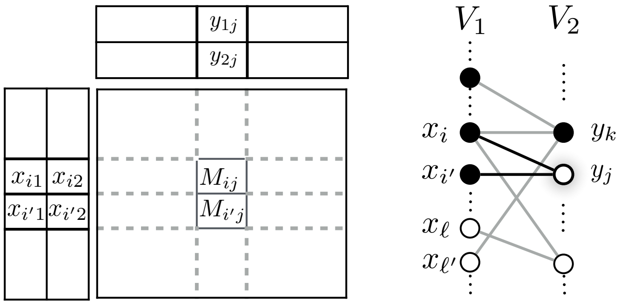


Figure 4·2: An intermediate step of `Sequential` algorithm.

of two matrices $\mathbf{X}$ and $\mathbf{Y}$ of sizes $n \times r$ and $r \times m$; that is, $\mathbf{M} = \mathbf{XY}$. Furthermore, we assume that $\mathbf{M}$ is nondegenerate, meaning any subset of r rows of $\mathbf{X}$, or r columns of $\mathbf{Y}$, is linearly independent. (Later we will describe how `Order&Extend` addresses the case when these assumptions do not hold – i.e., when $\mathbf{M}$ is only *effectively* rank-r, or when an r-subset is linearly dependent). To complete $\mathbf{M}$, it suffices to find such factors $\mathbf{X}$ and $\mathbf{Y}$.[1].

The `Sequential` completion algorithm: We start by describing an algorithm we call `Sequential`, which estimates the rows of $\mathbf{X}$ and columns of $\mathbf{Y}$. `Sequential` takes two inputs: (1) an ordering π over the set of all rows of $\mathbf{X}$ and columns of $\mathbf{Y}$, which we call the *reconstruction order*, and (2) the partially observed matrix $\mathbf{M}_\Omega$.

To explain how `Sequential` works, consider the example in Figure 4·2, where $r = 2$, $\mathbf{M}$ is on the left and G_Ω is on the right. The factors $\mathbf{X}$ and $\mathbf{Y}$ are shown on the side of and above $\mathbf{M}$ to convey how their product results in $\mathbf{M}$. The nodes V_1 of G_Ω correspond to the rows of $\mathbf{X}$, and nodes V_2 to the columns of $\mathbf{Y}$. In this figure, we illustrate an intermediate step of `Sequential`, in which the values of the i-th and i'-th rows of $\mathbf{X}$ have already been computed. Each entry of $\mathbf{M}$ is the inner product of a row of $\mathbf{X}$ and a column of $\mathbf{Y}$.

---

[1] Note that $\mathbf{X}$ and $\mathbf{Y}$ are not uniquely determined; any invertible $r \times r$ matrix $\mathbf{W}$ yields new factors $\mathbf{XW}^{-1}$ and $\mathbf{WY}$ which also multiply to yield $\mathbf{M}$

Hence we can represent the depicted entries in $\mathbf{M}$ by the following linear system:

$$\mathbf{M}_{ij} = x_{i1}y_{1j} + x_{i2}y_{2j} \tag{4.2}$$

$$\mathbf{M}_{i'j} = x_{i'1}y_{1j} + x_{i'2}y_{2j} \tag{4.3}$$

Observe that x_i and $x_{i'}$ are known, and that the edges (x_i, y_j) and $(x_{i'}, y_j)$ corresponding to $\mathbf{M}_{ij}$ and $\mathbf{M}_{i'j}$ exist in G_Ω. The only unknowns in (4.2) and (4.3) are y_{1j} and y_{2j}, which leaves us with two equations in two unknowns. As stated above and by assumption, any r-subset of $\mathbf{X}$ or $\mathbf{Y}$ is linearly independent; hence one can solve uniquely for y_{1j} and y_{2j} and fill in column j of $\mathbf{Y}$.

To generalize the example above, the steps of `Sequential` can be partitioned in x- and y-steps; at every y-step the algorithm solves a system of the form

$$A_x y = t. \tag{4.4}$$

In this system, y is a vector of r unknowns corresponding to the values of the column of $\mathbf{Y}$ we are going to compute; A_x is an $r \times r$ fully-known submatrix of $\mathbf{X}$ and t is a vector of r known entries of $\mathbf{M}$ which are located on the same column as the column index of y. If A_x and t are known, and A_x is full rank, then y can be computed exactly and the algorithm can proceed to the next step.

In the x-steps `Sequential` evaluates a row of $\mathbf{X}$ using an $r \times r$ already-computed subset of columns of $\mathbf{Y}$, and a set of r entries of $\mathbf{M}$ from the row of $\mathbf{M}$ corresponding to the current row of $\mathbf{X}$ being solved for. Following the same notational conventions as above, the corresponding system becomes $A_y x = t'$. For simplicity we will focus our discussion on y-steps; the discussion on x-steps is symmetric.

The completion on the mask graph: The execution of `Sequential` is also captured in the mask graph shown in the right part of Figure 4·2. In the beginning, no rows or columns have been recovered, and all nodes of G_Ω are white (unknown). As the algorithm proceeds they become black (known), and this transformation occurs in the order suggested by the input reconstruction order π. Thus a black node denotes a row of $\mathbf{X}$ or column of $\mathbf{Y}$ that has been computed. In our example, the fact that we can solve for the j-th column of $\mathbf{Y}$ (using Equations (4.2) and (4.3)) is captured by y_j's *two* connections to black/known nodes (recall $r = 2$). For general rank r, the j-th column of $\mathbf{Y}$ can be estimated by a linear system if in the mask graph, y_j is connected to at least r already computed (black) nodes. This is symmetric for the i-th row of $\mathbf{X}$ and node x_i. Intuitively, this transformation of nodes from black to white is reminiscent of an information propagation process. This procedure is inspired by the `check-ICMC` algorithm and the connections first drawn by Meka et al. Meka et al. (2009).

Incomplete and unstable linear systems: As it has already been discussed in the literature by Meka et al. (2009), the performance of an algorithm like `Sequential` is heavily dependent on the input reconstruction order. Meka et al. (2009) have discussed methods for finding a good reconstruction order in the special case where the mask graph has a power-law degree distribution. However, even with the best possible reconstruction order `Sequential` may still encounter linear systems which are either *incomplete* or *unstable*. Incomplete linear systems are those for which the vector t has some missing values and therefore the system $A_x y = t$ cannot be solved. Unstable linear systems are those in which all the entries in t are known, but the resulting expression $A_x^{-1} t$ may be very sensitive to small changes in t. These systems raise numerous problems in the case where the input $\mathbf{M}$ is a noisy version of a rank r matrix, i.e., it is a matrix of effective rank r.

In the next two sections we describe how `Order&Extend` deals with such systems.

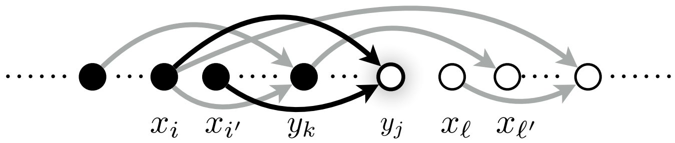


Figure 4·3: The direction on the edges imposed by the ordering of nodes shown in Figure 4·2.

4.3.2 Ordering and fixing incomplete systems

First, `Order&Extend` devises an order that minimizes the number of incomplete systems encountered in the completion process.

Let us consider again the execution of `Sequential` on the mask graph, and the sequential transformation of the nodes in $G_\Omega = (V_1, V_2, E)$ from white to black. Recall that in this setting, an incomplete system occurs when the node in G_Ω that corresponds to y is connected to less than r black nodes.

Consider an order π of the nodes $V_1 \cup V_2$. This order conceptually imposes a direction on the edges of E; if x is before y in that order, then $\pi(x) < \pi(y)$, and edge (x, y) becomes directed edge $(x \rightarrow y)$. Figure 4·3 shows this transformation for the mask graph in Figure 4·2 and the order implied there. For fixed π, a node becomes black if it has at least r incoming edges, i.e., indegree at least r. In this view, an incomplete system manifests itself by the existence of a node that has indegree less than r. Clearly, if an order π guarantees that all nodes have r incoming edges, then there are no incomplete systems, and π is a *perfect reconstruction order*.

In practice such perfect orders are very hard to find; in most of the cases, they do not exist. The goal of the first step of `Order&Extend` is to find an order π that is as close as possible to a perfect reconstruction order. It does so by constructing an order that minimizes the number of edges that need to be added so that the indegree of any node is r.

To achieve this, the algorithm starts by choosing the node u from $G_\Omega = (V_1, V_2, E)$

with the lowest degree. This node is placed last in π (meaning $\pi(u)$ is large), and removed from $G_\Omega = (V_1, V_2, E)$ along with its incident edges. Of the remaining nodes, the one with the minimum degree is placed in the next-to-last position in π and again removed from $G_\Omega = (V_1, V_2, E)$. This process repeats until all nodes have been assigned a position in π.

Next, the algorithm makes an important set of adjustments to π by examining each node u in the order it occurs in π. For a particular u the adjustments can take two forms:

1. *if u has degree $\leq r$*: it is repositioned to appear immediately after the neighbor v with the largest $\pi(v)$.
2. *if u has degree $> r$*: it is repositioned to appear immediately after the neighbor its neighbor v with the the r-th smallest $\pi(v)$.

These adjustments aim to construct a π such that when the implied directionality is added to edges, each node has indegree as close to r as possible. While it is possible to iterate this adjustment process to further improve the ordering, in our experiments, this showed little benefit.

Once the order π is formed as described above, then the incomplete systems can be quickly identified: as `Order&Extend` traverses the nodes of G_Ω in the order implied by π, every time it encounters a node u with in-degree less than r, it adds edges so that u's indegree becomes r; by definition, the addition of a new edge (x, u) corresponds to querying a missing entry $\mathbf{M}_{xu}$ of $\mathbf{M}$.

4.3.3 Finding and alleviating unstable systems

Incomplete systems are easy to identify – they correspond to nodes in G_Ω with degree less than r. However, there are other "problematic" systems which do not appear to be incomplete, yet they are *unstable*. Such systems arise due to noise in the data matrix or to an accumulation of error that happens through the sequential system-solving process. These systems are harder to detect and alleviate. We discuss our methodology for this below.

Understanding unstable linear systems: Recall that a system $A_x y = t$ is unstable if its solution is very sensitive to the noise in t. To be more specific, consider the system $A_x y = t$, where A_x has full rank and t is fully known. Recall that the solution of this system, $y = A_x^{-1} t$, will be used as part of a subsequent system: $A_y x = t'$, where y will become a row of matrix A_y. Let $A_x = U \Sigma V^T$ be the singular value decomposition of A_x with singular values $\sigma_1 \geq \ldots \geq \sigma_r$. Now if there is a σ_j such that σ_j is very small, then the solution to the linear system will be very unstable when the singular vector v_j corresponding to σ_j has a large projection on t. This is because in A_x^{-1}, the small σ_j will be inverted to a very large $1/\sigma_j$. Thus the inverse operation will cause any component of t that is in the direction of v_j to be *disproportionally-strongly expressed*, and any small amount of noise in t to be amplified in y. Thus, unstable systems may be catastrophic for the reconstruction error of `Sequential` as a single such system may initiate a sequence of unstable systems, which can amplify the overall error.

Unstable vs ill-conditioned systems: It is important to contrast the notion of an unstable system with that of an ill-conditioned system, which is widely used in the literature. Recall, that system $A_x y = t$ is ill-conditioned if *there exists* a vector s and a small perturbation s' of s, such that the results of systems $A_x y = s$ and $A_x y' = s'$ are significantly different. Thus, whether or not a system is ill-conditioned depends only on A_x, and not on its relationship with any target vector t in particular. An ill-conditioned system is also characterized by a large condition number $\kappa(A_x) = \frac{\sigma_1}{\sigma_n}$. This way of stating ill-conditioning emphasizes that $\kappa(A_x)$ measures a property of A_x and *does not depend* on t. Consequently (as we will document in Section 4.4.2) the condition number $\kappa(A_x)$ generates too many false positives to be used for identifying unstable systems.

Identifying unstable systems: To provide a more precise measure of whether a system $A_x y = t$ is unstable, we compute the following quantity:

$$\ell(A_x, t) = \|A_x^{-1}\| \frac{\|t\|}{\|y\|}. \tag{4.5}$$

We call this quantity the *local condition number*, which was also discussed by Trefethen and Bau Trefethen and Bau III (1997). The local condition number is more tailored to our goal as we want to quantify the proneness of a system to error with respect to a particular target vector t. In our experiments, we characterize a system $A_x y = t$ as unstable if $\ell(A_x, t) \geq \theta$. We call the threshold θ the *stability threshold* and in our experiments we use $\theta = 1$. Loosely, one can think of this threshold as a way to control for the error allowed in the entries of reconstructed matrix. Although it is related, the value of this parameter does not directly translate into a bound on the RelError of the overall reconstruction.

Selecting queries to alleviate unstable systems: One could think of dealing with an unstable system via regularization, such as ridge regression (Tikhonov Regularization) which was also suggested by Meka et al. (2009). However, for systems $A_x y = t$, such regularization techniques aim to dampen the contribution of the singular vector that corresponds to the smallest singular value, as opposed to boosting the contribution of the singular vectors that are in the direction of t. Further, the procedure can be expressed in terms of only A_x without taking t into account; as we have discussed this is not a good measure for our approach.

The advantage of our setting is that we can actively query entries from $\mathbf{M}$. Therefore, our way of dealing with this problem is by adding a direction to A_x (or as many as are needed until there are r strong ones). We do that by extending our system from $A_x y = t$ to $\begin{bmatrix} A_x \\ \alpha \end{bmatrix} \tilde{y} = \begin{bmatrix} t \\ \tau \end{bmatrix}$. Of course, in doing so we implicitly shift from looking for an exact solution to the system, to looking for a least-squares solution.

Clearly α cannot be an arbitrary vector. It must be an already computed row of $\mathbf{X}$, it should be independent of A_x, and it must boost a direction in A_x which is poorly expressed and also in the direction of t. Given the intuition we developed above, we iterate over all previously computed rows of $\mathbf{X}$ that are not in A_x, and set each row as a candidate α. Among all such α's we pick α^* as the one with the smallest $\ell(A_x, t)$, and use it to extend A_x to $\begin{bmatrix} A_x \\ \alpha^* \end{bmatrix}$.

Algorithm 6 The `local_condition` routine

```
Input: $C, A_x, \alpha, t$
$D = C - \frac{C\alpha^T \alpha C}{1 + \alpha C \alpha^T}$
$\tilde{A}_x = \begin{bmatrix} A_x \\ \alpha \end{bmatrix}$
$\tau = \texttt{Random}\,(\mathbf{M}(i,:), \mathbf{M}(:,j))$
$\tilde{t} = \begin{bmatrix} t \\ \tau \end{bmatrix}$
$\tilde{y} = D\tilde{A}_x \tilde{t}$
return $\|D\tilde{A}_x\| \frac{\|\tilde{t}\|}{\|\tilde{y}\|}$
```

Algorithm 7 The `Stabilize` routine

```
Input: $A_x, t, \theta$
$j$ : the column of $\mathbf{Y}$ being computed
$C = (A_x^T A_x)^{-1}$
for $i \in \{$Computed rows of X$\}$ do
    $\alpha_i = \mathbf{X_i}$
    if $\mathbf{X_i}$ not in $A_x$ then
        $c(i) = \texttt{local\_condition}(C, A_x, \alpha_i, t, \tau)$
    end if
end for
$i^* = \arg\min_i c(i)$, $\alpha^* = \mathbf{X_{i^*}}$
if $c(i^*) < \theta$ then
    return $(i^*, j), \alpha^*$
end if
return null
```

Querying M judiciously: Although the above procedure is conceptually clear, it raises a number of practical issues. If the system $A_x y = t$ solves for the j-th column of matrix $\mathbf{Y}$, then every time we try a different α, which suppose is the already computed $\mathbf{X_i}$, then the corresponding τ must be the entry $\mathbf{M}_{ij}$. Since $\mathbf{M}_{ij}$ is not necessarily an observed entry, this would require a query even for rows $\alpha \neq \alpha^*$, which is clearly a waste of queries since we will only pick one α^*. Therefore, instead of querying the unobserved values of τ, `Order&Extend` simply uses random values following the distribution of the values observed in the i-th row and j-th column of $\mathbf{M}$. Once α^* is identified, we only query the value of τ corresponding to row α^* and column j.

If there is no α^* that leads to a system with local condition number below our threshold, we postpone solving this system by moving the corresponding node of the mask graph to the end of the order π.

Computational speedups: From the computational point of view, the above approach requires computing a matrix inversion per α. With a cubic algorithm for matrix inversion, this could induce significant computational cost. However, we observe that this can be done efficiently as all the matrix inversions we need to perform are for matrices that differ only in their last row – the one occupied by α.

Recall that the least-squares solution of the system $A_x y = t$ is $y = (A_x^T A_x)^{-1} A_x^T t$. Now in the extended system $\begin{bmatrix} A_x \\ \alpha \end{bmatrix} \tilde{y} = \begin{bmatrix} t \\ \tau \end{bmatrix}$ or $\tilde{A}_x \tilde{y} = \tilde{t}$, the corresponding solution is $\tilde{y} = (\tilde{A}_x{}^T \tilde{A}_x)^{-1} \tilde{A}_x{}^T \tilde{t}$. Observe that we can write:

$$\tilde{A}_x{}^T \tilde{A}_x = \begin{bmatrix} A_x^T & \alpha^T \end{bmatrix} \begin{bmatrix} A \\ \alpha \end{bmatrix} = A_x^T A_x + \alpha^T \alpha.$$

Thus, $\tilde{A}_x{}^T \tilde{A}_x$ can be seen as a rank-one update to $A_x^T A_x$. In such a setting the Sherman-Morrison Formula Golub and Loan (2012) provides a way to efficiently calculate $D = (\tilde{A}_x{}^T \tilde{A}_x)^{-1}$ given $C = (A_x^T A_x)^{-1}$. The details are shown in Algorithm 6.

Using the Sherman-Morrison Formula we can find $\tilde{y}$ via matrix multiplication, which requires $O(r^2)$ for at most $n = \max\{n, m\}$ candidate queries. Since the values of r we encounter in real datasets are small constants (in the range of 5-40), this running time is small.

The pseudocode of this process is shown in Algorithm 7. The process of selecting the right entry to query is summarized in the `Stabilize` routine. Observe that `Stabilize` either returns the entry to be queried, or if there is no entry that can lead to a stable systems it returns null. In the latter case the system is moved to the end of the order.

4.3.4 Putting everything together

Given all the steps we described above we are now ready to summarize `Order&Extend` in Algorithm 8.

`Order&Extend` constructs the rows of $\mathbf{X}$ and columns of $\mathbf{Y}$ in the order prescribed by π – the pseudocode shows the construction of columns of $\mathbf{Y}$, but it is symmetric for the rows of $\mathbf{X}$. For every linear system the algorithm encounters, it completes the system if it is incomplete and tries to make it stable if it is unstable. When a complete and stable version of the system is found, the system is solved using least squares. Otherwise, it is moved to the end of π.

Initialization: The algorithm starts with empty factors $\mathbf{X}$ and $\mathbf{Y}$ and no infected nodes. To begin the infection, the algorithm must start from an initial set of r nodes and call them infected; the initial set is chosen as the first r nodes in π from a single side of the bipartite graph. This initial set corresponds to r rows of $\mathbf{X}$ (or columns of $\mathbf{Y}$) whose values are set and who will be used as A_x in the first system. The values in these first r rows are arbitrary as long as they form r independent rows. This is because, as noted in footnote-1, the factors are not unique, hence any invertible $\mathbf{W}$ can account for the scaling and rotation $\mathbf{M} = \mathbf{XY} = (\mathbf{X'W^{-1}})(\mathbf{WY'})$. In our

Algorithm 8 The `Order&Extend` algorithm

```
Input: $\mathbf{M}_\Omega, r, \theta$
Compute $G_\Omega$
Find ordering $\pi$ (as per Section 4.3.2)
Select the first $r$ nodes in $\pi$ as the initial set $L_0$
Set rows $L_0$ of $\mathbf{X}$ to $I_r$ the $r \times r$ identity
for $A_x y = t$ (corresponding to the $j$-th column of $\mathbf{Y}$) encountered in $\pi$ do
    solve_system = $true$
    if $A_x y = t$ is incomplete then
        Query $\mathbf{M}$ and complete $A_x y = t$
    end if
    while local_condition($A_x, t$) $> \theta$ do
        if $\{(i^*, j), \alpha^*\} =$ Stabilize($A_x, t, \theta$) $\neq$ null then
            $A_x = \begin{bmatrix} A_x \\ \alpha^* \end{bmatrix}$
            $t = \begin{bmatrix} t \\ \mathbf{M}_{i^* j} \end{bmatrix}$
        else
            move $A_x y = t$ to the end of $\pi$
            solve_system = $false$
            break
        end if
    end while
    if solve_system then
        $\mathbf{Y_j} = y = A_x^\dagger t$ (using least squares)
    end if
end for
return $\hat{\mathbf{M}} = \mathbf{XY}$
```

experiments we set the initial r rows of $\mathbf{X}$ (according to π) to the $r \times r$ identity matrix.

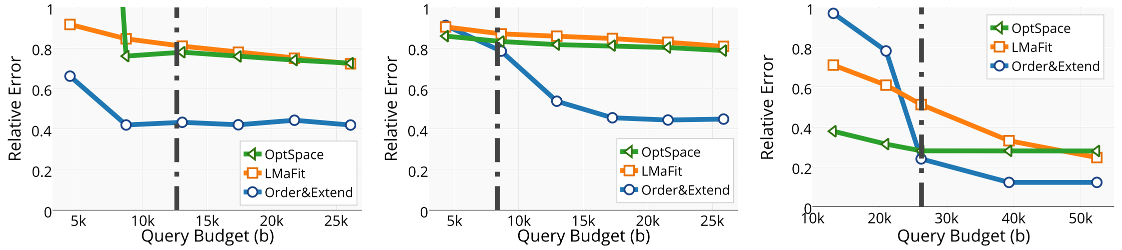
Running time: The running time of `Order&Extend` consists of the time to obtain the initial ordering, which using the algorithm of Matula and Beck Matula and Beck (1983) is $O(n+m)$, plus the time to detect and alleviate incomplete and unstable systems. Recall that for each unstable system we compute an inverse $O(r^3)$ and check n candidates $O(r^3 + r^2 n)$. Thus the overall running time of our algorithm is $O((n+m) + N \times (r^3 + r^2 n))$, where N is the number of unstable system the algorithm encounters. In practice, the closer a matrix is to being of rank exactly r, the smaller the number of error-prone systems it encounters and therefore the faster its execution time.[2]

Partial completions: If the budget b of allowed queries is not adequate to resolve the incomplete or the unstable systems, then `Order&Extend` will output $\hat{\mathbf{M}}$ with only a portion of the entries completed. The entries that remain unrecovered are those for which the algorithm claims inability to produce a good estimate. From the practical viewpoint, this is extremely useful information as the algorithm is able to inform the data analyst which entries it was not able to reconstruct from the observations in $\mathbf{M}_\Omega$. It also means the algorithm can adapt to privacy constraints or restrictions on which missing entries can be queried.

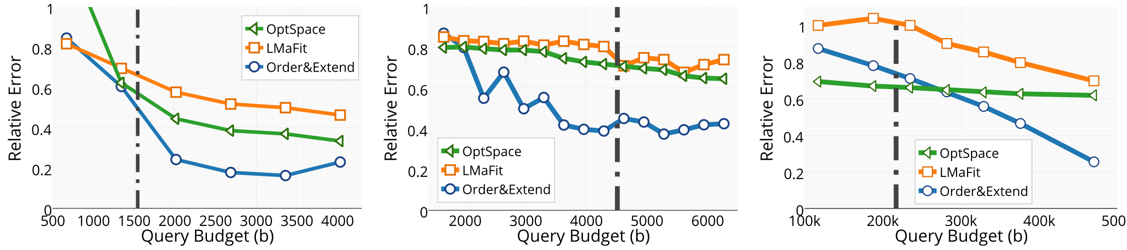
Dynamic Setting: The infection propagation view of matrix completion enables `Order&Extend` to extend naturally to dynamic settings where the data is received as a stream. Once a completion has been calculated using `Order&Extend` on a partially observed matrix, it does not need to be update because of the completability guarantee When a new row or column arrives, a new node is added to the graph, the infectability is evaluated, and edges added if required.

---

[2] Code and information are available at `http://cs-people.bu.edu/natalir/matrixComp`



(a) *Traffic1* (b) *Traffic2* (c) *Boat*



(d) *Latency1* (e) *Latency2* (f) *Jester*

Figure 4·4: RelError of completion achieved by `Order&Extend`, `LMaFit` and `OptSpace` on datasets with *approximate* rank; x-axis: query budget b; y-axis: RelError of the completion.

4.4 Experimental results

In this section, we experimentally evaluate the performance of `Order&Extend` both in terms of the reconstruction error as well as the number of queries it makes. Our experiments show that across all datasets `Order&Extend` requires very few queries to achieve a very low reconstruction error. All other baselines we compare against require many more queries for the same level of error, or can ever achieve the same level of reconstruction error.

Datasets: We experiment on the following nine real-world datasets, taken from a variety of applications.

MovieLens: This dataset contains ratings of users for movies from the MovieLens website.[3] The original dataset has size $6\,040 \times 3\,952$ and only 5% of its entries are observed. For our experiments we obtain a denser matrix of size $4\,832 \times 3\,162$.

---

[3] Source `http://www.grouplens.org/node/73`.

Netflix: This dataset also contains user movie-ratings, but from the Netflix website. The original size is $480\,189 \times 17\,770$ with 1% of observed entries. Again we focus on a submatrix with higher percentage of observe entries and size $48\,019 \times 8\,885$.

Jester: This dataset corresponds to a collection of user joke ratings obtained for joke recommendation on the Jester website.[4] For our experiments we use the whole dataset with size $23\,500 \times 100$ with 72% of its entries being observed.

Boat: This dataset is a fully-observed black and white image of size 512×512.

Traffic: This is a set of four datasets; each is part of a traffic matrix from a large Internet Service Provider where rows and the columns are source and destination prefixes (i.e., groups of IP addresses), and each entry is the volume of traffic between the corresponding source-destination pair. The largest dataset size $7\,371 \times 7\,430$ and 0.1% of its entries are observed; we call this *TrafficSparse*. The other two are fully-observed of sizes $2\,016 \times 107$, and $2\,016 \times 121$; we call these *Traffic1* and *Traffic2*.[5]

Latency: Here we use two datasets consisting of Internet network delay measurements. Rows and columns are hosts, and each entry indicates the minimum ping delay among a particular time window. The datasets are fully-observed and of sizes 116×116, and 869×19; we call these *Latency1* and *Latency2*.[5]

Baseline algorithms: We compare the performance of our algorithm to two state-of-the-art matrix-completion algorithms, `OptSpace` and `LMaFit` .

As neither `OptSpace` nor `LMaFit` are algorithms for active completion, we set up our experiment as follows: first, we run `Order&Extend` on $\mathbf{M}_{\Omega_0}$, which asks a budget of b queries. Before feeding $\mathbf{M}_{\Omega_0}$ to `LMaFit` and `OptSpace` we extend it with b *randomly* chosen queries. In this way, both algorithms query the same number of additional entries. A random distribution of observed entries has been proved to be (asymptotically) optimal for statistical methods like `OptSpace` and `LMaFit`

---

[4] Source `http://goldberg.berkeley.edu/jester-data/`

[5] Source `https://www.cs.bu.edu/~crovella/links.html`

by Candès and Recht (2012), Keshavan et al. (2010a), Wen et al. (2012). Therefore, picking randomly distributed b additional entries is the best querying strategy for these algorithms, and we have also verified that experimentally.

4.4.1 Methodology

For all our experiments, the ground-truth matrix $\mathbf{M}$ is known but not fully revealed to the algorithms. The input to the algorithms consists of an *initial mask* Ω_0, the observed matrix $\mathbf{M}_{\Omega_0}$, and a budget b on the number of queries they can ask. Each algorithm $\mathcal{A}$ outputs an estimate $\hat{\mathbf{M}}_{\mathcal{A},\Omega_0}$ of $\mathbf{M}$.

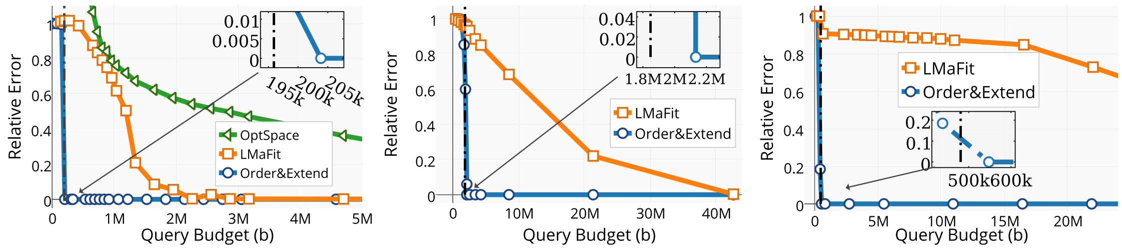
Selecting the input mask Ω_0: The initial mask Ω_0, with cardinality m_0, is selected by picking m_0 entries uniformly at random from the ground-truth matrix $\mathbf{M}$.[6]. The cardinality m_0 is selected so that $m_0 > 0$ and $m_0 < \phi(\mathbf{M}, r)$; usually we chose m_0 to be $\approx 30 - 50\%$ of $\phi(\mathbf{M}, r)$. The former constraint guarantees that the input is not trivial while the latter guarantees that additional queries are needed.

Range for the query budget b: We vary the number of queries, b, that an algorithm can issue among a wide range of values. Starting with $b < \phi(\mathbf{M}, r) - m_0$, we gradually increase it until we see that the performance of our algorithms stabilizes (i.e., further queries do not decrease the reconstruction error). Clearly, the smaller the value of b the larger the reconstruction error of the algorithms.

Reconstruction error: Given a ground-truth matrix $\mathbf{M}$ and input $\mathbf{M}_{\Omega_0}$, we evaluate the performance of a reconstruction algorithm $\mathcal{A}$, by computing the relative error of $\hat{\mathbf{M}}_{\mathcal{A},\Omega_0}$ with respect to $\mathbf{M}$, using the RelError function defined in Equation (5.1). This measure takes into consideration *all* entries of $\mathbf{M}$, both the observed and the unobserved. The closer $\hat{\mathbf{M}}_{\mathcal{A},\Omega}$ is to $\mathbf{M}$ the smaller the value of $\textsc{RelError}(\hat{\mathbf{M}}_{\mathcal{A},\Omega})$. In general, $\textsc{RelError}(\hat{\mathbf{M}}_{\mathcal{A},\Omega}) \in [0, \infty)$ and at perfect recon-

---

[6] We also test other sampling distributions, but the results are the same as the ones we report here and thus omitted



(a) *MovieLens* (b) *Netflix* (c) *TrafficSparse*

Figure 4·5: RelError of completion achieved by `Order&Extend`, `LMaFit` and `OptSpace` on datasets with *exact* rank; x-axis: query budget b; y-axis: RelError of the completion.

struction $\text{RelError}(\hat{\mathbf{M}}_{\mathcal{A},\Omega}) = 0$.

Although our baseline algorithms always produce a full estimate (i.e., they estimate all missing entries), `Order&Extend` may produce only partial completions (see Section 4.3.4 for a discussion in this). In these cases, we assign value 0 to the entries it does not estimate.

4.4.2 Evaluating `Order&Extend`

Experiments with real noisy data: For our first experiment, we use datasets for which we know all off the entries. This is true for six out of our nine datasets: *Traffic1* ,*Traffic2*, *Latency1*, *Latency2*, *Jester*, *Boat*. Note that *Jester* is missing 30% of the entries, but we treat them as true zero-values ratings; the remaining datasets are fully known and able to be queried as needed. As these are real datasets they are not exactly low rank, but plotting their singular values reveals that they have low effective rank. By inspecting their singular values, we chose: $r = 7$ for the *Traffic* and *Latency* datasets, $r = 10$ for *Jester* and $r = 40$ for *Boat*.

Figure 4·4 shows the results for each dataset. The x-axis is the query budget b; note that while `LMaFit` and `OptSpace` always exhaust this budget, for `Order&Extend` it is only an upper bound on the number of queries made. The y-axis is the

$\textsc{RelError}(T, \hat{\mathbf{M}}_{\mathcal{A},\Omega})$. The vertical black line marks the number of queries needed to reach the critical mask size; i.e., it corresponds to a budget of $(\phi(\mathbf{M}, r) - m_0)$. One should interpret this line as a very conservative lower bound on the number of queries that an optimal algorithm would need to achieve error-less reconstruction in the absence of noise.

From the figure, we observe that `Order&Extend` exhibits the lowest reconstruction error across all datasets. Moreover, it does so with a very small number of queries, compared to `LMaFit` and `OptSpace`; the latter algorithms achieve errors of approximately the same magnitude in all datasets. On some datasets `LMaFit` and `OptSpace` come close to the relative error of `Order&Extend` though with significantly more queries. For example for the *Latency1* dataset, `Order&Extend` achieves error of 0.24 with $b = 2K$ queries; `LMaFit` needs $b = 4K$ to exhibit an error of 0.33, which is still more than that of `Order&Extend`. In most datasets, the differences are even more pronounced; e.g., for *Traffic2*, `Order&Extend` achieves a relative error of 0.50 with about $b = 13K$ queries; `OptSpace` and `LMaFit` achieve error of more than 0.8 even after $b = 26K$ queries. Such large differences between `Order&Extend` and the baselines appear in all datasets, but *Boat*. For that dataset, `Order&Extend` is still better, but not as significantly as in other cases – likely an indication that the dataset is noisier. We also point out that the value of b for which the relative error of `Order&Extend` exhibits a significant drop is much closer to the indicated lower bound by the vertical black line. Again this phenomenon is not so evident for *Boat* probably because this dataset is further away from being low rank.

Extremely sparse real-world data: For the purpose of experimentation our algorithm needs to have access to all the entries of the ground truth matrix $\mathbf{M}$ – in order to be able to reveal the values of the queried entries. Unfortunately, the *Movielens*, *Netflix*, and *TrafficSparse* datasets consist mostly of missing entries, therefore we cannot query the majority of them. To be able to experiment with these datasets, we overcome this issue by approximating each dataset with its closest rank r matrix T_r. The approximation is obtained by first assigning 0 to all missing entries of the observed $\mathbf{M}$, and then taking the singular value decomposition and setting all but the largest r singular values to zero. This trick grants us the ability to study the special case discussed in Section 4.2 where the matrix is of exact rank r.

Using $r = 40$, the results for these datasets are depicted in Figure 4·5 with the same axes and vertical line as in Figure 4·4. Again, we observe a clear dominance of `Order&Extend`. In this case, the differences in the relative error it achieves are much more striking. Moreover,`Order&Extend` achieves almost 0 relative error for an extremely small number of queries b; in fact, the error of `Order&Extend` consistently drops to an extremely small value for b very close to the lower bound of the optimal algorithm (as marked by the black vertical line shown in the plot). On the other hand `LMaFit` and `OptSpace` are far from exhibiting such behavior. This signals that `Order&Extend` devises a querying strategy that is almost optimal. Interestingly the performance of `OptSpace` changes dramatically in these cases as compared to the approximate rank datasets. In fact on *TrafficSparse* and *Netflix* the error is so high it does not appear on the plot.

Note that the striking superiority of `Order&Extend` in the case of exact-rank is consistent across all datasets we considered, including others not shown here.

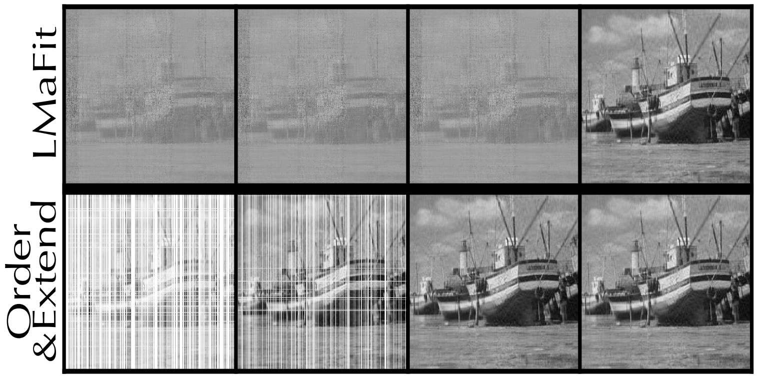


Figure 4·6: Recovery process using `LMaFit`, and `Order&Extend`. Each column is a particular b, increasing from left to right.

Running times: Though the algorithmic composition is quite different, we give some indicative running times for our algorithm as well as `LMaFit` and `OptSpace`. For example, in the *Netflix* dataset the running times were in the order of 11 000 seconds for `LMaFit`, 80 000 seconds for `Order&Extend`, and 200 000 seconds for `OptSpace`. These numbers indicate that `Order&Extend` is efficient despite the fact that in addition to matrix completion it also identifies the right queries; the running times of `LMaFit` and `OptSpace` simply correspond to running a single completion on the extended mask that is randomly formed. Note that these running times are computed using an unoptimized and serial implementation of our algorithm; improvements can be achieved easily e.g., by parallelizing the local condition number computations.

Partial completion of `Order&Extend`: As a final experiment, we provide anecdotal evidence that demonstrates the difference in the philosophy behind `Order&Extend` and other completion algorithms. Figure 4·6 provides a visual comparison of the recovery process of `Order&Extend` and `LMaFit` for different values of query budget b. For small values of b, `Order&Extend` does not have the sufficient information to resolve all incomplete and unstable systems. Therefore the algorithm does not estimate the entries of $\mathbf{M}$ corresponding to these systems, which renders the white areas in the two left-most images of `Order&Extend` . In contrast `LMaFit` outputs full estimates, though with significant error which can be seen by the incremental sharpening of the image, compared to the piece-by-piece reconstruction of `Order&Extend`.

Discussion Here we discuss some alternatives we have experimented with but omitted due to significantly poorer performance.

Alternative querying strategies: `Order&Extend` uses a rather intricate strategy for choosing its queries to $\mathbf{M}$. A natural question is whether a simpler strategy would be sufficient. To address this, we experimented with versions of `Sequential` that considered the same order as `Order&Extend` but when stuck with a problematic system they queried either randomly, or with probability proportional (or inversely proportional) to the number of observed entries in a cell's row or column. All these variants were significantly and consistently worse than the results we reported above.

Condition number: Instead of detecting unstable systems using the local condition number we also experimented with a modified version of `Order&Extend`, which characterized a system $A_x y = t$ as unstable if its condition number $\kappa(A_x)$ was above a threshold. For values of the threshold between 5 and 100 the results were consistently and significantly worse than the results of `Order&Extend` that we report here, both in terms of queries and in terms of error. Further, there was no threshold of the condition number that would perform comparably to `Order&Extend` for any dataset.

4.5 Extension to tensors

More and more, tensors are attracting attention as a way to model real world data. While the algebraic properties of tensors and the algorithms design for them are more complex than those for matrices, the state-of-the-art in tensor methods is rapidly improving. In this section, we give an overview of the extension of `Order&Extend` to tensor-completion. The main tools of `Order&Extend` can be summarized as:

1. **Systems of equations**: Expressing the matrix as a product of two factors $\mathbf{X}$ and $\mathbf{Y}$ allows the entries to be expressed in terms of linear systems.
2. **Nonsingular systems**: Ensuring that the linear systems are nonsingular ensures that there is a single solution, and the recovery is accurate.
3. **Graph propagation**: Modeling the completion with a bipartite graph infection propagation enables the ordering to complete as many missing entries as possible and to determine a small number of queries needed for full completion.

To the best of our knowledge there is not an extension of completion via graph rigidity to tensors that would mirror structural matrix-completion, although Kahle et al. (2016) has begun to address the topic. Since our approach is based on solving iteratively solving nonsingular systems of equations, the tensor completion we obtain is unique in that sense similar to $\mathbb{ICMC}$. However, we do not have extensions of the other completability classes.

4.5.1 Completion as a linear system of equations

The rank of an $n \times m$ matrix can be represented as the smallest integer r such that $\mathbf{M}$ can be written as a $\mathbf{M} = \mathbf{XY}$ where $\mathbf{X}$ is $n \times r$ and $\mathbf{Y}$ is $r \times m$; the factorization can also be viewed as a summation of r rank-one matrices. Work on tensors has brought about a number of decompositions, as for matrices, but also a number of definitions of rank (we refer the reader to a summary written by Kolda and Bader (2009)). However, the CP-decomposition provides an exact analog of the matrix decomposition and rank we use in this work.

The CP decomposition expresses a tensor $\mathbf{T}$ as the sum of rank-one tensors, where an N-way tensor is rank-one if it can be written as the outer product of N vectors. The tensor rank is exactly the minimum number rank-one tensors whose sum is $\mathbf{T}$. Consider a 3-way tensor $\mathbf{T}$, the CP-decomposition is written as:

$$\mathbf{T} = \sum_{a=1}^{r} \mathbf{X}_a \otimes \mathbf{Y}_a \otimes \mathbf{Z}_a$$

With this notation an entry of $\mathbf{T}$ can be expressed as:

$$t_{i,j,k} = \sum_{a=1}^{r} x_{i,a} y_{j,a} z_{k,a}$$

hence we can write the analogue of Equations 4.2 and 4.3:

$$t_{ijk} = \sum_{a=1}^{r} x_{i,a} y_{j,a} z_{k,a} \tag{4.6}$$

$$t_{i'jk} = \sum_{a=1}^{r} x_{i',a} y_{j,a} z_{k,a} \tag{4.7}$$

If the entries t_{ijk} and $t_{i'jk}$ are known in the partially observed tensor and the x's and z's have been previously solved for, then the y's are the only unknowns. Further since the number of unknowns (y's) is equal to the number of equations, the system can be solved for a unique solution of the y's. To be precise, what we recover is a an r-dimensional vector where each entry is the jth coordination of $\mathbf{Y}_a$ for $a = 1$ to r; this is represented with the vertical stripe in $\mathbf{Y}$ in Figure 4·7. What follows is a parallel of the iterative solving of systems of equations performed with matrices.

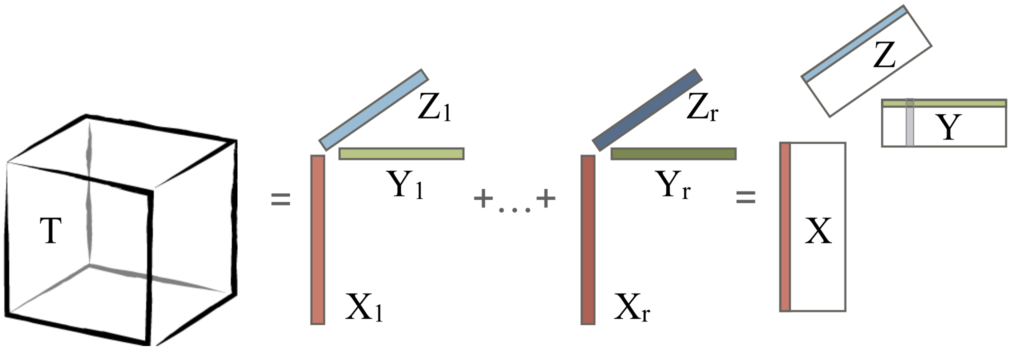


Figure 4·7: Visual of tensor CP-decompositon.

With matrices, we had the notion of the degrees of freedom $\theta = r(n + m - r)$.

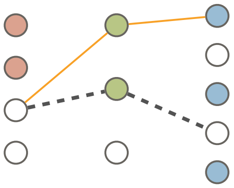


Figure 4·8: Hypergraph for tensor completion

We conjecture that under certain assumptions mimicking the genericity assumption, a similar notion can be defined for tensors. As a proxy we use $\theta_T = r(\sum n_i) - r^N$ where n_i is the ith dimension.

4.5.2 Hypergraph propogation

Once we've established that we can construct systems of equations just as in the matrix case, then next step is to setup an iterative solving of these systems so that the factors $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ can be recovered. Recall that in the matrix formulation we posed the iterative solving as an infection propagation on a bipartite graph, where each node (row in a factor) can be infected (recovered) if it has at least r infected neighbors (known entries in $\mathbf{M}$).

In the case a tensor, instead of two factors ($\mathbf{X}$ and $\mathbf{Y}$) and a bipartite graph, we have N-factors and an N-partite graph which for our purposes is a hypergraph. The infection is setup to begin from an r-dimensional hyperclique and propagate to susceptible nodes. Instead of evaluating whether a node has r infected neighbors, we need to check if it has r infected incident edges where a hyperedge is infected if $n-1$ of the nodes participating in the edge have been infected. For example, in Figure 4·8, the filled nodes are infected and the empty are not. Hence, the solid orange edge is labeled infected while the dashed black edge is not. If a node has r incident infected hyperedges (orange edges) then it can be infected, and the corresponding part of the factor can be recovered. If all nodes are infected, all factors can be recovered, and a full estimate of $\mathbf{T}$ can be computed.

Ordering and solving systems In principle, both the initial minimum degree ordering and adjustments that follow can be extended to hypergraphs. Define the hyperdegree of a node as the number of incident hyperedges, and infected hyperdegree as the number of incident hyperedges that are infected. Note this differs from matrices where the infected degree is the number of infected neighbors. The ordering and adjustments, as described in Section 4.3.2, can be modified for hyperdegree and applied to the hypergraph.

The linear systems are then solved according the the specified ordering. When an incomplete or unstable system is encountered, it is alleviated in the same way as in Section 4.3.4.

Initialization: One complication that arises is the question of initialization. With matrices, the decomposition $\mathbf{M} = \mathbf{XY}$ of a rank r matrix is not unique since it can be re-written as $(\mathbf{XW})(\mathbf{W}^{-1}\mathbf{Y})$ for any invertible $r \times r$ matrix $\mathbf{W}$. This allows for a simple initialization of the very first system A_x to an arbitrary set of r linearly independent rows or columns. Unfortunately, the CP decomposition is unique more often than not, hence setting the first system to arbitrary values could result in a highly inaccurate recovery.

However, having the ability to make queries to missing entries allows us to circumvent this complication. Recall that in the matrix case, the algorithm needs an $r \times r$ submatrix of known entries (an r-clique) to start the iterative solving. Similarly, in the tensor case we require an r-dimensional order-N subtensor of known entries to being the iterative solving procedure. Instead of initializing with a random nonsingular set of vectors, we initialize with the CP-decomposition of the subtensor. This initialization fixes the factors to the correct ones since the remainder will be recovered based on this starting point.

4.5.3 The `Order&ExtendTensor` algorithm

We now give the pseudocode for the extension of `Order&Extend` to tensors. The algorithm mimics `Order&Extend`, is capable of producing partial completions, and applies naturally to dynamic data.

Algorithm 9 The `Order&ExtendTensor` routine

```
Input: $\mathbf{T}_\Omega$,$r$,$\theta$
Compute $H_\Omega$
Find ordering $\pi$
For the starting $r$-hyperclique, query any missing entries in the subtensor
Initialize the factors of $\mathbf{T}$ with the CP-decomposition of the substensor
for each system encountered do
    Solve and query as in Order&Extend
end for
return $\hat{\mathbf{T}}$ as the CP-product of the estimated factors
```

The first step is the construct the hypergraph H_Ω. Once an ordering π is found, the first r nodes from each part of the N-partite graph are selected as the indices of the initial subtensor. The missing entries of the subtensor are queried, and the CP-decomposition of the subtensor is computed. The components in the factors of $\mathbf{T}$ corresponding the initial infected nodes are initialized with the CP-decomposition Each $r \times r$ factor of the decomposed subtensor corresponds to a subset of a factor of $\mathbf{T}$ (indicated by the infected nodes in each part of H_Ω) and is used to initialize the algorithm.

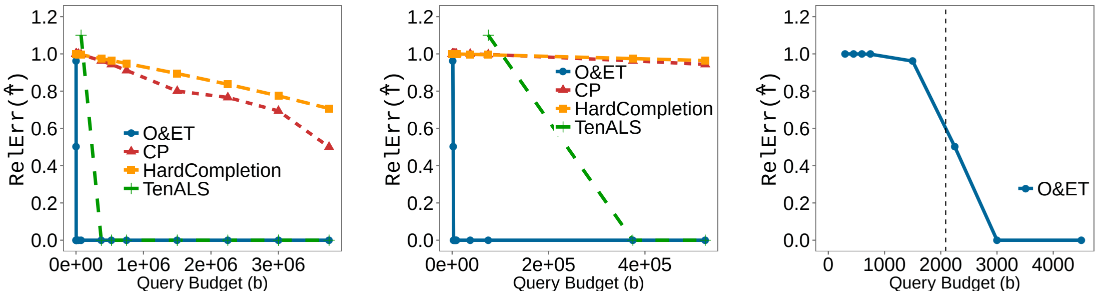
4.5.4 Experiments

In this section, we mirror the experiments in Section 4.4.2 where we evaluate both the accuracy and the number of queries that need to be made using synthetic data. We generate a $100 \times 250 \times 300$ random Gaussian tensor with tensor-rank$= 4$. The density of the input mask is chosen in the same way as in 4.4.2, and we measure the error as we vary the query budget b. To both generate tensors and apply the

CP-decomposition we use the code provide by the TensorLab Toolbox[7].

We compare to three tensor-completion methods. The first algorithm was presented by Signoretto et al. (2014) and is based on a convex formulation that generalizes the nuclear norm. Second, we compare to an alternative minimization approach proposed by Jain and Oh (2014). Finally, we include a heuristic of finding the CP-decomposition of the partially observed tensor and constructing the estimate from the factors – a proxy for the best rank-r approximation. In line with the matrix experiments, since these methods are not 'active', we adapt them a querying strategy that adds b entries in randomly chosen locations.

Figure 4·9a shows the RelError of `Order&ExtendTensor`, `CP`, `TenALS`, and `HardCompletion` as a function of the query budget. We observe that, as with matrices, our algorithm finds an accurate completion with fewer number of queries than the comparison algorithms.



(a) Tensor completion error (b) Close-up (c) Further Close-up

Figure 4·9: RelError of completion achieved by `Order&ExtendTensor`, `HardCompletion` and `CP` on datasets with *exact* rank; x-axis: query budget b; y-axis: RelError of the completion.

In Figure 4·9b and Figure 4·9c we show a close-ups of Figure 4·9a. The vertical line in Figure 4·9c represents our conjectured θ_T and we observe that `Order&ExtendTensor` asks almost only θ_T queries. The most competitive algorithm

---

[7] http://www.tensorlab.net/

is `TenALS`; however, while our approach requires approximately 3000 queries to attain almost perfect completion, `TenAls` has relative error on the order of thousands and requires around $400,000$ queries to reach an accurate completion.

4.6 Active completability for other classes

We have described an algorithm for matrix completion that is based on a routine to bring a mask into $\mathbb{ICMC}$. More precisely, for a partially observed dataset, the algorithm `Order&Extend` adds entries to ensure an accurate completion in a way that is based on `check-ICMC` (we refer the reader to Chapter 3 for the definition appropriate notations, and overview of the classes). In the special case of exact rank, `Order&Extend` can be viewed as adding entries so that the mask belongs to the $\mathbb{ICMC}$ class.

A similar idea can be considered with respect to the other classes, namely: *for a mask that is not a member of a given class, which* 0*s should be flipped to* 1*s so that the mask becomes a member of the class?*

4.6.1 Bringing a mask into $\mathbb{L}$

Recall that the condition for determining whether a mask is a member of $\mathbb{L}$ relates to the rank of the completion matrix. If the rank of $\mathcal{C}_\Omega$ is $\phi = r(n+m-r)$ the mask is in $\mathbb{L}$ and otherwise it is not in $\mathbb{L}$. Since an $n \times m$ matrix of rank r has exactly ϕ degrees of freedom the maximum rank of $\mathcal{C}$ is ϕ. Together, this means that if a mask is not in $\mathbb{L}$ it is because the rank of the completion matrix is less than ϕ; hence to bring a mask into $\mathbb{L}$ we simply need to increase the rank of its corresponding completion matrix.

A strategy for increasing the rank of a matrix we need to add independent rows. Since each row of the completion matrix corresponds to an edge, by increasing the rank we are adding edges to the mask. The special structure of the completion matrix simplifies the identification of which rows (edges) need to be added; the fact that the columns are broken up into non-overlapping groups of r allows us to identify which group of r (which node) is missing one of its r pivots. The procedure is shown in Algorithm 10.

Algorithm 10 $\mathbb{L}$`-addition`

```
   Input $\mathcal{C}_\Omega$,$\Omega$
1: Find the row-echelon form of $\mathcal{C}_\Omega$
2: while the rank of $\mathcal{C}_\Omega < \phi$ do
3:     Find the first column $i$ without a pivot, starting from the left
4:     Find node $u$ corresponding to $i$, and a random node $v$ not adjacent to $u$
5:     Add an edge $(u, v)$ to $\Omega$ and the proper row to $\mathcal{C}_\Omega$
6:     Recompute the row-echelon form of $\mathcal{C}_\Omega$
7: end while
8: Output $\Omega$, $\mathcal{C}_\Omega$
```

The algorithm finds the row-echelon form of the completion matrix, the pivots of which reveal which rows and columns are linearly independent. The pivots are then used to identify which rows (edges) need to be added to create a pivot and increase the rank. In step 4, the addition of a new row will not necessarily increase the rank of $\mathcal{C}_\Omega$. When this is the case, and the rank is unchanged, the algorithm tries adding and edge (u, v') to another non-adjacent node v'. When the rank of $\mathcal{C}_\Omega$ reaches ϕ, the algorithm is terminated, and the resulting mask belongs in $\mathbb{L}$.

Note that in (6) we often do not need to recompute the full row-echelon form. Because of the special structure of $\mathcal{C}_\Omega$ each new row can only affect the r columns corresponding to u and v (because all other entries are zero). Hence, we only need to update and perform mini Gaussian eliminations on the $r \times r$ submatrix of rows which corresponds to incident edges of u.

4.6.2 Bringing a mask into $\mathbb{SC}$

The $\mathbb{SC}$ condition is similar to the $\mathbb{L}$ condition in that it also relies on reaching a certain matrix rank. In this case, to bring a mask in $\mathbb{SC}$, we need to increase the rank of the stress-matrix $\mathcal{S}_\rho$. This increase can be done in a way similar the Algorithm 10, by finding columns without pivots in the row-echelon form of $\mathcal{S}_\rho$. However, here things become a bit more complicated.

For example, consider that a pivot is missing from column j and we decide to add the entry (i, j) to increase the rank of $\mathcal{S}_\rho$. (Recall that in $\mathcal{S}_\rho$ an entry corresponds directly to an edge and a row of $\mathcal{C}_\Omega$). While placing a random value in $\mathcal{S}_\rho(i, j)$ is likely to increase the rank, it must also be true that the corresponding vector ρ remains in the null space of $\mathcal{C}_\Omega$. This may not hold once we have added $\mathcal{S}_\rho(i, j)$ and the corresponding new row in $\mathcal{C}_\Omega$. Hence, once a row is added to $\mathcal{C}_\Omega$ the ρ must be recomputed, and the new $\mathcal{S}_\rho$ will not necessarily have the same rank as it did before (i, j).

In practice we observed that although not every addition resulted in a rank increase, adding in the greedy manner described above worked well. The approach is described in Algorithm 11.

Algorithm 11 $\mathbb{SC}$-addition

```
Input $\mathcal{C}_\Omega$,$\Omega$
$\Omega'$, $\mathcal{C}'_\Omega$=$\mathbb{L}$-addition($\Omega, \mathcal{C}_\Omega$)
Find the stress matrix $\mathcal{S}_\rho$ of $\mathcal{C}'_\Omega$
while the rank of $\mathcal{S}_\rho < \min(n, m) - r$ do
    Bring $\mathcal{S}_\rho$ to row-echelon form
    Find the first column $u$ without a pivot, starting from the left
    Add a row to $\mathcal{C}_\Omega$ for a new edges $(u, v)$
    Recompute $\mathcal{S}_\rho$
end while
Output $\Omega$, $\mathcal{C}_\Omega$
```

The first step of Algorithm 11 brings the mask into $\mathbb{L}$. This step is not strictly necessary as in principle we could add edges to bring the mask directly into $\mathbb{SC}$; however, bringing the mask into $\mathbb{L}$ before bringing it into $\mathbb{SC}$ was observed to require fewer edges. After that, the algorithm proceeds by adding entries to $\mathcal{S}_\rho$ until the rank reaches the required value.

4.6.3 Bringing a mask into $\mathbb{CC}$

Unlike the previous two conditions, $\mathbb{CC}$ is characterized by almost-cliques. The procedure iteratively identifies a $K^-_{r+1,r+1}$ ($(r+1) \times (r+1)$ submatrices with only one missing entries), and flips the 0 to a 1 in the mask. When a mask is not in $\mathbb{CC}$, it is because at some point in the iteration no $K^-_{r+1,r+1}$ can be formed yet there remain missing entries. Hence, an approach to bringing a mask into $\mathbb{CC}$ is adding entries to form a sequence of $K^-_{r+1,r+1}$ for the remaining entries.

Choosing an efficient strategy for adding entries is not straightforward. One approach is to iterate through missing entries (i, j) (through 0s), find a $(r+1)\times(r+1)$ submatrix for each, and add missing entries in the submatrix aside from (i, j) – forming a $K^-_{r+1,r+1}$. Clearly, this may require many more additional entries than necessary. In an attempt to minimize the number of edges added we take a greedy approach and add entries to the densest submatrices first.

Algorithm 12 $\mathbb{CC}$-addition

```
Input $\Omega$
Run $\Omega'$ =check-CC($\Omega, r$)
while $\Omega'$ does not contain all entries do
    Find the missing entry $(i, j)$ with the densest $(r+1) \times (r+1)$ submatrix $S$
    Add all missing entries in $S$ aside from $(i, j)$ to $\Omega$ and $\Omega'$
    Add $(i, j)$ to $\Omega'$
    Run $\Omega'$ =check-CC($\Omega', r$)
end while
Output $\Omega$
```

Algorithm 12 shows a heuristic procedure for bringing a mask into $\mathbb{CC}$. The set

Ω maintains the mask and the entries that have been added, while Ω' keeps track of which entries have been recovered in the procedure. Experiments showed that the greedy heuristic described required less additional entries than others, such as randomly added entries or ordering by row and column density.

Chapter 5

Localized matrix-completion

So far we have placed emphasis on the commonality and practicality of the low-rank assumption in matrix completion. This *global* low-rank assumption is at the center of traditional matrix-completion algorithms that optimize over the whole matrix and try to fit a single model to the whole data. However, in the design of many algorithms for analyzing and utilizing this data, there is the further assumption of the existence of even lower-rank submatrices. For example, in user-preference data, there is expected to be subsets of users that behave similarly on subsets of products. This is the underlying assumption made by Aggarwal (2016), Herlocker et al. (2004), Sarwar et al. (2001), and others in standard collaborative filtering, where recommendations are made to a user based on the preferences of other similar users. This is also a common assumption for analysis of traffic networks, user-preferences, and datasets in the natural sciences (e.g., gene expression data).

Despite the fact that the assumption of low-rank submatrices is prevalent to many data-analysis settings, there is very little work in the matrix-completion literature that captures such structure. We argue that matrices that contain low-rank submatrices are not accurately estimated by traditional, `Global`, approaches to matrix-completion that make a global assumption on the rank, such as the ones described in the previous chapters. Given that in many applications low-rank submatrices correspond to particularly interesting parts of the data, it is important to find a good completion of the entries that are missing in the submatrices.

In this chapter we explore the benefit of capturing substructure in the context of matrix completion, and ask: *Can the knowledge of the existence of a low-rank submatrix improve the accuracy of completion?*

5.1 Notation and assumptions

Throughout we use $\mathbf{M}$ to refer to an $n \times m$ fully-known matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ of rank r, and we use $\mathbf{M}_i$ to denote the i-th row of $\mathbf{M}$ and $\mathbf{M}_{ij}$ the entry (i, j). If $\mathbf{M}$ is not completely known we use $\Omega \subseteq \{(i, j)|1 \leq i \leq n, 1 \leq j \leq m\}$ to denote the subset of entries that are observed, and when a matrix completion algorithm is applied to the partially observed $\mathbf{M}_\Omega$, we use $\hat{\mathbf{M}}$ to denote the output estimate.

For a submatrix $\mathbf{S}$ of $\mathbf{M}$, we use $R_s \subseteq \{1, \dots, n\}$ and $C_s \subseteq \{1, \dots, m\}$ to denote the rows and columns of the original matrix that fully define $\mathbf{S}$. That is, $\mathbf{S} = \mathbf{M}(R_s, C_s)$. Also, for a set of rows R_s we use the term *complement* to refer to the set of rows $\overline{R_s} = \{\{1, \dots, n\} \backslash R_s\}$. The notation $\mathbf{v_1}$ is reserved for the first singular vector of $\mathbf{M}$, and for any other matrix $\mathbf{S}$ we use $\mathbf{s_i}$ to denote the i-th right singular vector of $\mathbf{S}$. We also use $\mathbf{Y}$ to denote the *complement* of $\mathbf{S}$, that is if $\overline{R_s} = \{1, \dots, n\} \setminus R_s$ and $\overline{C_s} = \{1, \dots, m\} \setminus C_s$, then $\mathbf{T} = \mathbf{M}(\overline{R_s}, \overline{C_s})$.

To measure the error of a completion we use the relative Frobenius error:

$$\texttt{RelErr}(\hat{\mathbf{M}}) = \frac{\|\mathbf{M} - \hat{\mathbf{M}}\|_f^2}{\|\mathbf{M}\|_f^2} \tag{5.1}$$

Finally, we use the following conventions: $\|\cdot\|$ as shorthand for the L_2 norm $\|\cdot\|_2$ of a vector or matrix. Recall, that the L_2 norm of a matrix is the value of its first singular value. For vectors $\mathbf{x}$ and $\mathbf{y}$, we also use $\langle \mathbf{x}, \mathbf{y} \rangle$ to denote the inner product of vectors $\mathbf{x}$ and $\mathbf{y}$, and $\mathbf{x} \otimes \mathbf{y}$ to denote their outer product..

The assumptions made in this chapter are the same as the previous chapter which are described in Section 4.1.

5.2 Localized matrix-completion

Traditional approaches to matrix completion, which we call `Global`, typically assume that the whole matrix is of a particular rank r. Together with assumptions such as incoherence or genericity, the problems is framed as the task of finding a single model that estimates the whole matrix. When, as it has been observed across various applications, a matrix contains low-rank submatrices, a single model does not accurately estimate both the submatrices and the rest of the data at once. Intuitively, this comes from the fact that the submatrix and its complement are deemed two separate matrices by the traditional notion of a matrix in the completion literature. In an extreme view, applying `Global` to such a matrix with a low-rank submatrix is akin to fitting one model to the concatenation of two separate datasets.

To address this limitation of the `Global` approach, we propose a `Localized` strategy that separates low-rank submatrices and completes them separately from the rest of the matrix.

On input $\mathbf{M}_\Omega$, the `Localized` algorithm:

1. Finds the locations of low-rank submatrices $\mathcal{S}= \{(R_s, C_s)\}$
2. Separates the entries of the low-rank submatrices from the rest $\Omega' = \Omega \setminus \mathcal{S}$
3. Completes the submatrices $\mathcal{S}$ separately from the rest of the matrix $\mathbf{T}$
4. Combines the estimates back into a matrix $\hat{\mathbf{M}}$ as the output completion

Since in practice the locations of low-rank submatrices within a partially-observed matrix are not known to analysts or data-owners, the first step of the algorithm searches for evidence of low-rank submatrices in the matrix. Next, the low-rank submatrices are separated from the complement and in step 3, a completion algorithm is applied to each component separately. Finally, step 4 merges the estimates back into a full completion (this is done to evaluate the error and is not strictly necessary). Note that if no submatrices are found in step 1, the algorithm simply reverts to `Global`.

To the best of our knowledge, step 1 has not been fully addressed in the literature to date. In fact, the problem of finding a low-rank submatrix in a fully known, let alone partially-known, matrix is an open and interesting problem. Hence before investigating the benefit of a `localized` approach to matrix completion, we first study step 1, which we call the LRDiscovery.

5.3 Low-rank submatrix discovery

In this section, we study the first step of `Localized`: the problem of finding a low-rank submatrix which we call LRDiscovery. We first focus on the setting where the matrix is fully-known since the problem is interesting in itself and has received little attention. After developing an algorithm and providing an analysis of when it is likely to succeed, we develop a natural extension to partially observed matrices.

Our approach to LRDiscovery is inspired by the Singular Value Decomposition (SVD) of a matrix, which has proven particularly useful in extracting structure from a matrix. We consider a data matrix $\mathbf{M}$ whose rows and columns are separated into sets that define $\mathbf{S}$ (the low-rank submatrix) and $\mathbf{T}$ (the matrix defined by the remaining rows and columns). Our main contribution is the characterization of when the singular vectors of $\mathbf{M}$ can be used to identify the rows and the columns that form $\mathbf{S}$. Intuitively we isolate two factors that are important for this task: the magnitude of the largest singular value of $\mathbf{S}$ and the singular vectors of $\mathbf{S}$ and $\mathbf{T}$. At a high level, our analysis reveals that when the magnitude of the first singular value of $\mathbf{S}$ is large and the singular vectors of $\mathbf{S}$ are sufficiently different from the singular vectors of $\mathbf{T}$, then the first singular vector $\mathbf{v_1}$ of $\mathbf{M}$ will orient towards the span of $\mathbf{S}$ and thus help identify $\mathbf{S}$ when the data is projected on $\mathbf{v_1}$. To the best of our knowledge, we are the first to provide such an analysis and characterization.

Our algorithm, which we call `SVP`, is inspired by the above analysis and it has two simple steps: first, it finds a projection of the rows (and columns) of $\mathbf{M}$ on the first left (resp. right) singular vector of $\mathbf{M}$. Then, it projects the rows (resp. columns) of $\mathbf{M}$ on the appropriate singular vector and clusters the 1-dimensional points into 2 clusters. The cluster of rows (resp. columns) with the largest mean is an estimate of the rows (resp. columns) of $\mathbf{S}$.

The key advantages of `SVP` is its simplicity and efficiency – it only requires the computation of the first left and right singular vectors of $\mathbf{M}$. Our experimental evaluation with data generated using different models demonstrates that `SVP` is able to succeed in the identification of $\mathbf{S}$ even when $\mathbf{S}$ has relatively large rank, when there is missing data in $\mathbf{M}$, or even when there are multiple low-rank submatrices planted in $\mathbf{M}$. None of the other methods available for the same problem today can exhibit the same level of success in such a wide range of inputs.

5.3.1 Exposing low-rank submatrices

First, we lay out our main analytical contribution in which we characterize when the singular vectors of $\mathbf{M}$ can be used to expose evidence that there exists a low-rank submatrix $\mathbf{S}$ in $\mathbf{M}$. We focus the discussion on finding the rows of this submatrix (i.e., we fix $C_s = \{1 \dots m\}$), but the analysis is symmetric for the columns. The following two parameters are key to our characterization:

$$\pi(\mathbf{S},\mathbf{T}) = \frac{\|\mathbf{S}\|^2}{\|\mathbf{T}\|^2} \qquad \text{and} \qquad \gamma(\mathbf{S},\mathbf{T}) = \max_j \langle \mathbf{s_1}, \mathbf{t_j} \rangle$$

The first parameter measures the magnitude of the L_2-norm of $\mathbf{S}$ with respect to $\mathbf{T}$; whenever $\mathbf{S}$ and $\mathbf{T}$ are clear from the context we use π to denote $\pi(\mathbf{S},\mathbf{T})$. The second parameter, $\gamma(\mathbf{S},\mathbf{T})$, measures the geometric orientation of $\mathbf{S}$ with respect to $\mathbf{T}$. Intuitively, when γ is small, $\mathbf{S}$ and $\mathbf{T}$ are well-separated. Again when $\mathbf{S}$ and $\mathbf{T}$ are clear from the context we use γ to denote $\gamma(\mathbf{S},\mathbf{T})$.

These two parameters allow us to characterize the behavior of $\mathbf{v_1}$ with respect to the magnitude and geometry of $\mathbf{S}$ in $\mathbf{M}$. In particular, we show that when π is large and γ is adequately small with respect to π, the normalized projections of $\mathbf{S}_i$ on $\mathbf{v_1}$ will be *larger* than the normalized projections of $\mathbf{T}_i$ on $\mathbf{v_1}$. The gap between the projections suggests an approach to LRDiscovery that uses $\mathbf{v_1}$ to expose evidence of $\mathbf{S}$.

To simplify the discussion and capture the intuition above we define $\Delta_{\mathbf{S},\mathbf{T}} = \frac{1}{|R_s|}\|\Omega(\mathbf{S})\mathbf{v_1}\|_1 - \frac{1}{|R_t|}\|\Omega(\mathbf{T})\mathbf{v_1}\|_1$, where $\Omega(\mathbf{X})$ is an operator that normalizes each row of matrix $\mathbf{X}$. The crux of our analysis lies in the following proposition:

Proposition 11. *If* $\pi > 1$ *and* $\gamma < \epsilon(\pi - 1)$*, then* $\Delta_{\mathbf{S},\mathbf{T}} > (1-\gamma)(1-\epsilon)$ *and we say* $\Delta_{\mathbf{S},\mathbf{T}}$ *is* ϵ*-close to* $(1-\gamma)$*.*

To prove Proposition 11, we first consider a special case that simplifies the setting. A series of propositions and lemmas are required before reaching the final result, afterwhich we address the more general case.

Special case of rank-1:

Consider the case in which $\mathbf{M}$ consists of two rank-one submatrices: $\mathbf{S} = \mathbf{M}\left(R_s, :\right)$ and its complement $\mathbf{T} = \mathbf{S}\left(\overline{R_s}, :\right)$, both of which span all the columns. The first singular vector and value of $\mathbf{S}$ are $\mathbf{s_1}$ and σ_1, and the first singular vector and value of $\mathbf{T}$ are $\mathbf{t_1}$ and τ_1. We start by describing the quadratic form of $\mathbf{M}$ in terms of $\mathbf{S}$ and $\mathbf{T}$.

Lemma 12. *The first singular vector* $\mathbf{v_1}$ *of* $\mathbf{M}$ *is equal to the first eigen vector of* $\mathbf{Q} = \sigma_1^2\mathbf{s_1}\mathbf{s_1}^T + \tau_1^2\mathbf{t_1}\mathbf{t_1}^T$*, i.e.* $\mathbf{v_1} = \max_{\mathbf{z},\|z\|^2=1}\mathbf{z}^T\mathbf{Q}\mathbf{z}$.

Proof of Lemma 12. Recall that the first right singular vector $\mathbf{v_1}$ of $\mathbf{M}$ is the one that maximizes $\|\mathbf{Mv_1}\|^2$ over all unit vectors. We now consider this expression $\|\mathbf{Mv_1}\|^2$ and write it as a quadratic form in terms of $\mathbf{S}$ and $\mathbf{T}$.

$$\begin{aligned}\|\mathbf{Mv_1}\|^2 &= \sum_{\mathbf{M}} \langle \mathbf{M}_i, \mathbf{v_1}\rangle^2 \\ &= \sum_{\mathbf{S}} \langle \mathbf{S}_i, \mathbf{v_1}\rangle^2 + \sum_{\mathbf{T}} \langle \mathbf{T}_i, \mathbf{v_1}\rangle^2 \\ &= \|\mathbf{Sv_1}\|^2 + \|\mathbf{Tv_1}\|^2 \\ &= \mathbf{v_1}^T(\mathbf{S}^T\mathbf{S})\mathbf{v_1} + \mathbf{v_1}^T(\mathbf{T}^T\mathbf{T})\mathbf{v_1} \\ &= \mathbf{v_1}^T(\sigma^2\mathbf{s_1} \otimes \mathbf{s_1}^T)\mathbf{v_1} + \mathbf{v_1}^T(\tau^2\mathbf{t_1} \otimes \mathbf{t_1}^T)\mathbf{v_1} \\ &= \mathbf{v_1}^T(\sigma^2\mathbf{s_1} \otimes \mathbf{s_1}^T + \tau^2\mathbf{t_1} \otimes \mathbf{t_1}^T)\mathbf{v_1} = \mathbf{v_1}^T(\mathbf{M}^T\mathbf{M})\mathbf{v_1} \qquad (5.2)\end{aligned}$$

The unit vector $\mathbf{v_1}$ that maximizes $\|\mathbf{Mv_1}\|^2$ will also maximize each intermediate expression. Hence, finding the eigenvector corresponding to the largest eigenvalue of $\sigma^2\mathbf{s_1} \otimes \mathbf{s_1}^T + \tau^2\mathbf{t_1} \otimes \mathbf{t_1}^T$ corresponds to finding the first singular vector of $\mathbf{M}$. From here on we used $\mathbf{Q}$ to refer to $\mathbf{Q} = \mathbf{M}^T\mathbf{M} = \sigma^2\mathbf{s_1} \otimes \mathbf{s_1}^T + \tau^2\mathbf{t_1} \otimes \mathbf{t_1}^T$. □

Next we derive an expression for λ, the first eigenvalue of $\mathbf{Q}$ in terms of σ and τ.

Proposition 13. *Using λ_i to denote the i-th largest eigenvalue of* $\mathbf{Q}$

$$\lambda_1, \lambda_2 = \frac{\tau^2}{2}\left[\pi + 1 \pm \sqrt{((1-\pi)^2 + 4\pi\gamma^2}\right]$$

Proof of Proposition 13. Consider the trace of the matrix $\mathbf{Q} = \mathbf{M}^T\mathbf{M}$

$$Tr(\mathbf{Q}) = \sum_i \lambda_i \text{ and } Tr(\mathbf{Q}^2) = \sum_i \lambda_i^2$$

Since the rank($\mathbf{M}$) ≤ 2, we can consider only the largest two eigenvalues and expand the expressions for the trace to:

$$Tr(\mathbf{Q}) = \lambda_1 + \lambda_2 \qquad (5.3)$$

$$Tr(\mathbf{Q}^2) = \lambda_1{}^2 + \lambda_2{}^2 \qquad (5.4)$$

At the same time we can compute the trace explicitly in terms of σ and τ:

$$\begin{aligned}
Tr(\mathbf{Q}) &= \tau^2 + \sigma^2 \\
Tr(\mathbf{Q}^2) &= Tr(\tau^2 \mathbf{t_1} \otimes \mathbf{t_1}^T \tau^2 \mathbf{t_1} \otimes \mathbf{t_1}^T + 2\tau^2 \mathbf{t_1} \otimes \mathbf{t_1}^T \sigma^2 \mathbf{s_1} \otimes \mathbf{s_1}^T + \sigma^2 \mathbf{s_1} \otimes \mathbf{s_1}^T \sigma^2 \mathbf{s_1} \otimes \mathbf{s_1}^T) \\
&= \tau^4 + \sigma^4 + 2\sigma^2\tau^2\gamma^2
\end{aligned}$$

Hence $(\lambda_1 + \lambda_2) = (\tau^2 + \sigma^2)$ and $({\lambda_1}^2 + {\lambda_2}^2) = (\tau^4 + \sigma^4 + 2\tau^2\sigma^2\gamma^2)$. By doing a bit of manipulation we can also recover the valuation of $\lambda_1\lambda_2$:

$$\begin{aligned}
(\lambda_1 + \lambda_2)^2 &= {\lambda_1}^2 + 2\lambda_1\lambda_2 + {\lambda_2}^2 \\
&= (\tau^2 + \sigma^2)^2 = \sigma^4 + 2\tau^2\sigma^2 + \tau^4 \\
(\lambda_1 + \lambda_2)^2 - (\lambda_1^2 + \lambda_2^2) &= (\tau^4 + 2\tau^2\sigma^2 + \tau^4) - (\tau^4 + 2\sigma^2\tau^2\gamma^2 + \sigma^4) \\
&= \lambda_1\lambda_2 = \tau^2\sigma 2 - \sigma^2\tau^2\gamma^2
\end{aligned}$$

Now that we have $\lambda_1\lambda_2$ and $(\lambda_2 + \lambda_2)$, combine these with the characteristic equation for which λ_1 and λ_2 are roots.

$$C^2 - C(\lambda_1 + \lambda_2) + \lambda_1\lambda_2 = C^2 - C(\tau^2 + \sigma^2) + \tau^2\sigma^2 - \sigma^2\tau^2\gamma^2$$

The discriminant of this equation is

$$(\tau^2 + \sigma^2)^2 - 4(\tau^2\sigma^2 - \sigma^2\tau^2\gamma^2) = (\sigma^2 - \tau^2)^2 + 4\sigma^2\tau^2\gamma^2$$

Since the discriminant if postive, > 0, there are two real roots.

$$\lambda_1 \text{ and } \lambda_2 = \frac{(\tau^2 + \sigma^2) \pm \sqrt{(\tau^2 - \sigma^2)^2 + 4\sigma^2\tau^2\gamma^2}}{2}$$

Observe that in the special case where $\mathbf{s_1}$ and $\mathbf{t_1}$ are orthogonal. In this case $\gamma^2 = 0$ and if $\sigma > \tau$ then $\lambda_1 = \sigma^2$ and $\lambda_2 = \tau^2$.
Recall that $\pi = \frac{\sigma^2}{\tau^2}$ and substitute σ^2 with $\pi\tau^2$.

$$\begin{aligned}
\lambda_1, \lambda_2 &= \frac{1}{2}\left[(\tau^2 + \sigma^2) \pm \sqrt{(\tau^2 - \sigma^2)^2 + 4\sigma^2\tau^2\gamma^2}\right] \\
&= \frac{1}{2}\left[(\pi + 1)\tau^2 \pm \sqrt{((1 - \pi)^2\tau^4 + 4\pi\tau^4\gamma^2}\right] \\
&= \frac{\tau^2}{2}\left[\pi + 1 \pm \sqrt{((1 - \pi)^2 + 4\pi\gamma^2}\right]
\end{aligned}$$

□

Using Proposition 13 we can obtain a bound on its magnitude. Suppose $\pi = 1$. Then $\lambda_1, \lambda_2 = \tau^2(1 \pm \gamma)$. If $\gamma = 0$ then $\mathbf{s_1}$ and $\mathbf{t_1}$ are orthogonal and the two eigenvalues will equal. If $\mathbf{s_1}$ and $\mathbf{t_1}$ are parallel, then there will be one eigenvalue which be equal to the sum of their magnitudes. Now suppose $\pi > 1$. In this case we can write $\lambda = \lambda_1 = \frac{\tau^2}{2}\left[\pi + 1 + \sqrt{((\pi - 1)^2 + 4\pi\gamma^2}\,\right]$. Since we know that $0 < \gamma < 1$ we can also obtain a bound on λ, namely $\pi\tau^2 < \lambda < (\pi + 1)\tau^2$.

Next we show that $\mathbf{v_1}$ can always be expressed as a linear combination of just $\mathbf{s_1}$ and $\mathbf{t_1}$, i.e. $\mathbf{v_1} = \alpha\mathbf{s_1} + \beta\mathbf{t_1}$.

Proposition 14. *The first right singular vector* $\mathbf{v_1}$ *of* $\mathbf{M}$ *can be written as* $\mathbf{v_1} = \alpha\mathbf{s_1} + \beta\mathbf{t_1}$ *for*

$$\alpha = \frac{\sigma}{c}\left[\mathbf{s_1}(n)(\lambda - \sigma^2) + \mathbf{t_1}(n)\sigma\tau\gamma\right] \text{ and } \beta = \frac{\tau}{c}\left[\mathbf{t_1}(n)(\lambda - \tau^2) + \mathbf{s_1}(n)\sigma\tau\gamma\right]$$

Fact 1. *For any matrix* $\mathbf{M}$*, its first right singular vector* $\mathbf{v_1}$*, the largest eigenvalue* λ *of* $\mathbf{Q} = \mathbf{M}^T\mathbf{M}$*, and* $\mathbf{I}$ *as the identity matrix, the following holds:* $(\mathbf{Q} - \mathbf{I}\lambda)\mathbf{v_1} = 0$.

We prove Proposition 14 by showing that Fact 1 holds for the proposed α and β, i.e. $(\mathbf{Q} - \mathbf{I}\lambda)(\alpha\mathbf{s_1} + \beta\mathbf{t_1}) = 0$.

Proof of Proposition 14.

$$\begin{aligned}(\mathbf{Q}-\mathbf{I}\lambda)\mathbf{v_1} &= (\sigma^2\mathbf{s_1}\otimes\mathbf{s_1}^T+\tau^2\mathbf{t_1}\otimes\mathbf{t_1}^T-\mathbf{I}\lambda)\mathbf{v_1}\\ &= \frac{1}{c}(\sigma^2\mathbf{s_1}\otimes\mathbf{s_1}^T+\tau^2\mathbf{t_1}\otimes\mathbf{t_1}^T-\mathbf{I}\lambda)(\mathbf{s_1}\alpha+\mathbf{t_1}\beta)\end{aligned}$$

Substituting α and β, multiplying out the terms and grouping them by $\mathbf{s_1}$ and $\mathbf{t_1}$, some terms cancel and we get:

$$\begin{aligned}&= \mathbf{s_1}\frac{\sigma}{c}\big[\mathbf{s_1}(n)(\sigma^2\lambda-\sigma^2\tau^2-\lambda^2+\lambda\tau^2+(\sigma\tau\gamma)^2)\big]+\\ &\quad\ \mathbf{t_1}\frac{\tau}{c}\big[\mathbf{t_1}(n)(\tau^2\lambda-\sigma^2\tau^2-\lambda^2+\lambda\sigma^2+(\sigma\tau\gamma)^2)\big]\\ &= \mathbf{s_1}\frac{\sigma}{c}\big[\mathbf{s_1}(n)(-(\lambda-\sigma^2)(\lambda-\tau^2)+(\sigma\tau\gamma)^2)\big]+\\ &\quad\ \mathbf{t_1}\frac{\tau}{c}\big[\mathbf{t_1}(n)(-(\lambda-\sigma^2)(\lambda-\tau^2)+(\sigma\tau\gamma)^2)\big]\\ &= \psi\frac{1}{c}\big[\sigma\mathbf{s_1}(n)\mathbf{s_1}+\tau\mathbf{t_1}(n)\mathbf{t_1}\big]\end{aligned}$$

Using the expression for λ we show that $\psi=-(\lambda-\sigma^2)(\lambda-\tau^2)+(\sigma\tau\gamma)^2$ is equal to zero.

$$\begin{aligned}\psi &= -(\lambda-\sigma^2)(\lambda-\tau^2)+(\sigma\tau\gamma)^2\\ 4\psi &= -(2\lambda-2\sigma^2)(2\lambda-2\tau^2)+4(\sigma\tau\gamma)^2\\ &= -(\tau^2-\sigma^2+\sqrt{(\sigma^2-\tau^2)^2+4(\sigma\tau\gamma)^2})(\sigma^2-\tau^2+\sqrt{(\sigma^2-\tau^2)^2+4(\sigma\tau\gamma)^2})+4(\sigma\tau\gamma)^2\\ &= -2\sigma^2\tau^2+\sigma^4+\tau^2-(\sigma^2-\tau^2)^2-4(\sigma\tau\gamma)^2+4(\sigma\tau\gamma)^2\\ &= 0\end{aligned}$$

Thus $\mathbf{v_1}=\alpha\mathbf{s_1}+\beta\mathbf{t_1}$ is in the null space of $(\mathbf{Q}-\mathbf{I}\lambda)$ and $\mathbf{v_1}$ is the first right singular vector of $\mathbf{M}$. □

As a consequence we have the following:

Corollary 15. *For σ and τ as the first singular values of rank-one components* $\mathbf{S}$ *and* $\mathbf{T}$ *and γ as the angle between them,* $(\lambda-\sigma^2)(\lambda-\tau^2)=(\sigma\tau\gamma)^2$

Observe that the factors α and β contain the terms $\mathbf{s_1}(n)$ and $\mathbf{t_1}(n)$ respectively, which are elements of $\mathbf{s_1}$ and $\mathbf{t_1}$. We claim that the index of these elements is arbitrary, and can be replaced with $\mathbf{s_1}(i)$ and $\mathbf{t_1}(i)$ upon proper scaling. The expression of $\mathbf{v_1}$ can be transformed to be in terms of $\mathbf{s_1}(i)$ and $\mathbf{t_1}(i)$ instead of $\mathbf{s_1}(n)$ and $\mathbf{t_1}(n)$, by multiplying by a factor z. Denote by $\alpha_i = \frac{\sigma}{c}\big[\mathbf{s_1}(i)(\lambda - \tau^2) + \mathbf{t_1}(i)(\sigma\tau\gamma)\big]$ and similarly for β_i, then in Proposition 14 we have shown that $\mathbf{v_1} = \alpha_n \mathbf{s_1} + \beta_n \mathbf{t_1}$.

Lemma 16. *For any $i \leq n$ we can write $\mathbf{v_1} = \alpha_n \mathbf{s_1} + \beta_n \mathbf{t_1} = z\big[\alpha_i \mathbf{s_1} + \beta_i \mathbf{t_1}\big]$ where*

$$z = \frac{\alpha_n}{\alpha_i} = \frac{\mathbf{s_1}(n)(\lambda - \tau^2) + \mathbf{t_1}(n)(\sigma\tau\gamma)}{\mathbf{s_1}(i)(\lambda - \tau^2) + \mathbf{t_1}(i)(\sigma\tau\gamma)}$$

Lemma 17. *$\frac{\alpha_i}{\alpha_j} = \frac{\beta_i}{\beta_j}$ for α_i and β_i as defined above and $\alpha_i, \alpha_j, \beta_i, \beta_j$ are all nonzero.*

Proof of Lemma 17. We proceed by showing $\alpha_i\beta_j = \alpha_j\beta_i$ using Corollary 15. Set $\Gamma = \sigma\tau\gamma$ for the sake of clarity and assume all α's and β's are nonzero.

$$\begin{aligned}\alpha_i\beta_j &= \frac{\sigma\tau}{c^2}(\mathbf{s_1}(i)(\lambda - \tau^2) + \mathbf{t_1}(i)(\sigma\tau\gamma))(\mathbf{t_1}(j)(\lambda - \sigma^2) + \mathbf{s_1}(j)(\sigma\tau\gamma)) \\ &= \frac{\sigma\tau}{c^2}\big[\mathbf{s_1}(i)\mathbf{t_1}(j)(\lambda - \tau^2)(\lambda - \sigma^2) + \mathbf{s_1}(i)\mathbf{s_1}(j)\Gamma(\lambda - \tau^2) \\ &\qquad + \mathbf{t_1}(i)\mathbf{t_1}(j)\Gamma(\lambda - \sigma^2) + \mathbf{s_1}(j)\mathbf{t_1}(i)\Gamma^2\big] \\ &= \frac{\sigma\tau}{c^2}\big[(\mathbf{s_1}(i)\mathbf{t_1}(j) + \mathbf{s_1}(i)\mathbf{t_1}(j))\Gamma^2 + \mathbf{s_1}(i)\mathbf{s_1}(j)(\lambda - \tau^2) + \mathbf{t_1}(i)\mathbf{t_1}(j)\alpha(\lambda - \sigma^2)\big]\end{aligned} \tag{5.5}$$

Using this we can show that

$$\alpha_j\beta_i = \frac{\sigma\tau}{c^2}\big[(\mathbf{s_1}(j)\mathbf{t_1}(i) + \mathbf{s_1}(j)\mathbf{t_1}(i))\Gamma^2 + \mathbf{s_1}(j)\mathbf{s_1}(i)(\lambda - \tau^2) + \mathbf{t_1}(j)\mathbf{t_1}(i)\alpha(\lambda - \sigma^2)\big]$$

which is equivalent to $\alpha_i\beta_j$ up to a factor of $(\mathbf{s_1}(j)\mathbf{t_1}(i) - \mathbf{s_1}(i)\mathbf{t_1}(j))O(\gamma^2)$.
Since $\alpha_i\beta_j = \alpha_j\beta_i$ and all are nonzero $\frac{\alpha_i}{\alpha_j} = \frac{\beta_i}{\beta_j}$. □

Proof of Lemma 16.

$$\begin{aligned}\mathbf{v_1} &= z\big[\alpha_i \mathbf{s_1} + \beta_i \mathbf{t_1}\big] = z\alpha_i \mathbf{s_1} + z\beta_i \mathbf{t_1} \\ &= \frac{\alpha_n}{\alpha_i}\alpha_i \mathbf{s_1} + \frac{\alpha_n}{\alpha_i}\beta_i \mathbf{t_1} \\ &= \alpha_n \mathbf{s_1} + \frac{\beta_n}{\beta_i}\beta_i \mathbf{t_1} = \alpha_n \mathbf{s_1} + \beta_n \mathbf{t_1}\end{aligned}$$

□

Thus the coordinate n in $\mathbf{s_1}(n)$ is arbitrary and will come into play only later.

We now combine the previous derivations to obtain an expressions for the inner products of $\mathbf{s_1}$ and $\mathbf{t_1}$ with $\mathbf{v_1}$ in terms of π and γ.

$$\begin{aligned}
\langle \mathbf{v_1}, \mathbf{s_1} \rangle &= \langle \alpha \mathbf{s_1} + \beta \mathbf{t_1}, \mathbf{s_1} \rangle = \alpha \langle \mathbf{s_1}, \mathbf{s_1} \rangle + \beta \langle \mathbf{t_1}, \mathbf{s_1} \rangle = \alpha + \gamma\beta \\
&= \frac{\sigma}{c}\big[\mathbf{s_1}(n)(\lambda - \tau^2) + \mathbf{t_1}(n)\sigma\tau\gamma\big] + \gamma\frac{\tau}{c}\big[\mathbf{t_1}(n)(\lambda - \sigma^2) + \mathbf{s_1}(n)\sigma\tau\gamma\big] \\
&= \frac{1}{c}\big[\sqrt{\pi}\tau\big[\mathbf{s_1}(n)(\lambda - \tau^2) + \mathbf{t_1}(n)\sqrt{\pi}\tau^2\gamma\big] + \tau\gamma\big[\mathbf{t_1}(n)(\lambda - \pi\tau^2) + \mathbf{s_1}(n)\sqrt{\pi}\tau^2\gamma\big]\big] \\
&= \frac{\tau}{c}\big[\lambda(\sqrt{\pi}\mathbf{s_1}(n) + \gamma\mathbf{t_1}(n)) + \mathbf{s_1}(n)\tau^2\sqrt{\pi}(\gamma^2 - 1)\big] \qquad (5.6)
\end{aligned}$$

Similarly for $\mathbf{t_1}$ we write the inner product, group terms, and simplify the expression.

$$\begin{aligned}
\langle \mathbf{v_1}, \mathbf{t_1} \rangle &= \gamma\alpha + \beta \\
&= \frac{1}{c}\big[\sigma\gamma\big[\mathbf{s_1}(n)(\lambda - \tau^2) + \mathbf{t_1}(n)\sigma\tau\gamma\big] + \tau\big[\mathbf{t_1}(n)(\lambda - \sigma^2) + \mathbf{s_1}(n)\sigma\tau\gamma\big]\big] \\
&= \frac{1}{c}\big[\sqrt{\pi}\tau\gamma\big[\mathbf{s_1}(n)(\lambda - \tau^2) + \mathbf{t_1}(n)\tau\sqrt{\pi}\gamma\big] + \tau\big[\mathbf{t_1}(n)(\lambda - \pi\tau^2) + \mathbf{s_1}(n)\sqrt{\pi}\tau^2\gamma\big]\big] \\
&= \frac{\tau}{c}\big[\lambda(\sqrt{\pi}\mathbf{s_1}(n)\gamma + \mathbf{t_1}(n)) + \mathbf{t_1}(n)\tau^2\pi(\gamma^2 - 1)\big] \qquad (5.7)
\end{aligned}$$

Finally, combining the expressions of the inner products, we compute $\Delta_{\mathbf{S},\mathbf{T}}$ as their differences and then simplify the expression to arrive at Proposition 11.

$$\begin{aligned}
\Delta_{\mathbf{S},\mathbf{T}} = &\frac{\tau}{c}\big[\lambda(\sqrt{\pi}\mathbf{s_1}(n) + \gamma\mathbf{t_1}(n)) + \mathbf{s_1}(n)\tau^2\sqrt{\pi}(\gamma^2 - 1)\big] \\
&-\frac{\tau}{c}\big[\lambda(\sqrt{\pi}\mathbf{s_1}(n)\gamma + \mathbf{t_1}(n)) + \mathbf{t_1}(n)\tau^2\pi(\gamma^2 - 1)\big]
\end{aligned}$$

By rearranging terms:

$$\Delta = \frac{\tau}{c}(1 - \gamma)\big[\sqrt{\pi}\mathbf{s_1}(n)(\lambda - \tau^2(\gamma + 1)) - \mathbf{t_1}(n)(\lambda - \pi\tau^2(\gamma + 1))\big]$$

Here we introduce the following assumptions: (1) $\pi > 1$, (2) $\gamma < \epsilon(\pi - 1)$, and (3) the index n is such that $\mathbf{t_1}(n)\sqrt{\pi} \leq \mathbf{s_1}(n)$.

Lemma 18. *If* $\pi > 1$, $\gamma < \epsilon(\pi - 1)$, *$n$ is such that* $\mathbf{t_1}(n)\sqrt{\pi} < \mathbf{s_1}(n)$ *then the constant* $c = \tau(\lambda - \tau^2)\mathbf{s_1}(n)\sqrt{\pi}$.

Proof of Lemma 18. The constant c comes from the fact that our derivation of the first singular vector as $\mathbf{v_1}{=}\alpha\mathbf{s_1} + \beta\mathbf{t_1}$ without the $\frac{1}{c}$ factor is not unit norm, that is

$$\mathbf{v} = \alpha'\mathbf{s_1} + \beta'\mathbf{t_1} = \big[\mathbf{s_1}(n)(\lambda - \sigma^2) + \mathbf{t_1}(n)\sigma\tau\gamma\big]\mathbf{s_1} + \big[\mathbf{t_1}(n)(\lambda - \tau^2) + \mathbf{s_1}(n)\sigma\tau\gamma\big]\mathbf{t_1}$$

is parallel to $\mathbf{v_1}$ but not unit norm. Since c is the normalization factor we can write

$$\|\mathbf{v}\|^2 = \sigma^2\alpha'^2 + \tau^2\beta'^2 + 2\alpha'\beta'\gamma$$

This simplifies to:

$$\begin{aligned}\|\mathbf{v}\|^2 = {} & \big[2\mathbf{s_1}(n)\mathbf{t_1}(n)\big]\gamma^3 \\ & + \big[2\lambda(\mathbf{s_1}(n)^2 + \mathbf{t_1}(n)^2) - \tau^2(\pi\mathbf{t_1}(n)^2 + \mathbf{s_1}(n)^2)\big]\gamma^2 \\ & + \big[2\mathbf{s_1}(n)\mathbf{t_1}(n)(\lambda^2 - \pi\tau^4)\big]\gamma \\ & + \big[(\tau\lambda - \pi\tau^3)^2\mathbf{t_1}{}^2 + (\tau\lambda - \tau^3)^2\mathbf{s_1}(n)^2\pi\big]\end{aligned} \tag{5.8}$$

Recall our assumptions that $\pi > 1$ and $\gamma < \epsilon(\pi - 1)$. As γ shrinks and π grows larger than 1, the term that is constant with respect to γ dominates the expression. Further the right hand side of the constant term is always larger, and its growth is quadratic to π while the left hand side shrinks. Hence, we approximate c^2 by the dominating term in $c^2 = (\tau\lambda - \pi\tau^3)^2\mathbf{s_1}(n)^2\pi$ and $c = \tau(\lambda - \pi\tau^2)\mathbf{s_1}(n)\sqrt{\pi}$ □

Proposition 19. *If* $\pi > 1$, $\gamma < \epsilon(\pi - 1)$, *$n$ is such that* $\mathbf{t_1}(n)\sqrt{\pi} < \mathbf{s_1}(n)$ *then* $\Delta \in [(1-\gamma)(1 - \frac{\gamma}{\pi}), (1-\gamma)(1 - \frac{\gamma}{\pi - 1})]$

Proof of Proposition 19.

$$\begin{aligned}\Delta &= \frac{\tau}{c}(1-\gamma)\big[\sqrt{\pi}\mathbf{s_1}(i)(\lambda - \tau^2(\gamma + 1))\big] \\ &= \tau(1-\gamma)\frac{\sqrt{\pi}\mathbf{s_1}(i)(\lambda - \tau^2(\gamma+1))}{\tau(\lambda - \pi\tau^2)\mathbf{s_1}(i)\sqrt{\pi}} \\ &= (1-\gamma)\frac{(\lambda - \tau^2(\gamma+1))}{(\lambda - \pi\tau^2)}\end{aligned}$$

Using the bounds on λ we obtain $\Delta \in [(1-\gamma)(1 - \frac{\gamma}{\pi}), (1-\gamma)(1 - \frac{\gamma}{\pi - 1})]$ □

Proposition 11 indicates that intuitively if $\mathbf{S}$ has large L_2-norm and is geometrically well-separated from $\mathbf{T}$, then the normalized projection of $\mathbf{M}_i$ on $\mathbf{v_1}$ will be larger for $i \in R_s$ than for $i \notin R_s$ – larger by approximately $(1-\gamma)$.

To explore the implications of Proposition 11, we continue to examine the case in which $\mathbf{M}$ consists of two rank-one submatrices. In this case, $\mathbf{S} = \mathbf{M}\left(R_s, :\right)$ has rank one, and its complement $\mathbf{T} = \mathbf{M}\left(\overline{R_s}, :\right)$ also has rank equal to one. In this case, $\gamma = \gamma(\mathbf{S}, \mathbf{T}) = |\langle \mathbf{s_1}, \mathbf{t_1} \rangle|$.

We now claim that $|\langle \mathbf{v_1}, \frac{\mathbf{M}_i}{\|\mathbf{M}_i\|} \rangle|$ are either identical for all $\mathbf{M}_i$, or take one of two values:

$$|\langle \mathbf{v_1}, \frac{\mathbf{M}_i}{\|\mathbf{M}_i\|} \rangle| = \begin{cases} |\langle \mathbf{v_1}, \mathbf{s_1} \rangle|, & \text{if } i \in R_s \\ |\langle \mathbf{v_1}, \mathbf{t_1} \rangle|, & \text{otherwise} \end{cases} \tag{5.9}$$

Since $\mathbf{S}$ and $\mathbf{T}$ are rank one, $\frac{\mathbf{M}_i}{\|\mathbf{M}_i\|} = \mathbf{s_1} \, \forall i \in R_s$ and $\frac{\mathbf{M}_i}{\|\mathbf{M}_i\|} = \mathbf{t_1} \, \forall \in \overline{R_s}$. Thus, the gap between the projections of $\mathbf{S}$ and the projections of $\mathbf{T}$ is $\Delta_{\mathbf{S},\mathbf{T}} = |\langle \mathbf{v_1}, \mathbf{s_1} \rangle| - |\langle \mathbf{v_1}, \mathbf{t_1} \rangle|$. Using Proposition 11, we can say that if the specified conditions on π and γ hold, then $\Delta_{\mathbf{S},\mathbf{T}}$ is ϵ-close to $(1-\gamma)$. Hence, as long as $\mathbf{s_1}$ and $\mathbf{t_1}$ are not parallel, the gap $\Delta_{\mathbf{S},\mathbf{T}}$ between $\mathbf{S}$ and $\mathbf{T}$ with respect to $\mathbf{v_1}$ is nonzero.

General case with larger ranks: We extend our notation to deal with several singular vectors. In particular we denote by σ_i and τ_i the i-th singular values of $\mathbf{S}$ and $\mathbf{T}$. In addition we expand γ and π from the rank-one case to: $\gamma_{i,j} = \langle \mathbf{s_i}, \mathbf{t_j} \rangle$,$\pi_{i,j} = \frac{\sigma_i^2}{\tau_j^2}$. Along the same lines, the assumptions laid out in the rank-one case are extended to hold for each pair $(\mathbf{s_1}, \mathbf{t_j})$, namely $\forall \mathbf{t_j}$: $\pi_{1,j} > 1$, $\gamma_{1,j} < \epsilon(\pi_{1,j} - 1)$, and the index n is such that $\mathbf{t_j}(n)\sqrt{\pi_{1,j}} \leq \mathbf{s_1}(n)$ and have the same sign.

As in the special case we start with λ. For larger ranks, espressing λ_1 in terms of σ_1 and τ_1 becomes more complex. One way to obtain an expression for λ_1 is by using the Cayley-Hamilton formula for the adjugate.

Recall that when **S** and **T** were rank-1, i.e. **M** was rank-2, we had:

$$Tr(\mathbf{Q}) = \lambda_1 + \lambda_2 \tag{5.10}$$

$$Tr(\mathbf{Q}^2) = {\lambda_1}^2 + {\lambda_2}^2 \tag{5.11}$$

We then wrote the characteristic equation as:

$$C^2 - C(\lambda_1 + \lambda_2) + \lambda_1\lambda_2 = C^2 - C(\tau^2 + \sigma^2) + \tau^2\sigma^2 - \sigma^2\tau^2\gamma^2$$

Which in more general terms is :

$$C^2 - C(Tr(\mathbf{Q})) + \frac{1}{2}(Tr(\mathbf{Q}^2) - Tr(\mathbf{Q})^2)$$

The above is in the form of the Cayley-Hamilton formula for the adjugate of a 3×3 matrix. Following this we can express the characteristic equation for λ of a rank-$(k+r)$ matrix **M** using the adjugate formula for a $(k+r+1) \times (k+r+1)$ matrix. For example, if **T** in **M** has rank-2 and **S** has rank-1, the λ will be a root of :

$$-C^3 + Tr(\mathbf{Q})C^2 - \frac{1}{2}(Tr(\mathbf{Q}^2) - Tr(\mathbf{Q})^2)C + \frac{1}{6}(Tr(\mathbf{Q})^3 - 3Tr(\mathbf{Q})Tr(\mathbf{Q}^2) + 2Tr(\mathbf{Q}^3))$$

Although this is promising, generalizing the equation and its roots for larger ranks is complex. We leave it as an interesting connection and we do not dig into this further in this document.

In this section we derive an expression for $\mathbf{v_1}$, in terms of $\mathbf{s_i}$'s and $\mathbf{t_j}$'s.

Proposition 20. *The first right singular vector of* **M** *can be written as*

$$\mathbf{v_1}(i) = \frac{1}{c}\mathbf{C}^A_{n,i}$$

for a constant c, and $\mathbf{C}^A_{n,i}$ *as the determinant of* $\mathbf{A} = \mathbf{Q} - \mathbf{I}\lambda$ *without the n-th row and i-th column, i.e. the cofactor* $C_{ij} = (-1)^{i+j}det(A_{ij})$.

Lemma 21. *Given an $n \times n$ matrix $\mathbf{A}$, for $i \neq k$ and $j \neq k$*

$$\mathbf{A}_{i1}\mathbf{C}^{A}_{k1} + \mathbf{A}_{i2}\mathbf{C}^{A}_{k2} + ... + \mathbf{A}_{in}\mathbf{C}^{A}_{kn} = 0$$

$$\mathbf{A}_{1j}\mathbf{C}^{A}_{1k} + \mathbf{A}_{2j}\mathbf{C}^{A}_{2k} + ... + \mathbf{A}_{nj}\mathbf{C}^{A}_{nk} = 0$$

Proof of Lemma 21. Construct the matrix $\mathbf{B}$ which is the same as $\mathbf{A}$ except with row k replaced with row i. Using the cofactor expansion of determinants we can say

$$det(\mathbf{B}) = \mathbf{B}_{k1}\mathbf{C}^{B}_{k1} + \mathbf{B}_{k2}\mathbf{C}^{B}k2 + ... + \mathbf{B}_{kn}\mathbf{C}^{B}_{kn} = 0$$

The above is equal to zero because by replacing row k with row i we created two identical rows in $\mathbf{B}$ making it singular. Further, since the kth row was the only one changed, $\mathbf{C}^{B}_{k\ell} = \mathbf{C}^{A}_{k\ell} \forall \ell$, in addition $\mathbf{B}_{k1} = \mathbf{A}_{i1}$ so

$$\mathbf{B}_{k1}\mathbf{C}^{B}_{k1} + \mathbf{B}_{k2}\mathbf{C}^{B}_{k2} + ... + \mathbf{B}_{kn}\mathbf{C}^{B}_{kn} = \mathbf{A}_{i1}\mathbf{C}^{A}_{k1} + \mathbf{A}_{i2}\mathbf{C}^{A}_{k2} + ... + \mathbf{A}_{in}\mathbf{C}^{A}_{kn} = 0$$

as desired. $\square$

Proof of Proposition 20. The first singular vector $\mathbf{v_1}$ of $\mathbf{M}$ will be in the null space of $\mathbf{A}$, i.e. $\mathbf{A}\mathbf{v_1} = 0$. We show that the inner product of each row $\mathbf{A}_i$ of $\mathbf{A}$ with $\mathbf{v_1}$ is zero when the latter has the cofactor form specified in Proposition 20. For $i \in [1, n-1]$, $\mathbf{A}_i\mathbf{v_1} = 0$ is true because of Lemma 21. For $i = n$, $\mathbf{A}_i\mathbf{v_1}$ is the expression for the determinant of $\mathbf{A}$, whole is zero by the assumption that $\mathbf{A}$ has a nullspace. $\square$

We conjecture that deeper analysis of the cofactors of $\mathbf{M}$ in terms of $\mathbf{s_i}$'s and $\mathbf{t_j}$'s can reveal a general expression for $\mathbf{v_1}$. However, we take a different approach by approximating $\mathbf{v_1}$ by a vector $\hat{\mathbf{v_1}}$ which is a linear combination of $\mathbf{s_i}$'s and $\mathbf{t_j}$'s. We show that $\hat{\mathbf{v_1}}$ is nearly parallel to $\mathbf{v_1}$ by showing that it is in the approximate null space of $\mathbf{Q}$.

Proposition 22. *If* $\pi_{1,1} > 1$, $\gamma_{ij} < \epsilon(\pi_{ij} - 1)$, *the index* n *is such that* $\mathbf{t_j}(n)\sqrt{\pi_{1j}} < \mathbf{s_1}(n)$ *for each* $\mathbf{t_j}$, *and* $\mathbf{S}$ *is nondegenerate, then the first right singular vector of* $\mathbf{A}$ *can be approximated by*

$$\hat{\mathbf{v_1}} = \frac{1}{c}\big[\sum_{\mathbf{s_i}} \alpha(i)\mathbf{s_i} + \sum_{\mathbf{t_j}} \beta(j)\mathbf{t_j}\big]$$

For $\alpha_i = \sigma_i\big[\mathbf{s_i}(n)(\lambda - \sum_{\mathbf{s_j}\neq\mathbf{s_i}} \sigma_j^2 - \sum_{\mathbf{t_j}} \tau_j^2) + \sum_{\mathbf{t_j}} \mathbf{t_j}(n)\sigma_i\tau_j\gamma_{ij}\big]$ *and* $\beta_j = \tau_j\big[\mathbf{t_j}(n)(\lambda - \sum_{\mathbf{t_k}\neq\mathbf{t_j}} \tau_k^2 - \sum_{\mathbf{s_i}} \sigma_i^2) + \sum_{\mathbf{s_i}} \mathbf{s_i}(n)\sigma_i\tau_j\gamma_{ij}\big]$.

Lemma 23. *If* $\pi_{1,1} > 1$, $\gamma_{ij} < \epsilon(\pi_{ij} - 1)$, n *is such that* $\mathbf{t_j}(n)\sqrt{\pi_{1j}} < \mathbf{s_1}(n)$ *for each* $\mathbf{t_j}$ *the normalization factor* $c > |\sum_{\mathbf{s_i}} \alpha_i + \sum_{\mathbf{t_j}} \beta_j|$

Proof of Lemma 23. For $\hat{\mathbf{v}} = c\hat{\mathbf{v_1}} = \big[\sum_{\mathbf{s_i}} \alpha(i)\mathbf{s_i} + \sum_{\mathbf{t_j}} \beta(j)\mathbf{t_j}\big]$, we compute $\|\hat{\mathbf{v}}\|^2 = c^2$.

$$\|\hat{\mathbf{v}}\|^2 = c^2 = \big[\sum_{\mathbf{s_i}} \alpha_i\mathbf{s_i} + \sum_{\mathbf{t_j}} \beta_j\mathbf{t_j}\big]^2 \tag{5.12}$$

$$> \sum_{\mathbf{s_1}} \alpha_i^2 + \sum_{\mathbf{t_j}} \beta_j^2 \tag{5.13}$$

$$c > \sqrt{\sum_{\mathbf{s_i}} \alpha_i^2 + \sum_{\mathbf{t_j}} \beta_j^2} \tag{5.14}$$

$$> \frac{1}{\sqrt{k+r}}(\sum_{\mathbf{s_i}} \alpha_i + \sum_{\mathbf{t_j}} \beta_j) \tag{5.15}$$

□

The introduction of the inequality from (5.12) to (5.13) is due to the fact that we consider γ_{ij} to be positive. The step from (5.14) to (5.15) uses the CauchySchwarz inequality. Unlike in the rank-one case we do not prove that $\mathbf{v_1}$ is exactly in the null space of $\mathbf{Q}$. Instead, we show that $\hat{\mathbf{v_1}}$ is nearly parallel to the true $\mathbf{v_1}$ by showing that $\hat{\mathbf{v_1}}$ is in the approximate null space of $\mathbf{Q}$, i.e. $\|\mathbf{Q}\hat{\mathbf{v_1}}\| < \epsilon$. We do this by showing that the coefficient in $\mathbf{Q}\hat{\mathbf{v_1}}$ for each vector in $\{\mathbf{s_1} \ldots \mathbf{s_k}, \mathbf{t_1} \ldots \mathbf{t_r}\}$ is small.

Lemma 24. *Given* $\mathbf{Q} = \mathbf{M}^T\mathbf{M}$ *with eigenvalues* $\Lambda = \lambda_1 > \cdots > \lambda_{(k+r)}$*, the eigenvalues of* $(\mathbf{Q} - \mathbf{I}p)$ *are* $\{|\lambda_i - p|\}$*. Hence the smallest eigenvalue of* $(\mathbf{Q} - \mathbf{I}p)$ *is* $\min_i(p, |\lambda_i - p|)$*.*

Thus, to show that $\hat{\mathbf{v_1}}$ is in the approximate null space we need to show that $\|M\hat{\mathbf{v_1}}\|$ is less than the smallest eigenvalue of $\mathbf{Q}$, i.e. $\|M\hat{\mathbf{v_1}}\| \leq \min(|\lambda_i - \lambda|) = \lambda - \lambda_2$. Without loss of generality we consider α_i's and β_j's to have the same sign by assuming the particular entries $\mathbf{s_i}(n)$ and $\mathbf{t_j}(n)$ have the same sign.

Proof of Proposition 22.

$$\begin{aligned}
\mathbf{Qv_1} &= (\sum_{\mathbf{s_i}} \sigma_i^2 \mathbf{s_i} \otimes \mathbf{s_i}^T + \sum_{\mathbf{s_i}} \tau_j^2 \mathbf{t_j} \otimes \mathbf{t_j}^T - I\lambda)\frac{1}{c}[\sum_{\mathbf{s_i}} \alpha(i)\mathbf{s_i} + \sum_{\mathbf{t_j}} \beta(j)\mathbf{t_j}] \\
&= \frac{1}{c}\sum_{\mathbf{s_i}} \mathbf{s_i}(\alpha_i(\sigma_i^2 - \lambda) + \sum_{\mathbf{t_j}} \gamma_{ij}\sigma_i^2\beta_j) + \frac{1}{c}\sum_{\mathbf{t_j}} \mathbf{t_j}(\beta_j(\tau_j^2 - \lambda) + \sum_{\mathbf{s_i}} \gamma_{ij}\tau_j^2\alpha_i) \\
&= \sum_{\mathbf{s_i}} \mathbf{s_i}\xi_{\mathbf{s_i}} + \sum_{\mathbf{t_j}} \mathbf{t_j}\xi_{\mathbf{t_j}}
\end{aligned}$$

Take $\nu = \lambda_1 - \lambda_2$.

$$\xi_{\mathbf{s_i}} < \left|\frac{\alpha_i(\sigma_i^2 - \lambda) + \sum_{\mathbf{t_j}} \gamma_{ij}\sigma_i^2\beta_j}{\frac{1}{\sqrt{k+r}}(\sum_{\mathbf{s_i}} \alpha_i + \sum_{\mathbf{t_j}} \beta_j)}\right| = \frac{\alpha_i(\lambda - \sigma_i^2) + \sum_{\mathbf{t_j}} \gamma_{ij}\sigma_i^2\beta_j}{\frac{1}{\sqrt{k+r}}(\sum_{\mathbf{s_i}} \alpha_i + \sum_{\mathbf{t_j}} \beta_j)}$$

We show that each of the coefficients ξ is small for each $\mathbf{s_i}$ and $\mathbf{t_j}$:

$$\begin{aligned}
\xi_{\mathbf{s_i}} &< \frac{\alpha_i(\lambda - \sigma_i^2) + \sum_{\mathbf{t_j}} \gamma_{ij}\sigma_i^2\beta_j}{\frac{1}{\sqrt{k+r}}(\sum_{\mathbf{s_i}} \alpha_i + \sum_{\mathbf{t_j}} \beta_j)} \\
&< \frac{2\alpha_i(\lambda - \sigma_i^2)}{\frac{1}{\sqrt{k+r}}(\sum_{\mathbf{s_i}} \alpha_i + \sum_{\mathbf{t_j}} \beta_j)} \\
&< 2(\lambda - \sigma_i^2) < \frac{2(\lambda - \nu^2)}{2\pi_{1,1}} = \frac{(\lambda - \nu^2)}{\pi_{1,1}}
\end{aligned}$$

Each of the inequalities is strict and so we arrive at $\xi_{\mathbf{s_i}} \ll (\lambda - \nu^2)$ Similarly for each β_j. By showing each coefficient is small we can bound $\|M\hat{\mathbf{v_1}}\| \ll \lambda_{\min}(\mathbf{Q} - \mathbf{I}\lambda_1)$ meaning that $\hat{\mathbf{v_1}}$ is in the approximate null space of $\mathbf{M}$. $\square$

Given the expression for $\hat{\mathbf{v_1}}$, we study the resulting inner products with $\mathbf{s_i}$'s and $\mathbf{t_j}$'s. Recall that the inner product in the rank-1 case had the form:

$$\langle \mathbf{v_1}, \mathbf{s_1} \rangle = \frac{\tau}{c}\big[\lambda(\sqrt{\pi}\mathbf{s_1}(n) + \gamma \mathbf{t_1}(n)) + \mathbf{s_1}(n)\tau^2\sqrt{\pi}(\gamma^2 - 1)\big]$$

Proposition 25. *If* $\pi_{1,1} > 1$, $\gamma_{ij} < \epsilon(\pi_{ij} - 1)$, n *is such that* $\mathbf{t_j}(n)\sqrt{\pi_{1j}} < \mathbf{s_1}(n)$, *then for a vector* $\mathbf{x_i} \in X = \{\mathbf{s_1} \ldots \mathbf{s_k}, \mathbf{t_1} \ldots \mathbf{t_r}\}$ *and* ν_i *as the associated singular value, we can approximate* $\langle \hat{\mathbf{v_1}}, \mathbf{x_i} \rangle$ *by*

$$\big[\frac{\nu_i}{c}\mathbf{x_i}(n)(\lambda - \sum_{\mathbf{x_j} \neq \mathbf{x_i}} \nu_j^2(\gamma_{\mathbf{x_i},\mathbf{x_j}}^2 - 1)) + \frac{1}{c}\sum_{\mathbf{x_j} \neq \mathbf{x_i}} \nu_j \mathbf{x_j}(n)\gamma_{\mathbf{x_i},\mathbf{x_j}}(\lambda - \sum_{\mathbf{x_j} \neq \mathbf{x_i}, \mathbf{x_j}} \nu_j^2)\big]$$

Following the rank-one case we make smilar statements about c as the normalization factor for the singular vector. This factor c will be useful in the coming steps where we consider the inner products and Δ.

Lemma 26. *If* $\pi_{1,1} > 1$, $\gamma_{ij} < \epsilon(\pi_{ij} - 1)$, n *is such that* $\mathbf{t_j}(n)\sqrt{\pi_{1j}} < \mathbf{s_1}(n)$, *then* $c = \mathbf{s_1}(n)\sigma_1^2(\lambda - \sum_{\mathbf{t_j}} \tau_j^2 - \sum_{\mathbf{s_i} \neq \mathbf{s_1}} \sigma_i^2)$.

Proof of Lemma 26.

$$|\hat{\mathbf{v_1}}\|^2 = \sum_{\mathbf{s_i}} \alpha_i^2 + \sum_{\mathbf{t_j}} \beta_j^2 + 2\sum_{(i,j)} (\alpha_i + \beta_j)\gamma_{ij}$$

As in the rank-one case, as γ_{ij} shrinks and π_{ij} grows, the terms that are constant with respect to γ_{ij}'s and polynomial in π_{ij}'s will dominate. Hence the dominant term in $\hat{\mathbf{v_1}}$ is $\mathbf{s_1}(n)^2\sigma_1^2(\lambda - \sum_{\mathbf{t_j}} \tau_j^2 - \sum_{\mathbf{s_i} \neq \mathbf{s_1}} \sigma_i^{2^2})$ and $c = \mathbf{s_1}(n)\sigma_1(\lambda - \sum_{\mathbf{t_j}} \tau_j^2 - \sum_{\mathbf{s_i} \neq \mathbf{s_1}} \sigma_i^2)$.

Proof.

$$\begin{aligned}
\langle \hat{\mathbf{v_1}}, \mathbf{s_1} \rangle &= \frac{1}{c}\alpha_1 + \frac{1}{c}\sum_{\mathbf{t_j}} \gamma_{ij}\beta_j \\
&= \frac{1}{c}\mathbf{s_1}(n)\sigma_1(\lambda - \sum_{\mathbf{s_i} \neq \mathbf{s_1}} \sigma_i^2 + \sum_{\mathbf{t_j}} \tau_j^2(\gamma_{1j}^2 - 1)) \\
&\quad + \frac{1}{c}\sum_{\mathbf{t_j}} \mathbf{t_j}(n)\gamma_{1j}\tau_j(\lambda - \sum_{\mathbf{s_i} \neq \mathbf{s_1}} \sigma_i^2 - \sum_{\mathbf{t_q} \neq \mathbf{t_j}} \tau_q^2) + O(\gamma^2) \\
&> \frac{\tau}{c}\big[\mathbf{s_1}(n)\sqrt{\pi_{1,1}}(\lambda - \sum_{\mathbf{s_i} \neq \mathbf{s_1}} \sigma_i^2 + \sum_{\mathbf{t_j}} \tau_j^2(\gamma_{1j}^2 - 1)) \\
&\quad + \frac{1}{c}\sum_{\mathbf{t_j}} \mathbf{t_j}(n)\gamma_{1j}\frac{\pi_{1,1}}{\pi_{1,j}}(\lambda - \sum_{\mathbf{s_i} \neq \mathbf{s_1}} (\sigma_i^2 - \sum_{\mathbf{t_q} \neq \mathbf{t_j}} \tau_q^2))\big]
\end{aligned}$$

The above is only in terms of $\mathbf{s_1}$, but it generalizes directly to any other $\mathbf{s_i}$ or $\mathbf{t_j}$ because of the the symmetric nature of α_i's and β_j's. □

In this section we show the final step in which we simplify an expression for Δ. We start by considering $\Delta_{\mathbf{s_1},\mathbf{t_j}}$ – the gap between the singular vectors on $\mathbf{S}$ and $\mathbf{T}$ with respect to $\mathbf{v_1}$. As in the rank-1 case, we show that $\Delta_{\mathbf{s_1},\mathbf{t_j}}$ is tied to a factor of $1-\gamma$. However, unlike in the rank-one case, the expression for $\Delta_{\mathbf{s_1},\mathbf{t_j}}$ is not equivalent to that of $\Delta_{\mathbf{S},\mathbf{T}}$, which is the on in Proposition 11. The reason is that it is no longer true that $\langle \mathbf{v_1}, \frac{\S_i}{\|\mathbf{S}_i\|} \rangle = \langle \mathbf{v_1}, \mathbf{s_1} \rangle$. Hence after $\Delta_{\mathbf{s_1},\mathbf{t_j}}$ we also consider the expression for $\Delta_{\mathbf{S},\mathbf{T}}$, and show that the gap in the data $\Delta_{\mathbf{S},\mathbf{T}}$ will also contain a factor of $(1 - \gamma)$.

Spectral gap: First we show that $\Delta_{\mathbf{s_i},\mathbf{t_j}} > (1 - \gamma_{ij})(1 - \epsilon)$. The idea mirrors that of the rank-one case, where we consider the difference in inner products with $\mathbf{v_1}$ and show that we obtain a factor of $(1 - \gamma_{ij})$.

Proposition 27. *If* $\pi_{1,1} > 1$, $\gamma_{ij} < \epsilon(\pi_{ij} - 1)$, n *is such that* $\mathbf{t_j}(n)\sqrt{\pi_{1j}} < \mathbf{s_1}(n)$, *then for each* $\mathbf{t_j}$ *Proposition 11 applies, i.e.* $\Delta_{\mathbf{s_1},\mathbf{t_j}} > (1 - \gamma_{1,j})(1 - \epsilon)$.

This can be shown by following the same rearrangements and manipulation as in the rank-1 section.

Proof.

$$\begin{aligned}
\Delta_{\mathbf{s_i},\mathbf{t_j}} &= |\langle \mathbf{v_1}, \mathbf{s_1}\rangle| - |\langle \mathbf{v_1}, \mathbf{t_j}\rangle| = |\langle \hat{\mathbf{v_1}}, \mathbf{s_1}\rangle| - |\langle \hat{\mathbf{v_1}}, \mathbf{t_j}\rangle| \\
&= \frac{1}{c}\big[\mathbf{s_1}(n)\sigma_1(\lambda - \sum_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2 + \sum_{\mathbf{t_j}} \tau_j^2(\gamma_{1j}^2 - 1)) + \sum_{\mathbf{t_j}} \mathbf{t_j}(n)\gamma_{1j}\tau_j(\lambda - \sum_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2 - \sum_{\mathbf{t_q}\neq\mathbf{t_j}} \tau_q^2) \\
&\qquad - \mathbf{t_j}(n)\tau_j(\lambda - \sum_{\mathbf{t_q}\neq\mathbf{t_j}} \tau_q^2 + \sum_{\mathbf{s_i}} \sigma_i^2(\gamma_{ij}^2 - 1)) + \sum_{\mathbf{s_i}} \mathbf{s_i}(n)\gamma_{ij}\sigma_j(\lambda - \sum_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2 - \sum_{\mathbf{t_q}\neq\mathbf{t_j}} \tau_q^2)\big] \\
&> \frac{\tau(1-\gamma_{1j})\big[\mathbf{s_1}(n)\sqrt{\pi_{1,j}}(\lambda - \sum\limits_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2 - \sum\limits_{\mathbf{t_q}\neq\mathbf{t_j}} \tau_q^2 - \tau_j^2(\gamma_{1j}+1))\big]}{\mathbf{s_1}(n)\sigma_1(\lambda - \sum\limits_{\mathbf{t_j}} \tau_j^2 - \sum\limits_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2)} \\
&\qquad + \frac{\tau(1-\gamma_{1j})\big[\mathbf{t_j}(n)(\lambda - \sum\limits_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2 - \sum\limits_{\mathbf{t_q}\neq\mathbf{t_j}} \tau_q^2 - \tau_j^2\pi_{1j}(\gamma_{1j}+1))\big]}{\mathbf{s_1}(n)\sigma_1(\lambda - \sum\limits_{\mathbf{t_j}} \tau_j^2 - \sum\limits_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2)} \\
&> \frac{\tau(1-\gamma_{1j})\big[\mathbf{s_1}(n)\sqrt{\pi_{1,j}}(\lambda - \sum_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2 - \sum_{\mathbf{t_q}\neq\mathbf{t_j}} \tau_q^2 - \tau_j^2(\gamma_{1j}+1))\big]}{\mathbf{s_1}(n)\sigma_1(\lambda - \sum_{\mathbf{t_j}} \tau_j^2 - \sum_{\mathbf{s_i}\neq\mathbf{s_1}} \sigma_i^2)} \\
&= (1-\gamma_{1j})(1 - \frac{\tau_j^2\gamma_{1j}}{\lambda - \tau_j^2}) > (1-\gamma_{1j})(1 - \frac{\gamma_{1j}}{\pi_{1j} - 1}) \qquad (5.16)
\end{aligned}$$

□

The majority of steps are results of simplification, and the final inequality in (5.16) appears since $\pi > 1$ implies that $\lambda > \pi\tau^2$. We now have that if γ_{ij} is small and π_{ij} is large, then $\Delta_{\mathbf{s_1},\mathbf{t_j}}$ is large.

Data gap: Now we have show the gap Δ with respect to $\mathbf{s_i}$ and $\mathbf{t_j}$; however, unlike the rank-one case this is not equivalent to $\Delta_{\mathbf{S},\mathbf{T}}$. In particular, recall that the definition of $\Delta_{\mathbf{S},\mathbf{T}}$ includes inner products with the rows of $\mathbf{A}$, which are not necessarily parallel to the singular vectors as in the rank-one case. Hence, $\Delta_{\mathbf{S},\mathbf{T}} = \frac{1}{|R_s|}\sum_{i\in R_s}\langle \mathbf{v_1}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle - \frac{1}{|R_t|}\sum_{i\in R_t}\langle \mathbf{v_1}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle$ is no longer as directly expressed in terms of $\Delta_{\mathbf{s_1},\mathbf{t_j}}$.

Conjecture 28. *If* $\mathbf{S}$ *is nondegenerate, then there exists a subset of rows* $\mathbf{S}$*' of* $\mathbf{S}$ *whose average inner product with* $\hat{\mathbf{v_1}}$ *is at least* $1 - e$ *for small* e

The definition of nondegeneracy implies that each subset of $\leq k$ rows (or columns) is independent. In particular this means that each subset of $\geq k$ rows (or columns) spans all k singular vectors $\mathbf{s_i}$. In addition since $\mathbf{s_1}$ is the *first* singular vector, it is the most strongly expressed direction in $\mathbf{S}$. Putting all this together we claim that there is a subset of rows $\mathbf{S}$' of $\mathbf{S}$ that strongly express $\mathbf{s_1}$ (i.e. have high inner product with $\mathbf{s_1}$) and $|\mathbf{S}'| \geq k$.

Proposition 29. *If the rank-ℓ submatrix $\mathbf{S}$ is nondegenerate, $\pi > 1$, and $\gamma = \epsilon(1-\pi)$, then $\exists R_{s'} \subset R_s$ with $|R_{s'}| \geq \ell$ such that $\Delta_{\mathbf{S'},\mathbf{T}}$ is ϵ-close to $1-\gamma$.* Note that we say that an $n \times m$ rank r matrix is *nondegenerate* if there is no dependent subset of rows or columns of size less than $(r+1)$.

Proof of Proposition 29. Assume $\frac{1}{|R_t|}\sum_i \sum_j \langle \mathbf{t_j}, \frac{\mathbf{T}_i}{\|\mathbf{T}_i\|}\rangle \leq 1$ and denote by $\mathbf{t}_*= \arg\max_{\mathbf{t_j}} \langle \mathbf{v_1}, \mathbf{t_j}\rangle$. Further recall that $\Delta_{\mathbf{s_i},\mathbf{t_j}} > (1-\gamma_{ij})(1-\epsilon)$ and so $\langle \hat{\mathbf{v_1}}, \mathbf{t_j}\rangle < \langle \hat{\mathbf{v_1}}, \mathbf{s_i}\rangle - (1-\gamma_{ij})(1-\epsilon)$.

$$\begin{aligned}
\Delta_{\mathbf{S'},\mathbf{T}} &= \frac{1}{|R_{s'}|}\sum_{i\in R_{s'}} \langle \mathbf{v_1}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle - \frac{1}{|R_t|}\sum_{i\in R_t} \langle \mathbf{v_1}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle && (5.17)\\
&= (1-e)\langle \mathbf{v_1}, \mathbf{s_1}\rangle - \frac{1}{|R_t|}\sum_{i\in R_t} \langle \mathbf{v_1}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle && (5.18)\\
&= (1-e)\langle \mathbf{v_1}, \mathbf{s_1}\rangle - \frac{1}{|R_t|}\sum_{i\in R_t} \langle \mathbf{v_1}, \sum_j \mathbf{t_j}\langle \mathbf{t_j}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle\rangle && (5.19)\\
&= (1-e)\langle \mathbf{v_1}, \mathbf{s_1}\rangle - \frac{1}{|R_t|}\sum_{i\in R_t}\sum_{\mathbf{t_j}} \langle \mathbf{v_1}, \mathbf{t_j}\rangle\langle \mathbf{t_j}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle && (5.20)\\
&= (1-e)\langle \mathbf{v_1}, \mathbf{s_1}\rangle - \frac{1}{|R_t|}\sum_{\mathbf{t_j}} \langle \mathbf{v_1}, \mathbf{t_j}\rangle \sum_{i\in R_t}\langle \mathbf{t_j}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle && (5.21)\\
&> (1-e)\langle \mathbf{v_1}, \mathbf{s_1}\rangle - \frac{1}{|R_t|}\langle \mathbf{v_1}, \mathbf{t}_*\rangle \sum_{\mathbf{t_j}}\sum_{i\in R_t}\langle \mathbf{t_j}, \frac{\mathbf{A}_i}{\|\mathbf{A}_i\|}\rangle && (5.22)\\
&> (1-e)\langle \hat{\mathbf{v_1}}, \mathbf{s_1}\rangle - \langle \hat{\mathbf{v_1}}, \mathbf{t}_*\rangle && (5.23)\\
&> (1-e)\langle \hat{\mathbf{v_1}}, \mathbf{s_1}\rangle - \langle \hat{\mathbf{v_1}}, \mathbf{s_1}\rangle + (1-\gamma_{1,*})(1-\epsilon) > (1-\gamma_{1,*})(1-\epsilon) && (5.24)
\end{aligned}$$

□

Discussion: Recall that in the above discussion we focused on the case where $\mathbf{S} = \mathbf{M}(R_s, :)$. The results generalize to the case where $\mathbf{S} = \mathbf{M}(R_s, C_s)$, for a subset of the columns C_s, by applying the above analysis recursively on $\mathbf{M}(R_s, :)$ and $\mathbf{M}(:, C_s)$. Hence, we have expressed conditions on a low-rank submatrix to be discoverable by the first singular vectors of a matrix. In real data, low-rank submatrices correspond to subsets of the data which have a particularly dependent relationship only within the subset; thus, a low-rank submatrix is likely to be well-separated from the rest of the data. Further, in the common case where a dataset meets the traditional incoherence assumptions on the distribution of the values, a low-rank submatrix is also very likely to be separated in norm and discoverable by `SVP`. We also conjecture that a more general and symmetric analysis can be carried out for when $\pi \ll 1$.

5.3.2 An algorithm for extracting low-rank submatrices

In this section, we provide a simple and effective algorithm for finding a low-rank submatrix $\mathbf{S}$, that is planted in a larger-rank matrix $\mathbf{M}$. Our algorithm builds upon the results of the previous section, which imply the following: the projections of $\mathbf{M}_i$ on $\mathbf{v_1}$ for $i \in R_s$ will have larger values than the corresponding projections of $\mathbf{M}_i$ for $i \notin R_s$. Hence, a straightforward way to find $\mathbf{S}$ is to project $\mathbf{M}$ onto $\mathbf{v_1}$ and partition the projections into high and low values. This is exactly the idea behind our algorithm for finding $\mathbf{S}$. The pseudocode of this algorithm, which we call `SVP` standing for Singular Vector Projection, is given below:

1. $\mathbf{p} = \mathbf{Project}(\mathbf{M}, \mathbf{v_1})$.

   In this step, the algorithm computes vector $\mathbf{p} = \{p_1, \ldots, p_n\}$, where $p_i = \langle \mathbf{v_1}, \frac{\mathbf{M}_i}{\|\mathbf{M}_i\|} \rangle$.

2. $\{R_s, \overline{R_s}\} = \mathbf{Partition}(p)$

   Partition the rows of $\mathbf{M}$ into two groups based on the corresponding values in

$\mathbf{p}$. Identify the set of rows R_s to be the group that has the largest average of the corresponding p_i values.

3. **Repeat** steps (1) and (2) on $\mathbf{M}^T$ to find C_s.

The `SVP` algorithm consists of two main steps (applied independently to the rows and the columns of $\mathbf{M}$). In the first step, the rows (resp. columns) of $\mathbf{M}$ are (normalized and) projected on $\mathbf{v_1}$. Then, these 1-dimensional points that correspond to the projections are clustered into two clusters – in our implementation we use k-means to obtain this clustering. The cluster of rows (resp. columns) with the largest mean are the ones assigned to $\mathbf{S}$.

A reasonable extension is to create a multi-dimensional $\mathbf{p}_i$'s by projecting into more than one of the singular vectors of $\mathbf{M}$. Our experiments indicated that on average using three singular vectors improved the accuracy of our results and had a minor affect the efficiency of our method; using more than three singular vectors did give any significant improvement.

To find multiple low-rank submatrices, `SVP` can be run on the matrix formed by removing the rows and columns of $\mathbf{S}$, i.e. $\mathbf{M}(R_t, C_t)$. If the number of low-rank submatrices in $\mathbf{M}$ is known apriori, `SVP` can be repeated exactly this many times. Otherwise, the difference in the average projection between the output partitions, $\Delta_{\hat{\mathbf{S}},\hat{\mathbf{T}}}$, can be used as a stopping criterion where $\hat{\mathbf{S}}$ is the output of `SVP`. For example, allow the algorithm to keep iterating while $\Delta_{\hat{\mathbf{S}},\hat{\mathbf{T}}}$ is greater than a threshold, and stop when it falls below; in our experiments, we found 0.2 to work well. Another possibility is to replace the submatrix with random noise that is small relative to the values in the rest of the data and rerun on the modified matrix; such an approach would allow for the discovery of submatrices that overlap in rows or columns.

Interestingly, the `SVP` algorithm bears resemblances to the algorithm for the Planted Clique problem presented by Alon et al. (1998); both use some of the singular vectors of a matrix to discover a submatrix with a particular property. Although a perfect clique is a rank-one submatrix of the adjacency matrix, a low-rank submatrix is not necessarily a clique. Further, the algorithm developed by Alon et al. (1998) relies on assumptions on the node degrees that allows bounds on the gap between the second and third eigenvalues. Not only is it unclear that such assumptions hold in our settings, but it is also not clear whether they apply to general low-rank submatrices. Without our analysis, there is not apparent that the singular vectors can be used to address LRDiscovery.

5.3.3 Improvements for matrices with missing entries

Up to now, we have presented an algorithm called `SVP` for finding low-rank submatrices in a fully-known matrix and analyzed the conditions under which it is expected to succeed. With our end-goal of matrix-completion in mind, we now improve the performance of `SVP` on incomplete data and demonstrate that the high level of accuracy is retained.

The main tool that `SVP` relies on is finding the first singular vectors of a matrix. Since the Singular Value Decomposition of an incomplete matrix is not defined, extending `SVP` to handle incomplete data translates to developing an approach to estimating the singular vectors of a matrix with missing entries. The majority of the related literature examines finding the full Singular Value Decomposition of a partially observed matrix in the context of producing an accurate factorization of the data. Typically the accuracy is related to the error in the entries of the estimate matrix produced by the factorization. As an aside, since `SVP` needs only the first singular vectors and uses them for inner products, our objective is to accurately estimate the direction of the singular vectors as opposed to the values of the matrix per-se; unfortunately, we are not aware of any algorithms tailored towards our needs. The algorithms that do exist can be broadly categorized into (1) those that treat the missing entries as knowns, and (2) those that try to ignore the missing values. We highlight three approaches in particular:

> `Simple`: The simplest approach falls into the first category and consists of treating the missing entries as zeros and applying the same `SVD`-algorithm as one would on a fully-observed matrix.

However, treating the missing entries as knowns runs the danger of over-fitting these entries and producing singular vectors whose direction differs greatly from that of the singular vectors of the fully-known matrix. To avoid over-fitting, we implement a heuristic built around a QR-based Power Method for computing the singular vectors as in Bentbib and Kanber (2015) (the power method starts from an estimate singular vector and improves the estimate until convergence.)

> `EarlyStop`: The main idea of `EarlyStop` is, as the name suggests, to stop the power method before it converges and over-fits the missing entries. To decide the stopping point we calculate the projections $\mathbf{p}$ at each intermediate iteration, partition them as in `SVP`, and measure the difference in average value between the two partitions ($\Delta_{\mathbf{S},\mathbf{T}}$). The power method is stopped when the $\Delta_{\mathbf{S},\mathbf{T}}$ does not change for five consecutive iterations.

Alternatively the missing values can be ignored altogether. While various methods have been proposed in the literature, we focus on one that was used by Simon Funk (Brandyn Webb) in the Netflix Prize Challenge detailed in Funk (2006).

> `Incremental`: This line of work features algorithms designed for matrices in online settings that allow for the `SVD` to be developed using only parts of the data. We use an implementation based Sequential Karhunen-Loeve provided in Baker (2012).

We observe that all three methods described above are equally efficient. With a slight trade off of efficiency for robustness we choose to combine all three.

> `Consensus:` For each of `Simple`, `EarlyStop`, and `Incremental`, produce a partitioning $\mathbf{p}$, and then aggregate the three partitionings into a single one by using clustering aggregation as in Iam-on and Garrett (2010).

In our experiments we observed the performance of the algorithms was generally similar but not one was consistently accurate. For the sake of robustness we adapt `SVP` to use `Consensus` when the input data is incomplete. We provide a comparison analysis in the next section, and add a side that note that if efficiency is a strong concern we recommend simply using `EarlyStop` or `Incremental`. The overall routine becomes:

1. $\mathbf{v_1}$=`Consensus`($\mathbf{M}_\Omega$)
2. $\mathbf{p}$ =**Project**($\mathbf{M}_\Omega, \mathbf{v_1}$).
   Compute vector $\mathbf{p} = \{p_1, \ldots, p_n\}$, where $p_i = \langle \mathbf{v_1}, \frac{\mathbf{M}_{\Omega i}}{\|\mathbf{M}_{\Omega i}\|} \rangle$.
3. $\{R_s, \overline{R_s}\}$ = **Partition**(p)
   Partition the rows of $\mathbf{M}_\Omega$ into two groups based on the corresponding values in $\mathbf{p}$. Identify the set of rows R_s to be the group that has the largest average of the corresponding p_i values.
4. **Repeat** steps (1) and (2) on $\mathbf{M}_\Omega^T$ to find C_s.

The extension to finding multiple submatrices follows exactly the technique for fully-known matrices: after the first submatrix is found, it is removed, and the algorithm continues on the remaining data. For a refresher we point the reader to Section 4.3.4.

5.4 Experiments

In this section we answer the question posed in Section 5, namely: *Can the knowledge of the existence of a low-rank submatrix improve the accuracy of completion?* We do this by evaluating the performance of the proposed `Localized` approach to matrix completion on datasets with missing entries, and demonstrate the large improvement in accuracy over the `Global` approach in both the low-rank submatrix and its complement. After this main set of experiments, we study the behavior of `SVP` (step 1 in the `Localized` approach) over a wide range of matrix instances, and show that `SVP` succeeds in more cases than the comparison algorithms.

Setup: Throughout the experiments we generate a 1000×1000 random matrix $\mathbf{M}$ of rank r with entries following a standard gaussian distribution (mean zero and variance one), we call this the *background matrix*. We then plant a 100×100 submatrix $\mathbf{S}$ of rank ℓ with entries adhering to a centered Gaussian distribution. We maintain π constant at $\pi = 1.2$; as this value is only slightly larger than 1, the instances we generate are hard for `SVP`.

5.4.1 Evaluation of localized matrix-completion

To quantify the improvement in matrix completion when using a `Localized` versus a `Global` approach, we compare the accuracy of the completions produced by each. We setup the experiments as described above and set the rank of the matrix $\mathbf{M}$ to low-rank $r = 30$ and the rank of the submatrix $\mathbf{S}$ to a lower rank $\ell = 2$. To form the partially observed $\mathbf{M}_\Omega$ we vary the percentage of observed entries from 0% to 100% , and measure the relative error (RelError) of each completion according to Equation 5.1 in Section 5.1.

We compare two approaches for completing each partially observed $\mathbf{M}_\Omega$:

`Global`:

1. Run `LMaFit` on $\mathbf{M}_\Omega$.

`Localized`:

1. Run `SVP` on $\mathbf{M}_\Omega$ to find $\mathcal{S}$.
2. Separate the entries of the low-rank submatrices from the rest $\Omega' = \Omega \setminus \mathcal{S}$
3. Run `LMaFit` on $\mathbf{M}_\mathcal{S}$ and separately on $\mathbf{M}_{\Omega'}$
4. Combine the estimates back into a matrix $\hat{\mathbf{M}}$ as the completion

For the actual completion we focus on one state-of-art *global* matrix-completion algorithm called `LMaFit` for its accuracy, efficiency, and ability to estimate the rank. Step 2 and 3 is implemented by running the completion algorithm on each submatrix $\mathbf{M}_\Omega(R_s, C_s)$ and then on $\mathbf{M}_\Omega$ with the entries in $\mathcal{S}$ set to zero. For the latter (the completion of the complement) we experimented with other approaches such as replacing the entries in $\mathcal{S}$ with the computed $\hat{\mathbf{S}}$ or alternatively small random values, but replacing with zeros worked best. This makes sense since the idea is to run completion on separate components so that the algorithm does not try to fit both. Figure 5·1 shows the relative error over $\hat{\mathbf{S}}$ (left) and over $\hat{\mathbf{M}}$ (right). Immediately we observe the large increase in accuracy when using the `Localized` as opposed to the `Global` approach. Whereas `Global` completion requires approximately 60% of the entries to be known to achieve a relative error lower than 0.2, `Localized` completion only needs about 20% of the entries for the same accuracy.

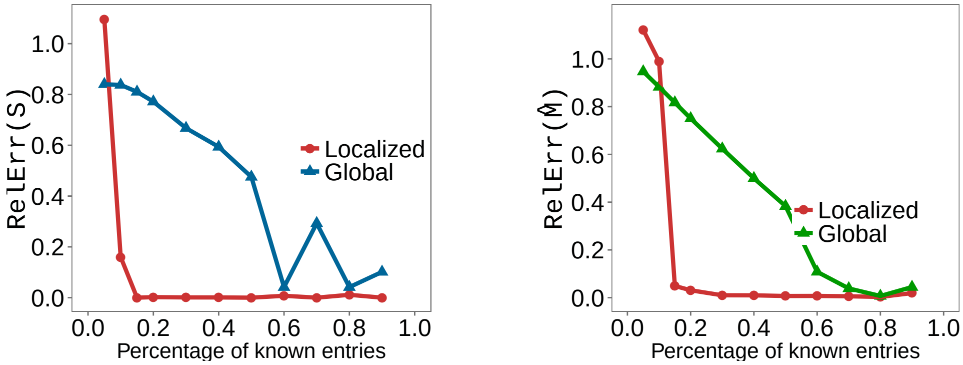


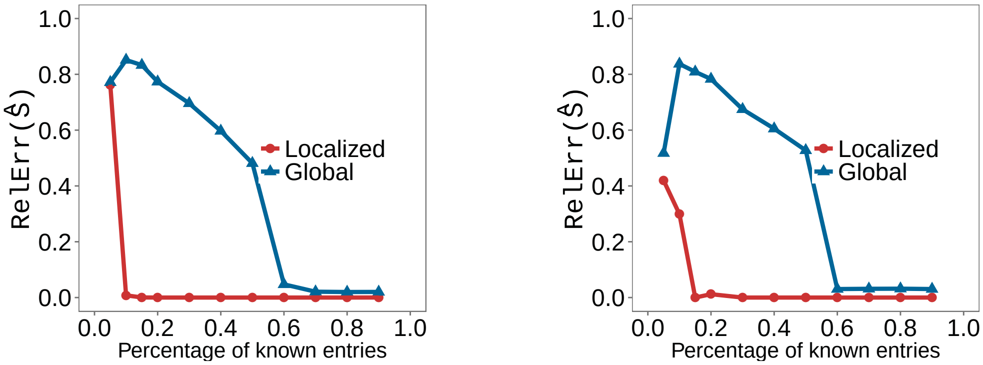
Figure 5·1: Relative error in completion on the synthetic data.

In Figure 5·2 we see the results of the same experiment except when $\mathbf{M}$ contains multiple low-rank submatrices, each of the same size and rank. The results are consistent for each submatrix; we observe that for each submatrix the `Localized` approach achieves a more accurate completion than `Global`, and the same is true for the whole matrix.

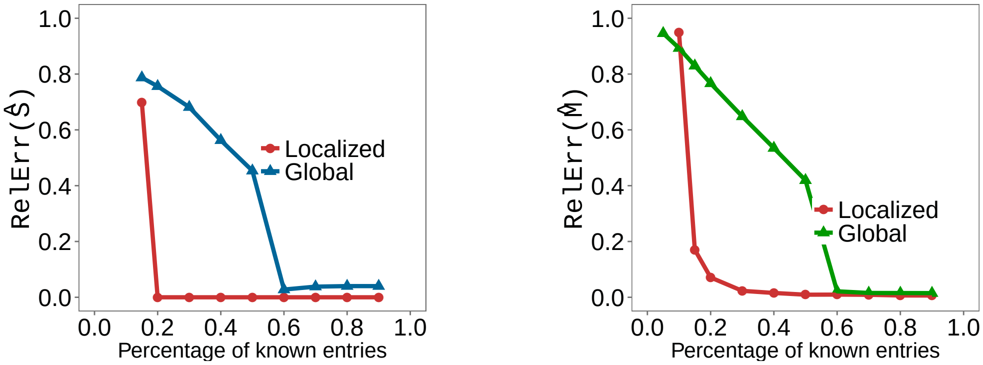
One way to explain this result is that the low-rank submatrix and its complement are deemed different datasets by the standard notion of a matrix in the completion literature. In the extreme, applying `Global` to the whole matrix is akin to fitting one model to the concatenation of two different datasets – since this requires two models, it results in error.

An observation we made in practice was that the success of the localized approach was partly due to the reliance of matrix completion on accurate rank-estimation. Techniques for estimating the rank of a partially-observed matrix, such as the ones in `LMaFit` and `OptSpace`, do not produce an accurate estimate when the matrix contains a low-rank submatrix; however, they do perform well when applied separately to each component (submatrix and the complement). In turn, this allows the algorithms to give a more accurate completion.

Running time evaluation: Despite the large increase in performance, one concern regarding the `Localized` approach maybe be running time, since the completion is



(a) Error over first **S** (b) Error over second **S**



(c) Error over third **S** (d) Error over the whole matrix.

Figure 5·2: Relative error in completion on a matrix with multiple low-rank submatrices.

run multiple times as opposed to once. However, `Localized` matrix completion turns out to be more efficient. The reason for this is that the completion is run on smaller matrices, but also because the rank estimation procedures execute faster. In fact, we observe the speedup with `Localized` ranges from 2X to 15X faster than `Global` depending on the percentage of known entries, with a speedup of approximately 6X at 0.2 the density at which `Localized` achieves an accurate completion.

Case study with traffic data: The dataset we use is a 3000×3000 partially observed Internet traffic matrix. Each row corresponds to a source AS, each column corresponds to a destination prefix, and each entry holds the volume of traffic that flowed from an AS to a prefix. The dataset is only partially observed with 70% of the entries missing. Hence, we repeat the synthetic experiments by hiding a portion of the 30% known entries and evaluating the error at each step.

In Figure 5·3 we see the same behavior as on synthetic data. The `Localized` approach achieves higher accuracy on the low-rank submatrix. The effect is less pronounced on the whole matrix $\mathbf{M}$. We observed that this was because the complement $\mathbf{T}$ of the submatrices was high rank, and `LMaFit` had lower accuracy on matrices with very high rank.

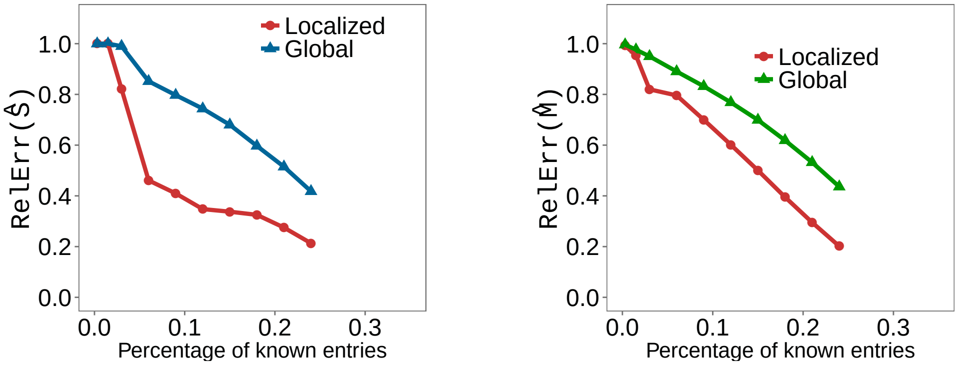


Figure 5·3: Relative error in completion for the Traffic dataset.

5.4.2 Evaluation of `SVP`

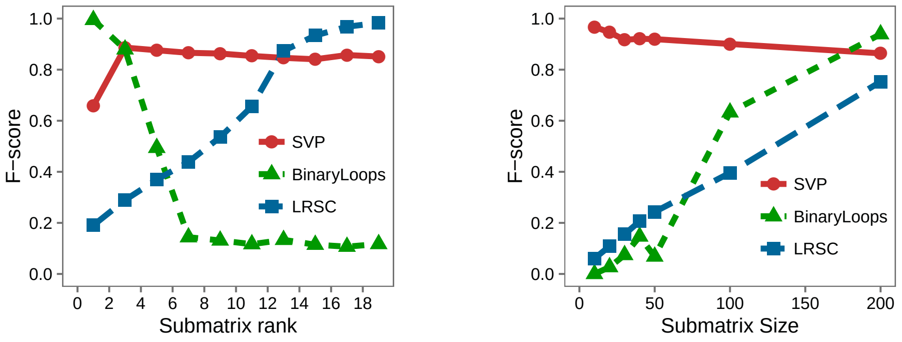
In the experiments above we have demonstrated the improvement in a `Localized` approach to matrix completion. We now isolate and analyze the algorithmic approach to step 1 – the task of finding a low-rank submatrix. Our results demonstrate that `SVP` is effective, efficient, and outperforms other heuristics for the same problem for a large variety of instances.

For experiments we set to rank of the matrix $\mathbf{M}$ to $r = 1000$ and the rank of the planted submatrix $\mathbf{S}$ to $\ell = 5$. Since the objective of our problem is to find the indices R_s and C_s, we evaluate the accuracy of our method using the combination of precision and recall to the standard F-Score$= 2\frac{Pr \times Rcl}{Pr + Rcl}$. F-score takes values in $[0, 1]$ and the higher its value the better.

We compare `SVP` against the algorithm in Rangan (2012) (which we call `BinaryLoops`) and algorithms for Subspace Clustering; the approaches are discussed in detail in Chapter 2. For Subspace Clustering we show results for `LRSC` by Vidal and Favaro (2014) since it offers the best balance of accuracy and efficiency; for `LRSC` we used the authors' original implementations. The baseline algorithms required minor modifications since none of them explicitly output the indices of a submatrix. For `BinaryLoops` we set $\gamma_{\text{row}} = \gamma_{\text{col}} = 0.75$, according to the author's recommendation. For `LRSC` we set $k = 2$ when clustering and select the output $\{R_s, C_s\}$ with the highest accuracy (though a user could pick the one with lowest empirical rank).

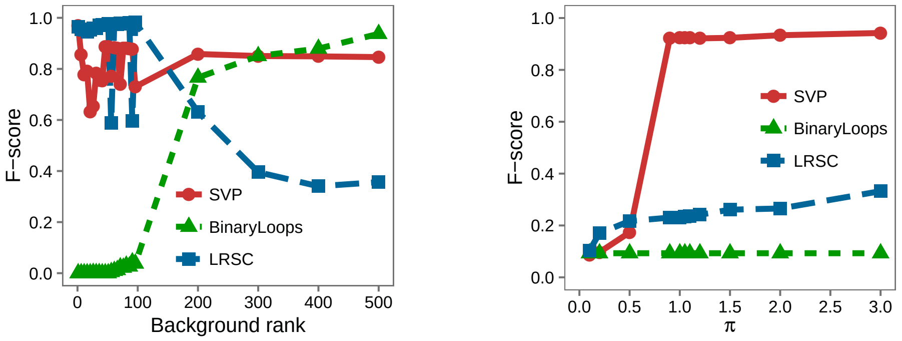
Varying size and rank: In the main set of experiments we examine a variety of problem instances by varying (a) the rank ℓ of $\mathbf{S}$ (b) the size of $\mathbf{S}$ (c) the rank r of the background matrix and (d) π. Figures 5·4a, 5·4b, 5·4c and 5·4d display the F-scores of the different algorithms in each case.

The results indicate that `SVP` accurately locates the low-rank submatrix $\mathbf{S}$ for the majority of instances, whether $\mathbf{S}$ is large or small. In line with our analysis, we see



(a) Rank ℓ of $\mathbf{S}$.

(b) Size of $\mathbf{S}$



(c) Background rank r.

(d) π

Figure 5·4: F-Score achieved by the `SVP`, `BinaryLoops`, and `LRSC` algorithms as: the rank ℓ of $\mathbf{S}$, the size of $\mathbf{S}$, the background rank r, or π vary.

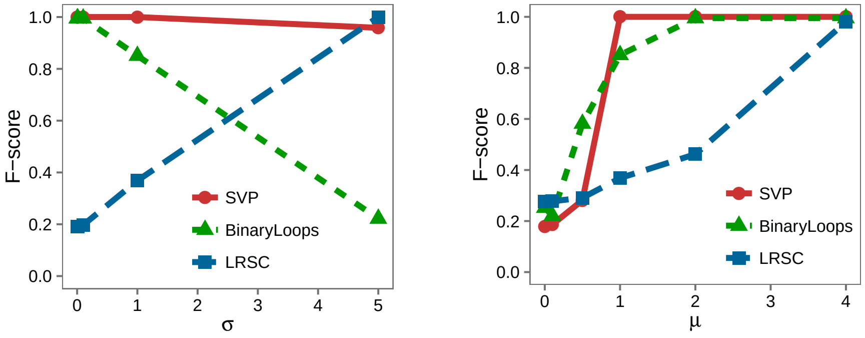
in Figure 5·4d that `SVP` succeeds precisely as π grows larger than one.

In contrast to the resilience observed in the performance of `SVP`, `LRSC` and `BinaryLoops` are more sensitive to changes in the problem instance. `LRSC` performs best when the rank and size of $\mathbf{S}$ are large, and the background rank is small. On the other hand, `BinaryLoops` performs best on instances where the rank of $\mathbf{S}$ is less than five, and the background rank is large. These behaviors are in agreement with the analysis and the design of these algorithms in the original papers where they were introduced. An interesting observation form Figures 5·4a and 5·4c is that `LRSC` and `BinaryLoops` are opposite in their behavior; each succeeds in complementary instances while `SVP` performs consistently well in all instances.

Different distributions: In this experiment, the goal is to test the resilience of the different algorithms for matrices generated from different distributions. We generate a matrix $\mathbf{M}$ with a planted low-rank submatrix $\mathbf{S}$ we describe in the setup at the beginning of Section 5.4. With the entries in $\mathbf{S}$ drawn from a Gaussian distribution, we vary the center of the distribution μ in $[0, 4]$ and the standard deviation σ in $[0.001, 5]$.

The results are shown in Figures 5·5a and 5·5b. Looking at Figures 5·5a we see the F-Score of each algorithm as a function σ as it varies from 0.001 to 5. Observe that `SVP` consistently achieves an F-Score close to 1 while `BinaryLoops` decreases as σ grows and `LRSC` increases as σ grows. The resilience of `SVP` to changes in σ can be attribute to the fact that $\pi > 1$ in each of these instances, and this is the region in which `SVP` succeeds according to our analysis. For small σ `BinaryLoops` achieves a high F-Score, but the perfromance degrades as σ grows because the $\pm$-sign pattern becomes more varied and `BinaryLoops` cannot distinguish between $\mathbf{S}$ and $\mathbf{T}$. In contrast, `LRSC` has exactly the opposite behavior; the F-Score starts out low and increases with σ. We attribute this to the experimental observation that `LRSC` is very dependent on $\delta = \text{mean}(|\mathbf{S}|) - \text{mean}(|\mathbf{T}|)$ (the difference in the average magnitude of values between $\mathbf{S}$ and $\mathbf{T}$).

Figure 5·5b shows the F-Score of each algorithm as a function of μ as it ranges from $\mu = 0$ (a centered distribution) to $\mu = 4$. For all algorithms we observe an increased accuracy as μ grows, though at different rates. As μ varies there is a point at which `SVP` increases from 0.28 to 0.98, this happens exactly for the combination of μ and σ for which $\pi > 1$. The explanation of `BinaryLoops` and `LRSC` is similar to that of Figure 5·5a. As μ increases `BinaryLoops` achieve higher F-Score because there is less variation in ±-pattern of entires in $\mathbf{S}$ which it exploits. As with σ, the value of δ grows with μ, making it easier it identify $\mathbf{S}$; for example, $\mu = 1$, the value of $\pi = 1.6$ and $\delta = 0.4$, while at $\mu = 4$ we have $\delta = 3.2$.



(a) Fix $\mu = 1$ are vary σ (b) Fix $\sigma = 1$ vary μ

Figure 5·5: F-Scores of `SVP`, `BinaryLoops`, and `LRSC` in robustness studies.In (a) $\mu = 1$ and σ varies and in (b) $\sigma = 1$ and μ varies.

Data with missing entries: To evaluate the methods we compare the F-Score achieved by `SVP` and it uses a given methods to find the first singular vectors. Figure 5·6 shows the results over four matrix instances that are representative of our experimental evaluation. What we observe is that not one of `Simple`, `EarlyStop`, or `Incremental` performs best over all instances. For this reason, we propose to use `Concensus`, which provides a common ground among the approaches for a small price in efficiency. Nevertheless, the performance of both `EarlyStop` and `Incremental` was observed to be high most of the time, and we recommend these if efficiency is a strong concern.

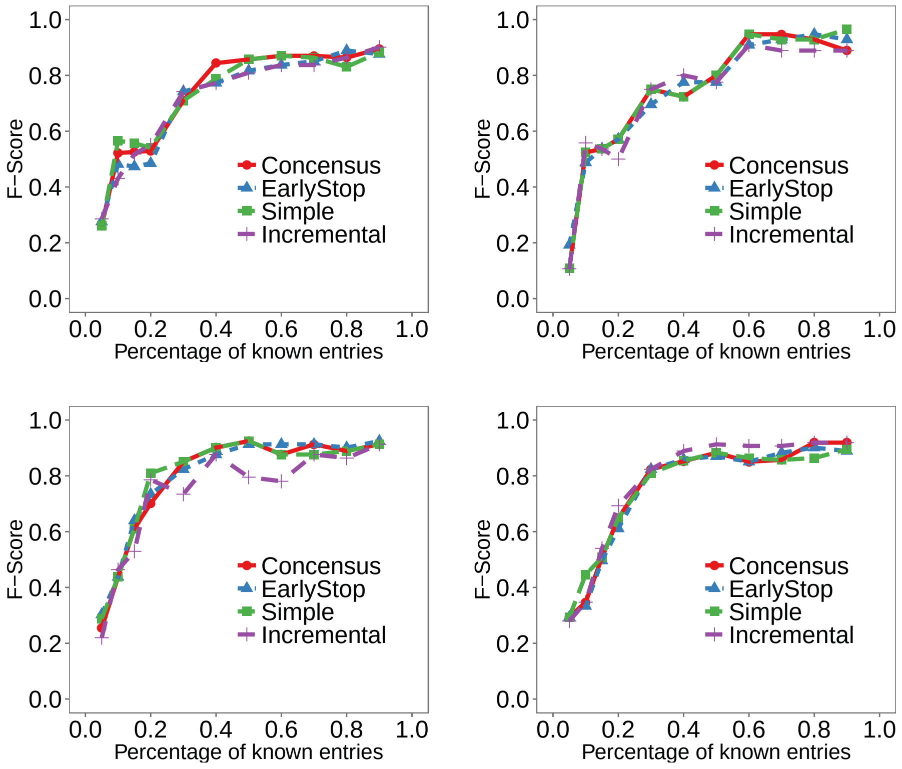
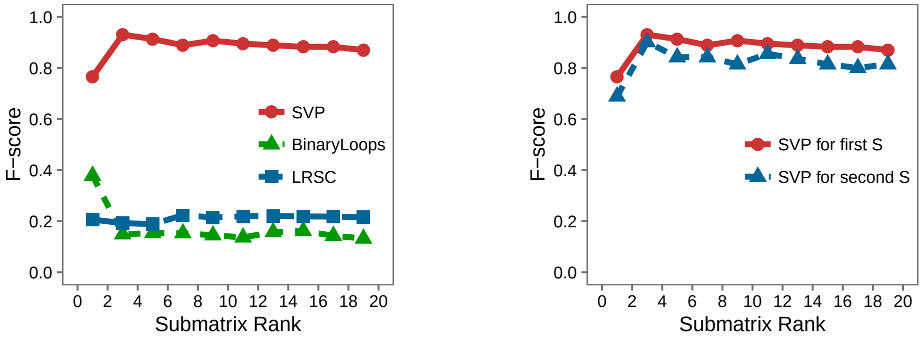


Figure 5·6: Comparison of algorithms for finding the singular vectors of an incomplete matrix in terms of the F-Score of `SVP` as a function of the percentage of known entries in the data.

Multiple submatrices: In this experiment our goal is to test whether the different algorithms are affected by the presence of multiple low-rank submatrices and are able to identify multiple low-rank submatrices. For this we generate $\mathbf{M}$ and $\mathbf{S}$ as described in the beginning of Section 5.4. In addition we plant a second submatrix $\mathbf{S}'$ of the same size and rank as $\mathbf{S}$ with $\pi = 1.2$ for each.

n	100	400	800	1000	2000	4000	6000
`SVP`	0.080	0.122	0.182	0.226	**0.745**	**2.665**	**11.168**
`BinaryLoops`	**0.004**	**0.024**	**0.111**	**0.201**	1.357	8.680	30.694
`LRSC`	0.092	0.441	1.483	2.191	9.561	83.587	310.8707

Table 5.1: The running time of `SVP`, `BinaryLoops`, and `LRSC` in seconds for an $n \times n$ input matrix $\mathbf{M}$.



(a) F-Score for one submatrix.

(b) `SVP` for two submatrices.

Figure 5·7: Left: F-Score of `SVP`, `BinaryLoops`, and `LRSC` on data with multiple low-rank submatrices as a function of the rank of the submatrices.

Figure 5·7a shows the F-Score of each algorithm for the task of finding one hidden submatrix (either $\mathbf{S}$ or $\mathbf{S}'$) as a function of the ranks of the submatrices. We observe that `SVP` significantly outperforms `LRSC` and `BinaryLoops` in finding a low-rank submatrix, and is unaffected by the presence of the second submatrix. In fact, when asked to report two submatrices, `SVP` can detect both $\mathbf{S}$ and $\mathbf{S}'$ with high accuracy. The F-Scores of this more complex task are shown in Figure 5·7b as a function ranks.

We note that when an algorithm reports subsets of rows and columns, we do not know whether this corresponds to $\mathbf{S}$ or $\mathbf{S}'$, hence we take the F-Score for whichever submatrix has larger intersection with the output indices.

Running time: We also compare the running times of `SVP`, `BinaryLoops` and `LRSC` as the size of the input matrix $\mathbf{M}$ increases. The running times of the different algorithms in seconds are shown in Table 5.1. From the table, we observe that the difference in the running times is mostly pronounced for large matrices. For those matrices `SVP` is the most efficient one followed by `BinaryLoops`. On the other hand, `LRSC`'s running time is 10 times larger than the running time of `BinaryLoops` and 30 times larger than the running time of `SVP`. Note that the running time of `SVP` is dominated by computing the first singular vectors of $\mathbf{M}$ and our implementation of `SVP` uses the off-the-shelf SVD decomposition of MatLab. In principle, we could improve this running time by using other SVD speedups and approximations, as in Drineas et al. (2006) for example.

Case study with gene-expression data: To further validate the usefulness of our approach we use a real-world yeast dataset described in Pavlidis and Grundy (2000), of size 2417×89 yeast dataset formed by micro-array expression data [8]. Each row i in the data represents a gene and each column j is an experiment; thus, entry (i, j) is the expression level of gene i in experiment j. For each gene, the dataset also provides a phylogenetic profile over 14 groups in the form of a binary vector for each gene.

We run `SVP` on the dataset and discover a 527×34 submatrix $\mathbf{S}$ with $\pi = 1.2$. By comparing the phylogenetic profiles (binary vectors), we found that the genes in $\mathbf{S}$ are similar. More specifically, when we compared the profiles of the genes in $\mathbf{S}$ with the genes not in the rows of $\mathbf{S}$, we found that the average Hamming distance between the profiles of $\mathbf{S}$ was 0.17 with a median of 0.2 and a standard deviation of 0.1, while the average Hamming distance among the rest of the genes was 0.35 and a median of 0.2 and a standard deviation of 0.2. Thus we conclude that the `SVP` helped isolate a subset of genes with similar profile patterns, and in the description of the dataset. According to Pavlidis and Grundy (2000), genes with similar profile patterns have similar behavior in biology.

---

[8] The data can be downloaded at http://mulan.sourceforge.net/datasets-mlc.html

Chapter 6

Conclusions

In this thesis, we addressed the limitations of traditional matrix completion in the initial assumptions on the problem instance and on the modeling of the ground truth matrix. In doing so, we made use of two sources of information in a partially observed matrix: the locations and the values of the observed entries.

To start, we drew a distinction between the traditional *statistical* matrix completion approaches and a new line of *structural* approaches. Using just the locations of known entries, structural matrix completion approaches explicitly analyze the information in the partially observed matrix to quantify the possibility of completion: whether the number of completions is infinite, finite, one, or none. In our first contribution, we constructed the first organizing framework of structural completion. Using this framework we give insight into the relationships between the different structural approaches and when they can be expected to succeed in practice.

In the case that structural approaches cannot succeed, our second contribution introduced the active matrix completion problem: which small number of unknown entries can be uncovered so that accurate matrix completion is possible? We developed active approaches for each structural matrix completion algorithm laid out in our framework. In particular, we expand one active approach into an active-completion algorithm, `Order&Extend`, which minimizes the number of additional queries and the error in the completion.

Finally, motivated by observations in real data we challenged the traditional *global* low-rank assumption. We demonstrated that traditional matrix completion algorithms cannot accurately estimate both low-rank submatrices and the rest of the data at once, and developed an algorithm targeted these components separately. In doing so, our first contribution was an algorithm that uses the first singular vectors to find low-rank submatrices in fully- and partially-known matrices. Next, we proposed a *localized* to matrix completion that first discovers low-rank submatrices, then completes them separately, attaining a more accurate completion.

There are two open directions which we feel are most interesting and valuable. First, is the study of finite completability. Although guaranteeing that a partially observed matrix has a unique completion gives high confidence in the accuracy, perhaps it is too strict. In practice, achieving a finite number of completions may be enough as long as this finite number is small. Hence, an open direction is the study of the finiteness of completion, and how entries can be added to guarantee a small number of possible completions.

The second direction is regarding rank estimation. To the best of our knowledge, there is no single approach that is known to in general give an accurate estimate of the rank of matrix based on its partially-observed instance. Hence, the establishment of such an algorithm together with an analysis of when it can be used reliably is a valuable direction. Further, based on such an analysis, the active version of the problem would be of high interest: which queries should be made on a partially observed matrix to accurately estimate the rank of the true underlying matrix?

Overall, while the problem of matrix completion draws a low of attention, there remains a gap between literature and practice. In general, there is either a lack of analysis on when the algorithms can be used, or real data is too sparse and structured for the algorithms to be applicable. The algorithms proposed in this thesis make the first steps at bridging this gap, but there remains work to be done.

Appendix A

A.1 Background on graph rigidity

The theory of graph rigidity revolves around analyzing the amount of movement possible in a particular graph. Suppose we are given n points in a k-dimensional space and the distance between *some* pairs of points. Considering each dimension as a coordinate in space, a central question in graph rigidity is: *are the coordinates of the points uniquely determined according to the known distance?* In other words, is it possible to assign different coordinates to some points and still maintain the specified distances?

This problem is easily posed in terms of an undirected graph of nodes and edges $G = (V, E)$, and a *configuration* which is a mapping of the nodes to k-dimensional points ($p : V \to \mathbb{R}^k$). Together, a graph and configuration make up a *framework* (G, p) where edges capture the subset of specified distances d_{ij}. Graph rigidity studies whether it is possible to move nodes in the graph while preserving the edge lengths, or even while preserving the distances between *all* nodes implied by the configuration.

Let us now provide several examples to illustrate these ideas. First, consider a triangle as in Figure A·1 in 2D spaces, i.e. a Cartesian planeb. There are three nodes $V = \{1, 2, 3\}$, and three edges $E = \{(1, 2), (2, 3), (3, 1)\}$ that specify and fix the distances d_{12}, d_{23}, d_{31}. Considering Figure A·1 we ask: can any single node be moved in a non-trivial way without moving the other two and without violating the distance constraints? The answer is no. Aside from rotation and translation which are *trivial* motions, any movement will alter the distances specified by the edges. Since no movement is possible, we say that the triangle structure is *rigid* in 2-dimensional space.

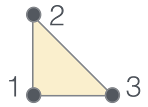
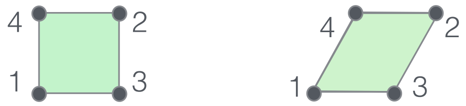


Figure A·1: A rigid structure.

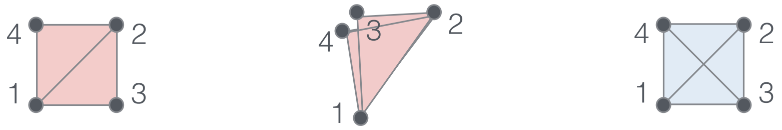
On the other hand consider a square as in Figure A·2a. Imagine pulling node-2 to the right; such a movement will transform the square into the parallelogram in Figure A·2b and change the coordinates of the nodes without changing any of the edge-lengths. In particular this motion, called an *infinitesimal motion*, maintains the distances on the edges but changes the distance between nodes 3 and 4; when this type of continuous motion is possible the structure is labeled as *flexible*.

Consider adding a single edge between nodes 1 and 2 as in Figure A·2c. On one hand, there is no continuous movement along the x or y axis that will maintain the distances along the edges. On the other hand, imagine the shape formed by folding Figure A·2c along the hinge-edge between nodes 1 and 2; this shape has the same distances between adjacent nodes but a different distance between nodes 3 and 4. When no motion is possible, but there are multiple shapes consistent with the specified distances, the structure is called *locally rigid*.

Now suppose we were to add two diagonal edges to the square as in Figure A·2e.



(a) Flexible (b) Nontrivial motion



(c) Locally Rigid (d) Consistent shape (e) Globally Rigid

With these edges there is no possible movement that maintains the distances, and any shape with the same distances between adjacent nodes will have the same distance between all nodes. This structure, along with the triangle in Figure A·1, is caller *globally rigid*.

Characterizing the rigidity of a structure requires analyzing whether a continuous motion is possible to transform one framework (G, p) to another framework (G, q), or in terms of our example from one shape to another. It can be show that any such motion over time t that preserves the distances must satisfy $\frac{d}{dt}\|p_i - p_j\|^2 = 0$ for all edges $(i, j) \in E$. Differentiating, this translates to an equation of the form $(p_i - p_j)(\check{p}_i - \check{p}_j) = 0$ for each edge, where $\check{p}_i$ is the instantaneous velocity of point p_i and is unknown. Over n_v points and n_e edges, the equations can be represented with an $n_e \times kn_v$ matrix called the *rigidity matrix.* To evaluate the rigidity of a framework it suffices to study the rank and null space of this rigidity matrix, which can be constructed using random realizations of the points p_i (the proof and reasoning are beyond the scope of this article). While various conditions exist for determining flexibility or local rigidity, characterizing global rigidity for $k > 2$ is unsolved. We refer the curious reader to Jackson (2007), Singer and Cucuringu (2010), Jackson (2007) for a more thorough explanation.

We refer the reader to Appendix A for an a high level overview of graph rigidity. Singer and Cucuringu (2010) were the first to draw the connection between graph rigidity and matrix completion. Recall that matrix completion takes as input a partially observed $n \times m$ matrix $\mathbf{M}_\Omega$ with known entries in only a subset of locations specified by the mask Ω. The partially observed matrix can be represented as a *mask* graph $G_\Omega = (E, V)$ where each row and column corresponds to a node, and each known entry $(i, j) \in \Omega$ corresponds to an edge between node i and node J. Note that $n_v = |V| = n + m$ and $n_e = |E| = |\Omega|$ Singer and Cucuringu (2010) showed that a characterization of the rigidity of G_Ω is indicative of the number of possible completions of a partially observed matrix $\mathbf{M}_\Omega$. In fact, the analysis is with regards to the completability of a mask as opposed to a partially observed matrix, but as we have shown, the two imply each other and we use them interchangeably.

We carry the thread of the visual examples. Consider the red square as a mask graph G_{Ω_1} in Figure A·2c and its folded version G_{Ω_2} in Figure A·2d to be graphs representing a partially observed matrices $\mathbf{M}_\Omega$. Recall that each edge (i, j) corresponds to a value $\mathbf{M}(i, j)$. Since the two graphs have exactly the same set of specified distances (any edge that is in G_{Ω_1} is also in G_{Ω_2} with the same distance and vice versa), the two graphs encode the same partially-observe matrix. The missing entries in $\mathbf{M}_\Omega$ correspond to pairs of nodes whose distances are not specified, in our example since the edge $(3, 4)$ is not in the graph, $\mathbf{M}(3, 4)$ is not known in $\mathbf{M}_\Omega$. Notice that Figure A·2c and Figure A·2d disagree on the distance between nodes 3 and 4. This implies that for the same set of observed entries, there are (at least) two possible values for the missing entries, resulting in multiple possible completions on $\mathbf{M}_\Omega$. Local rigidity of a mask implies that its corresponding matrix is *locally completable* and has a finite number of completions. On the other hand, for a partially observed $\mathbf{M}$ whose corresponding graph is globally rigid there is only a single possible completion, and the matrix is said to be *uniquely completable*; As you may have guessed, a flexible framework corresponds to an infinite number of completion. We highlight that the classification of the number of possible completions of a partially observed matrix uses only the locations of known entries and not their values; in other words the shape in Figure A·2c is locally rigid not matter the distances on the edges, and its corresponding $\mathbf{M}_\Omega$ is locally completable regardless of the specific values in the observed entries.

References

Aggarwal, C. C. (2016). *Recommender Systems*. Springer.

Alon, N., Krivelevich, M., and Sudakov, B. (1998). Finding a large hidden clique in a random graph. *Random Structures and Algorithms*, 13(3-4):457–466.

Baker, C. (2012). Incremental svd package. `http://www.math.fsu.edu/~cbaker/IncPACK/`.

Bentbib, A. and Kanber, A. (2015). Block power method for svd decomposition. *Analele Stiintifice Ale Universitatii Ovidius Constanta-seria Matematica*, 23(2):45–58.

Bhojanapalli, S. and Jain, P. (2014). Universal Matrix Completion. *ArXiv e-prints*.

Candès, E. J. and Recht, B. (2012). Exact matrix completion via convex optimization. *Communications ACM*, 55(6):111–119.

Candès, E. J. and Tao, T. (2010). The power of convex relaxation: near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080.

Chakraborty, S., Zhou, J., Balasubramanian, V. N., Panchanathan, S., Davidson, I., and Ye, J. (2013). Active matrix completion. In *ICDM*.

Chen, Y., Bhojanapalli, S., Sanghavi, S., and Ward, R. (2014). Coherent matrix completion. In *International Conference on Machine Learning (ICML)*.

Drineas, P., Kannan, R., and Mahoney, M. W. (2006). Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183.

Elhamifar, E. and Vidal, R. (2009). Sparse subspace clustering. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2790–2797. IEEE.

Funk, S. (2006). Incremental svd for the netflix prize. `http://sifter.org/~simon/journal/20061211.html`.

Golub, G. H. and Loan, C. F. V. (2012). *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences.

Guillaume, J.-L. and Latapy, M. (2006). Bipartite graphs as models of complex networks. *Physica A: Statistical Mechanics and its Applications*, 371(2):795 – 813. Software available at `http://jlguillaume.free.fr/www/programs.php`.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53.

Iam-on, N. and Garrett, S. (2010). Linkclue: A matlab package for link-based cluster ensembles. *Journal of Statistical Software*, 36(1):1–36.

Jackson, B. (2007). Notes on the rigidity of graphs.

Jain, P., Netrapalli, P., and Sanghavi, S. (2013). Low-rank matrix completion using alternating minimization. In *Symposium on Theory of Computing (STOC)*, pages 665–674.

Jain, P. and Oh, S. (2014). Provable tensor factorization with missing data. In *Advances in Neural Information Processing Systems*, pages 1431–1439.

Johnson, C. R. (1990). Matrix completion problems: a survey. In *Matrix theory and applications*, volume 40, pages 171–198. Providence, RI.

Kahle, T., Kubjas, K., Kummer, M., and Rosen, Z. (2016). The geometry of rank-one tensor completion. *arXiv preprint arXiv:1605.01678.*

Keshavan, R. H., Montanari, A., and Oh, S. (2010a). Matrix completion from a few entries. *IEEE Transactions on Information Theory*, 56(6):2980–2998.

Keshavan, R. H., Montanari, A., and Oh, S. (2010b). Matrix completion from noisy entries. *Journal of Machine Learning Research*, 11:2057–2078.

Király, F. and Tomioka, R. (2012a). A combinatorial algebraic approach for the identifiability of low-rank matrix completion. *arXiv preprint arXiv:1206.6470.*

Király, F. J., Theran, L., Tomioka, R., and Uno, T. (2013). The algebraic combinatorial approach for low-rank matrix completion. *Computing Research Repository.*

Király, F. J. and Tomioka, R. (2012b). A combinatorial algebraic approach for the identifiability of low-rank matrix completion. In *International Conference on Machine Learning (ICML).*

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review*, 51(3):455–500.

Kwok, Q. Y. J. T. (2015). Colorization by patch-based local low-rank matrix completion.

Lee, J., Kim, S., Lebanon, G., and Singer, Y. (2013). Local low-rank matrix approximation. In *Proceedings of The 30th International Conference on Machine Learning*, pages 82–90.

Lee, S.-g. and Seol, H.-g. (2001). A survey on the matrix completion problem. *Trends in Mathematics*, 4(1):38–43.

Matula, D. W. and Beck, L. L. (1983). Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427.

Meka, R., Jain, P., and Dhillon, I. S. (2009). Matrix completion from power-law distributed samples. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1258–1266.

Negahban, S. and Wainwright, M. J. (2012). Restricted strong convexity and weighted matrix completion: Optimal bounds with noise. *Journal of Machine Learning Research*.

Pavlidis, P. and Grundy, W. N. (2000). Combining microarray expression data and phylogenetic profiles to learn gene functional categories using support vector machines.

Rangan, A. V. (2012). A simple filter for detecting low-rank submatrices. *Journal of Computational Physics*, 231(7):2682–2690.

Roth, B. (1981). Rigid and flexible frameworks. *Amer. Math. Monthly*.

Ruchansky, N., Crovella, M., and Terzi, E. (2015). Matrix completion with queries. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1025–1034. ACM.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*, pages 285–295. ACM.

Signoretto, M., Dinh, Q. T., De Lathauwer, L., and Suykens, J. A. (2014). Learning with tensors: a framework based on convex optimization and spectral regularization. *Machine Learning*, 94(3):303–351.

Singer, A. and Cucuringu, M. (2010). Uniqueness of low-rank matrix completion by rigidity theory. *SIAM J. Matrix Analysis Applications*, 31(4):1621–1641.

Sutherland, D. J., Póczos, B., and Schneider, J. (2013). Active learning and search on low-rank matrices. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Trefethen, L. N. and Bau III, D. (1997). *Numerical linear algebra*. SIAM Press.

Vidal, R. and Favaro, P. (2014). Low rank subspace clustering (lrsc). *Pattern Recognition Letters*, 43:47–61.

Wen, Z., Yin, W., and Zhang, Y. (2012). Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm. *Math. Program. Comput.*

CURRICULUM VITAE

CONTACT INFORMATION

natalir@bu.edu
Boston University, Department of Computer Science
111 Cummington Mall, Boston MA 02215

EDUCATION

PhD, Computer Science: *Data Mining* 2011-2016
Boston University, Boston MA

B.S *Mathematics and Computer Science* 2008-2011
Boston University, Boston MA *Magna Cum Laude*

LANGUAGES

English, Russian.

RESEARCH EXPERIENCE

Research Intern May-August 2014
Yahoo! Labs, Barcelona,
Supervisor: Francesco Bonchi

Research Assistant September 2011 - Present
Data Management Group,Boston University.
Supervisors: Evimaria Terzi, and Mark Crovella

Research Intern May-August 2010
Center for Embedded Networked Sensing (CENS),
UCLA, University of California Los Angeles
Supervisors: Nabil Hajj Chehade, Greg Pottie.

Research Intern January-May 2010
Security Group, Boston University.
Supervisors: Leo Reyzin.

Research Assistant 2005-2009
DeCaprio Medical Oncology Lab,
Dana Farber Cancer Institute
Supervisors: Larisa Litovchick.

Refereed Conference Publications

1. **N.Ruchansky**, M.Crovella, and E.Terzi.
   "Matrix Completion with Queries."
   KDD 2015. Sydney, Australia.

2. **N.Ruchansky**, F.Bonchi, D.Garcia-Soriano, F.Gullo, and N.Kourtellis.
   "The Minimum Wiener Connector."
   SIGMOD 2015. Melbourne, Australia.

3. **N.Ruchansky** and D.Proserpio.
   " A (Not) NICE way to verify the OpenFlow Switch Specification."
   SIGCOMM 2013 (Poster). Hong Kong, China.

4. G.Gursun, **N.Ruchansky**, E.Terzi and M.Crovella
   "Routing State Distance: A Path-based Metric for Network Analysis."
   IMC 2012. Boston, USA.
   2013 Applied Networking Research Prize, awarded by the IETF and IRTF.

5. G.Gursun, **N.Ruchansky**, E.Terzi and M.Crovella
   "Inferring Visibility: Who's (not) Talking to Whom?"
   SIGCOMM 2012. Helsinki, Finland. and TinyToCS Vol 1

6. **N.Ruchansky**, E.Do, C.Lochner, T.Rawls, N.Chehade, J.Chien, G.Pottie, W.Kaiser. *"Monitoring Workspace Activities Using Accelerometers."*
   ICASSP 2011. Prague, Czech Republic.

Teaching Experience

Guest Lecturer Fall 2015
Cracking the Coding Interview workshop series

Teaching Assistant Fall 2014,Summer I and II 2012
Data Mining, Intro to Computer Science

Private Tutor 2010-2014
Math and Computer Science

Math Teacher Fall Semester 2011
YearUp, Boston MA.

Teaching Assistant for Scratch Fall 2010, Spring 2011
Tufts University DevTech.

Teachers Aid January-May 2008
Longfellow Children's Center, Sudbury MA.

WORK EXPERIENCE

Intern Zigelbaum + Coelho LLC.	May-August 2011
Grader Computer Science, Boston University.	Fall 2010, Spring/Fall 2011
Cashier/Waitress Boston University Shelton Hall Late Night.	2008-2009

AWARDS

Conference Travel
- KDD 2016
- SIGMOD 2015
- ACM-W Scholarship 2011 and 2013
- SIGCOMM 2012

Academic
- Computer Science GAANN Fellowship 2011-2014
- Transition Limited Consulting Scholarship.
- College Scholar (top 5% of the College of Art and Sciences Class of 2011)

SERVICE

Engagement of Girls and Women in CS
- Guest lecturing with the Boston University Artemis Project
- Voluntering at events sponsored by DotDiva, WISE, JumpStart

Mentoring and Outreach
- SeniorNet at Boston University
- Computer Science mentor at majors fairs.
- Haley House Cafe and Harvard Square Homeless Shelter

ACADEMIC SERVICE

External Reviewer/Reviewer:

- IJCAI,VLDB,NIPS,ICDM	(2016)
- WSDM, KDD	(2015)
- WSDM, SDM, AAAI	(2014)
- EDBT, WWW, SIGMOD, ICDM	(2013)
- PKDD, ICDE, WSDM	(2012)

Departmental:
- Organizer of Data Management Seminar
- Founder and Organizer of the Graduate Student Seminar
- Representative at prospective-student events.